# A Dataflow Approach To Efficient Change Detection of HTML/XML Documents in WebVigiL[*]

Anoop Sanka, Shravan Chamakura, Sharma Chakravarthy
Department of Computer Science & Engineering
The University of Texas at Arlington
{asanka,chamakur,sharma}@cse.uta.edu

## Abstract

The burgeoning data on the Web makes it difficult for one to keep track of the changes that constantly occur to specific information of interest. Currently, the most widespread way of detecting changes occurring to Web content is to manually retrieve the pages of interest and check them for changes. This mode of action not only wastes useful resources, but also presents information that may not be relevant to the given context.

In this paper, we present a change-monitoring system – WebVigiL – which efficiently monitors user-specified web pages for customized changes and notifies the user in a timely manner. We present the dataflow approach used for detecting multiple types of changes to a page. This approach has been optimized to group similar/same specifications to reduce the computation of changes. Multiple changes to the same page can also be handled in our approach. We also provide the overall architecture of WebVigiL to highlight role of change detection graph (CDG) which forms the core of the WebVigiL system.

## 1 Introduction

The world wide web has outdone traditional media such as television, to become an indispensable source of information. There is data for everyone and everything on the Web, and this data is increasing at a rapid pace. This plethora of information often leads to situations wherein users looking for specific pieces of information are flooded with irrelevant data. For example, an information technology professional who is only interested in news related to his areas of interest, is flooded with news on many topics when he goes to any popular news website. There are other situations in which users are interested in knowing about the updates happening to a particular web page of their interest. For example, students might want to know about the updates that are made to their course websites regarding any new projects. In other scenarios there are large software development projects where there are number of documents, such as require-

[*]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).

ments analysis, design specification, detailed design document and implementation documents. 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. In general, the ability to specify changes to arbitrary documents and get notified in different ways will be useful for reducing the wasteful navigation. The proposed architecture also provides a powerful way to disseminate information efficiently without sending unnecessary or irrelevant information.

Today, information retrieval is mostly done using the pull paradigm where the information of interest is pulled from its source by the user giving a query (or queries). The user takes the burden of analyzing the pulled information for any changes of interest. Although there are approaches, such as mailing lists, to notify users of the changes that happen to information of interest, there is little or no scope for the user to customize those notifications. The user has to be satisfied with whatever information the source wishes to send rather than what specific information the user wants.

A great deal of research has been done in the field of active technology to provide the capability of timely responses for many applications. Event-Condition-Action (or ECA) rules are used to provide active capability to a system. In the case of large-scale distributed environments such as the Web, users are interested in monitoring changes to a particular web page. But there are instances in which the change detection is required at a finer granularity, such as specifying changes to links, images, phrases or keywords in a page. Web pages that are monitored for detecting changes may be either HTML or XML pages. Changes to pages and changes to images, links, keywords and etc. correspond to primitive events when mapped to the ECA paradigm and their combinations form composite events. Thus, some of the techniques developed for active databases, when extended appropriately, will provide a solution to detect changes to web pages. This paper focuses on developing a framework and to provide a selective propagation approach to detect changes that are of interest to the users in the context of web and other large-scale network-centric environments by adapting and extending the existing active technology.

The remainder of the paper is organized as follows. In section 2 we present the related work that has been done in the field of change detection of web content. In section 3 we present the architecture of the WebVigiL system. In section 4 we show how the ECA paradigm is incorporated into the system. In section 5 we present the core component of the WebVigiL system which is the Change Detection Graph (CDG). Finally, we discuss future work in section 6.

## 2 Related Work

Many tools have been developed and are currently available for tracking changes to web pages. **AIDE** (AT&T Internet Difference Engine) [4], developed by AT&T 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, tables, lists or phrases.

**NetMind** [9] formerly known as URL-minder provides keyword or text-based change detection and notification service over web pages. Net-

Mind detects changes to links, images, keywords and phrases in an HTML page. The media of notification are e-mail or mobile phone. The system lacks the support to specify ignoring changes which the user is not interested in. Also, it lacks the support to specify composite changes (when both links AND images change) on a page. Also there is no provision for the user to come back later and view the last changes that have been detected. The frequency of when to poll the page is predefined.

**WebCQ** [8] is a prototype system for large-scale web information monitoring and delivery, which makes use of the structure present in hypertext and the concept of continuous queries. WebCQ is designed to discover and detect changes to the web pages and to provide a personalized notification of the changes to the users. User's monitoring requests are modelled as continuous queries on the web. The authors specify that composite changes can be detected but , currently the system does not seem to support them. WebCQ lacks a fine grouping strategy which results in the change being computed more than once for two users having the same type of request. The system only supports HTML documents and the frequency of fetching is at the level of day which does not prove good for situations which require a short frequency of fetching. Also there is no provision to specify new requests based on old requests.

**WYSIGOT** [11]is a commercial application that can be used to detect changes to HTML pages. This system has to be installed on the local machine, which is not always possible. The system has the feature to monitor an HTML page and also all the pages that it points to. But the granularity of change detection is at the page level.

Some of the salient features of WebVigiL when compared with the above systems are:

1. *Properties of monitoring requests can be inherited*: The user has the option of specifying the monitoring request to be dependent on the status of other monitoring requests. One can specify the start/end of a request to be the start/end of another request.

2. *Flexible specification of versions*: All the above systems compute changes between two successive pages. In WebVigiL, the user can explicitly specify the pages that can participate in change detection.

3. *Composite change detection*: WebVigiL provides an elegant way to specify multiple change types, such as 'changes occurring to either images or links', which none of the above systems provide.

## 3 WebVigiL Architecture

WebVigiL is a change-detection and notification system, which can monitor and detect changes to unstructured documents in general. 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 1 summarizes the complete architecture of WebVigiL. The functionality of each module is described briefly in the following sections.

### 3.1 Sentinel

WebVigiL provides an expressive language with well-defined semantics for specifying the monitoring requirements pertaining to the Web. Each monitoring request is termed a Sentinel. A Sentinel encompasses the following: the target URL, the change type desired (which can be links, images, phrases, keywords or a combination of these using the OR, AND, NOT operators), specifying a fetch frequency if known or leaving it to the system to adapt to the changes, what versions of fetched pages to compare changes (pairwise, every n or moving n) and the change notification mode (e-mail, PDA, fax). A Sentinel can also inherit the properties of previously defined sentinels and depend on their life-cycles.

For example, the *Sentinel* created for a request to monitor *http://www.cnn.com* for any updates related to *Iraq*, for a *period of 2 years starting from now*, would be mapped as follows:

*Create Sentinel s1 on the URL http://www.cnn.com*

*Monitor for keywords "Iraq", Fetch using 'Best effort'*

*From NOW to NOW + 2 Years*

*Notify by e-mail to user@uta.edu every 4th day*

*Compare alternate versions of fetched pages.*

A detailed explanation is given in [7].

## 3.2 Verification Module

This module processes user requests for syntactic and semantic correctness. Valid sentinels are populated in the Knowledge base (currently, Oracle) and a notification of the valid sentinels is sent to the change detection module. Briefly, the main functions of this module are: load balancing and syntactic validation between client and server and semantic validation of sentinels at the server, as the dependency information specified in the sentinel is available at the server.

## 3.3 Knowledge Base

Knowledge base is a persistent repository containing meta-data about each user, count and details of the sentinels set by them. The details of the sentinels have to be stored on a persistent medium for the purposes of various modules and also to provide recovery to a stable state in case of system failure. All the modules in the system interact with the Knowledge base for their proper functioning.

## 3.4 Change Detection Module

Every valid user request arriving at WebVigiL initiates a series of operations that occur at different points in time. Some of these operations 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 sentinels. This module generates ECA [1] rules to perform the actions of: activating and deactivating sentinels, constructing and maintaining Change Detection Graph and generating fetch rules. Change detection algorithms that give the difference between two HTML/XML pages have been developed [7] [10].

## 3.5 Fetch Module

The fetch module [3] is responsible for retrieving the pages registered with it and thus serves as a local wrapper for the task of fetching pages depending upon the user-set fetching policy. This module informs the version controller of every version it fetches, stores it in the page repository and notifies the CDG of a successful fetch. The wrapper fetches a page only when its properties indicate that a change has occurred. These properties are: the last-modified time for static web pages and checksum for dynamic web pages.

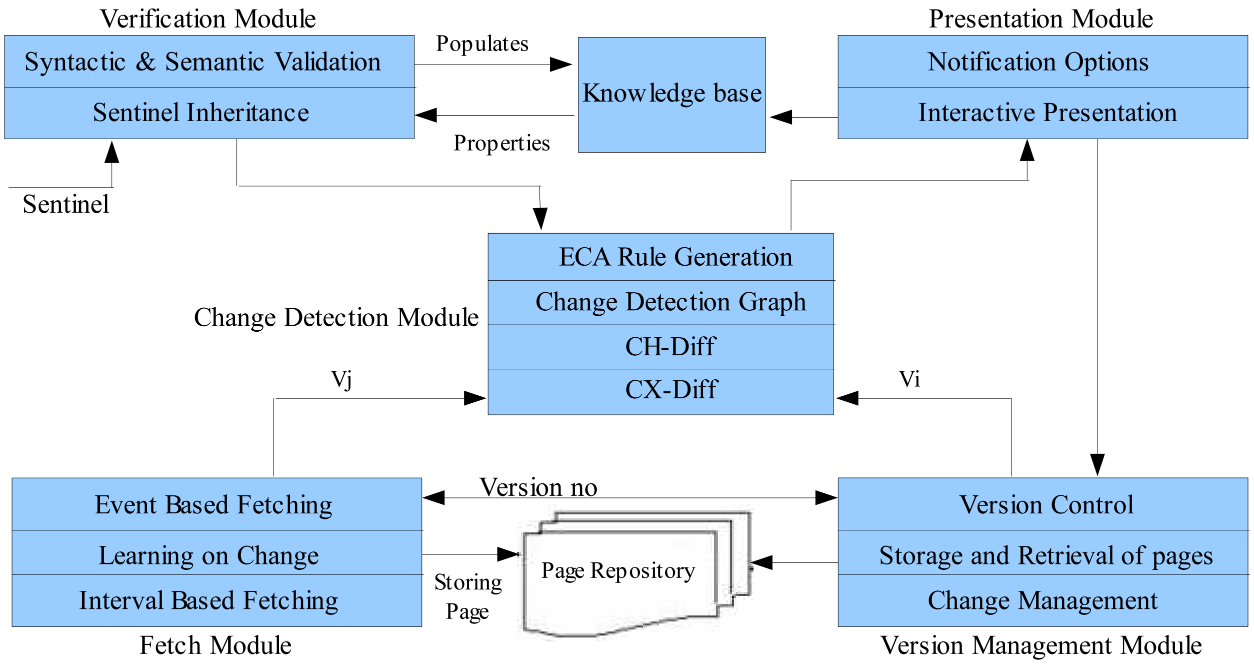


Figure 1: WebVigiL Architecture

### 3.6 Version Management

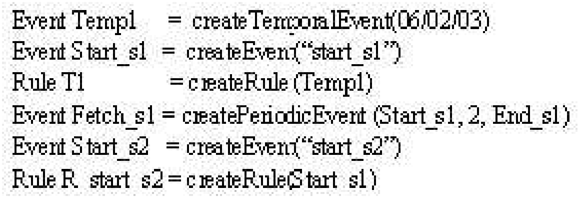
An important feature of WebVigiL architecture is its server-based repository service, which archives and manages different versions of pages. The primary purpose of the repository service is to reduce the network traffic by reducing the number of network connections made to the remote server. When a request is made for a page, the version management checks for the page in its cache and returns it if it is present. Otherwise, the page is fetched and stored for future requests.

### 3.7 Presentation Module

The primary functionality of this module is to clearly present the detected changes in a legible manner. The different ways in which changes can be displayed are: merging two documents with the changes highlighted, displaying only the changes in a single page and highlighting the differences in both the pages side-by-side. Notification of changes can be done according to the user-specified frequency or whenever a change is detected.

## 4 Activation & Deactivation of ECA rules

WebVigiL uses the Java Local Event Detector(LED) to incorporate active capability in the system. LED is a library designed to provide support for primitive and composite events, and rules in Java applications in a *seamless* manner.



```
Event Templ     = createTemporalEvent(06/02/03)
Event Start_s1  = createEvent("start_s1")
Rule T1         = createRule (Templ)
Event Fetch_s1 = createPeriodicEvent (Start_s1, 2, End_s1)
Event Start_s2  = createEvent("start_s2")
Rule R start s2 = createRule(Start s1)
```

Figure 2: Event Generation

Primitive and composite event detection in various parameter contexts and coupling modes has been implemented.

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 default, a sentinel is enabled during its lifespan. The user can also explicitly change the state of the sentinel during its lifespan. The start/end of a sentinel can be time points or events. 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 the event of interest has occurred. In WebVigiL, the ECA rule generation module creates the appropriate events and rules to enable/disable sentinels. We achieve this as follows. Consider the scenario where S1 is defined in the interval *[06/02/04, 01/02/05]*. At time *06/02/04* sentinel S1 has to be enabled. Figure 2 shows the events and rules that are generated to enable sentinel S1.

Fetch_S1 is a periodic 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 page 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 and also initiates 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 events and rules are generated in order to enable S2 are shown in Figure 2. Here, we are associating the rule R_start_S2 with 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.

## 5 Change Detection Graph

The assumption while developing the WebVigiL system has been that even though there will be requests for different change types on different URLs, there will be overlaps among URLs, types of changes, frequency of access, etc. One of the goals of WebVigiL is to process sentinels efficiently and be able to scale to a very large number of sentinels. A very naïve approach for change detection is to maintain a hash table with the pages of interest as the keys and the values being the list of sentinels monitoring that page. When a page is fetched, the sentinel(s) on that page are extracted and change type is detected for *each* sentinel. This approach is not efficient as it results in redundant change calculations if more than one sentinel is interested on the same page for the same change-type. To remove the redundant calculations, a different approach can be employed wherein the sentinels in the hashtable are grouped based on the change-type. This means that the values present in the hashtable will contain a list of sentinel lists which are grouped on the same change-type. This results in some efficiency, but it becomes difficult to implement composite change types (like images AND links) using this approach.

We need a data structure that will allow us to asynchronously feed fetched pages for change detection, allow parallelism where possible, optimize the computation by grouping sentinels over URL's and change types, and facilitate composite change detection using the same paradigm as primitive change detection. Deletion and propagation of delete semantics must be straightforward in the representation chosen. Although a number of data structures have been proposed in the literature for event detection, such as Petri nets [5], extended automata [6], it has been shown that event graphs support the requirements at the granularity and grouping that is appropriate for our problem. Hence, we have adapted and extended the event graph approach proposed for snoop [3] for detecting primitive as well as composite change.

Primitive change detection involves detecting changes to links, images, keywords, etc., in a page. In order to facilitate primitive change detection, grouping of sentinels, and data flow we construct a graph. This graph is referred to as the **Change Detection Graph (CDG)**. The graph is constructed bottom-up as shown in Figure 3.

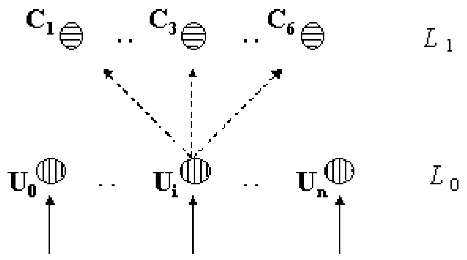


Figure 3: Change Detection Graph

The different types of nodes in the graph are as follow:

- **URL node (U$_n$)**: A URL node is a leaf node at level-0 (L$_0$) that denotes the *page of interest* (e.g., "www.uta.edu"). The number of URL nodes in the graph is equal to the number of *distinct* pages the system is monitoring at that particular instant of time. A page is fetched at this level and propagated to respective nodes at level-1.
- **Change type node (C$_n$)**: All level-1 (L$_1$) nodes in the graph are change type nodes. These nodes represent the *the type of change* on a page (links, images, keywords, phrases etc.,). Change detection of pages is performed at this level. The maximum number of change type nodes that are created in the system is equal to the product of the number of change types supported and the number of URL nodes present at that instant.

In the graph, to facilitate the 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. This subscriber list at each node contains the following:

Level-0: Contains references of level-1 nodes.

Level-1: Contains references of sentinels

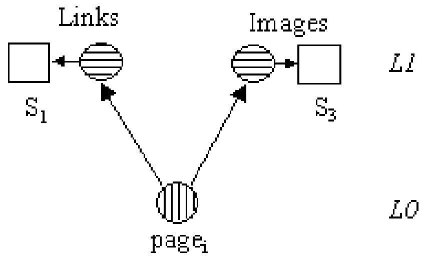


Figure 4: Primitive Change Example

monitoring that change.

The arrows in the graph represent the data flow. For example, consider two sentinels, $S_1$ monitoring changes to links and $S_3$ monitoring changes to images on $URL_i$ as shown in Figure 4. When a new version of the page with $URL_i$ is fetched, it is propagated to the links and images node. At each change type node, change is computed using a previous version of the page. If there is a change, sentinels S1 and S2 are notified.

When a sentinel is disabled, the information to stop it from performing change computation is propagated to the change type node with which it is associated. The change type node decides on whether to propagate this information to the URL node based on other sentinels on that URL. Once the URL node gets the information, it removes the references of the corresponding change type nodes and does not send any more new versions to it. For example, if $S_1$ shown in Figure 3 is disabled, the next version of the page fetched is not propagated to the links node from the URL node.

To attain system scalability and better performance, sentinels are grouped when there is more than one sentinel interested on the same change type on the same URL. For example, consider a course webpage listing the assignments

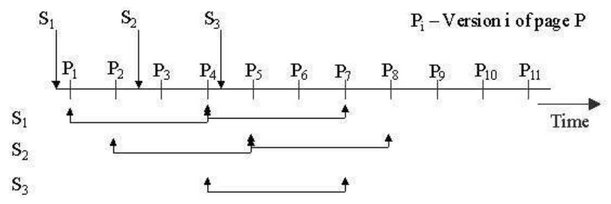


Figure 5: Compare-Options with Every:4

and projects related to that course. This webpage is updated as the course progresses and all the students would be interested in the updates happening to this page. Similarly, many users might be interested on updates happening to a news website on different topics. One might be interested in sports while another might be interested in politics. The sentinels on these keywords can be grouped and the page can be monitored for a union of the keywords {sports, politics, ...}. But sentinel grouping is not possible for fixed-interval sentinels as each sentinel has a different version. In spite of a sentinel belonging to on-change, the other attribute that plays a role in the grouping strategy is the compare-options (pair-wise, moving:n, every:n). For example, requests for pairwise comparison cannot be grouped with requests for every:3 (i.e, the current version compared with the third version from now). Consider the following example where $S_1$ and $S_2$ are on-change sentinels monitoring the same change on the same page (P) with the compare-option of "every:4" as shown in Figure 5. The points on the time line denote the arrival of sentinels and version os page P. For $S_1$, change is detected between the versions ($p_1$, $p_4$). When $S_2$ arrives, since there is already a cached version $p_2$, the change is detected between ($p_2$, $p_5$) and so on. For $S_3$, the change is computed with ($p_4$, $p_7$). Hence $S_1$ and $S_2$ cannot be grouped together whereas $S_1$ and $S_3$ can be

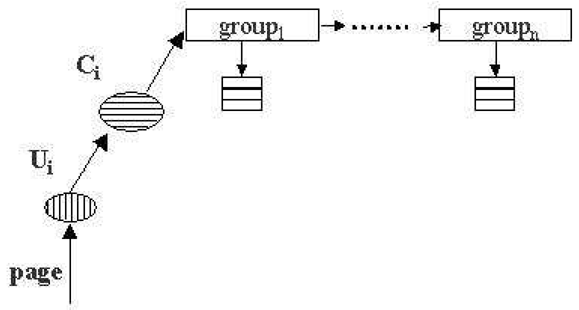


Figure 6: Grouping Data Structure

grouped.

Thus, the sentinels are grouped on a combination of change type, fetch type (on-change) and compare-options. When the compare-option is "every:n", then the corresponding time at which the sentinel arrives is taken into consideration. Figure 6 shows the grouping data structure.

So far primitive change detection has been discussed. A composite event is an event expression comprising a set of events connected through one or more composite event operators which are NOT, AND, OR. As shown in Figure 7, composite event nodes are at levels $L_2$ and above.

The change is computed for all the sentinels present in the subscriber list at the change type node. Hence, for sentinels monitoring composite changes, a representation at its constituent change type node is needed. This is implemented by creating proxy sentinels with the same properties of the original sentinel at each of the constituent change type node. Consider the scenario where sentinel $S_5$ is interested in *links and images* change on $page_i$ (Figure 8). When a new version of the $page_i$ is fetched, it is propagated to the links and images nodes. If there is a change, sentinels subscribed to it are notified. Sentinel $S_{and}$ acts as a proxy for $S_5$. When $S_{and}$ is notified, the change computed is propagated to the

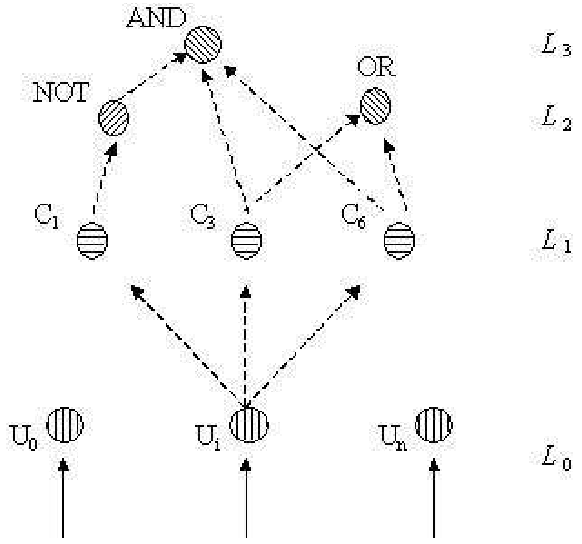


Figure 7: Composite Change Detection Graph

AND node. At the AND node, $S_5$ is informed only when it receives notifications from both its constituent S*and* sentinels.

## 6 Conclusion and Future Work

The first version of the WebVigiL system has been implemented and can be accessed from *http://berlin.uta.edu:8081/webvigil*. All the modules shown in Figure 1 has been implemented. The system is being extended in a number of ways. The current architecture of WebVigiL facilitates the monitoring of only pages without embedded frames. To monitor a page with frames, the exact URL of the frame has to be given. Work is in progress to extend the system to internally handle pages with frames. This will be achieved by extending the system to take sentinels which monitor more than one URL. Work is also in progress to provide an independent fetch module that can be installed on

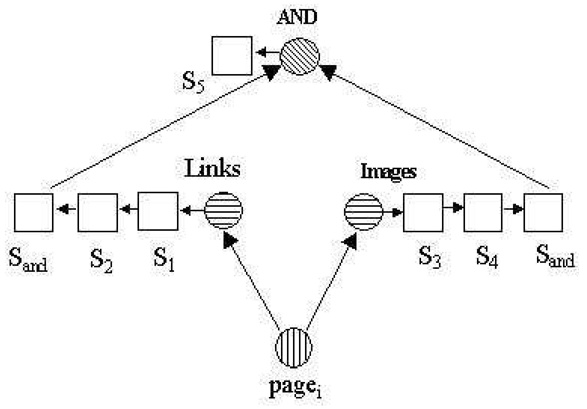


Figure 8: Composite Example

the system hosting a website. This module will locally detect changes and push that information to the remote WebVigiL server thereby reducing the network traffic.

## References

[1] E. Anwar, L. Maugis, and S. Chakravarthy. A new perspective on rule support for object-oriented databases. In *Proceedings, International Conference on Management of Data*, pages 99–108, Washington, D.C., May 1993.

[2] S. Chakravarthy et al. Hipac: A research project in active, time-constrained database management, final report. Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambridge, MA, Aug 1989.

[3] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data and Knowledge Engineering*, 14(10):1–26, October 1994.

[4] F. Douglis et al. The at&t internet difference engine: Tracking and viewing changes on the web. Technical report, AT&T Labs, 1998.

[5] S. Gatziu and K. Dittrich. Samos: an active, object-oriented database system. IEEE Quarterly Bulletin on Data Engineering, 1992.

[6] N. Gehani and H. Jagadish. Active database facilities in ode. IEEE Bulletin of the Technical Committee on Data Engineering, 1992.

[7] J. Jacob. Webvigil: Sentinel specificatin and user-intent based change detection for xml. Master's thesis, The University of Texas at Arilngton, 2003.

[8] L. Ling, P. Calton, and T. Wei. Webcq: Detecting and delivering information changes on the web. In *Proceedings of International Conference on Information and Knowledge Management (CIKM)*, Washington D.C, 2000.

[9] Mind-it. http://zeneco.com/divingparadise/netminder.htm.

[10] N. Pandrangi et al. Webvigil: User-profile based change detection for html/xml documents. In *Proceedings 20th British National Conference on Data Bases*, Coventry, UK, 2003.

[11] WYSIGOT. http://www.wysigot.com/.