Change Detection and Correction Facilitation for Web Applications and Services

Alfredo Alba, Varun Bhagwan, Tyrone Grandison, Daniel Gruhl, Jan Pieper
IBM Almaden Research Center, 650 Harry Road, San Jose, California 95120 USA
{aalba, vbhagwan, tyroneg, dgruhl, jhpieper}@us.ibm.com

Abstract

There are a large number of websites serving valuable content that can be used by higher-level applications, Web Services, Mashups etc. Yet, due to various reasons (lack of computing resources, financial constraints etc.) they are unable to provide Web Service APIs to access their data. In their desire to incorporate the latest and greatest technologies, as well as to adapt layouts that are more preferred by users, websites undergo change over time. These changes can range from minor, e.g. function name changes, to major, e.g., shifting the web platform to AJAX technologies. This paper addresses the problem of detecting layout changes for websites which are unable to provide any Web Service to access their content, yet do not mind others harvesting said content.

1. Introduction

Each large website and portal on the Internet adheres to a standard theme (template). These themes determine the layout of the sites content. Some websites allow their users to customize parts of the layout (for example portals such as my.yahoo). In these cases, certain site content may be deemed mandatory (e.g. site menu, username, location, age & gender), while the users are able to pick and choose the content to be displayed in other parts of the page (e.g., favorite-lists, images, comments etc.). Examples of such websites include MySpace.com, Bebo.com, Lastfm.com, Youtube.com etc. While these layouts and templates can be explicit - for example using CSS, more often they are simply the result of machine generation of HTML from underlying data sources such as databases or content repositories.

Contrary to conventional wisdom, web scraping [1] remains the dominant and most reliable way for extracting deep Web data [2]. Moreover, a large number of websites simply do not have the resources or see the need for providing Web Service APIs to access their data directly. Web scraping describes the practice of automatically extracting information from webpages for use in other applications. Web scrapers generally utilize knowledge about the sites layout to find and extract the desired content. For example, a web scraper extracting stock information from a website may know that the stock symbol is contained in the first column of the second table on the page.

For clients who routinely scrape information from websites, detecting changes is critical to ensuring consistent and correct usage of their service. Not only is change detection needed, but it is important to determine the type and level of change in order to better estimate the restoration of services. A reliable means for gauging this early in the notification and troubleshooting process is invaluable to a timely resolution. In this paper, we describe such a system, which was implemented as a part of the Sound Index project [3]. This technology enabled the Sound Index system to reliable deliver a top 1000 list of artists and songs, in the face of changes to the web pages of the data sources.

2. Methodology

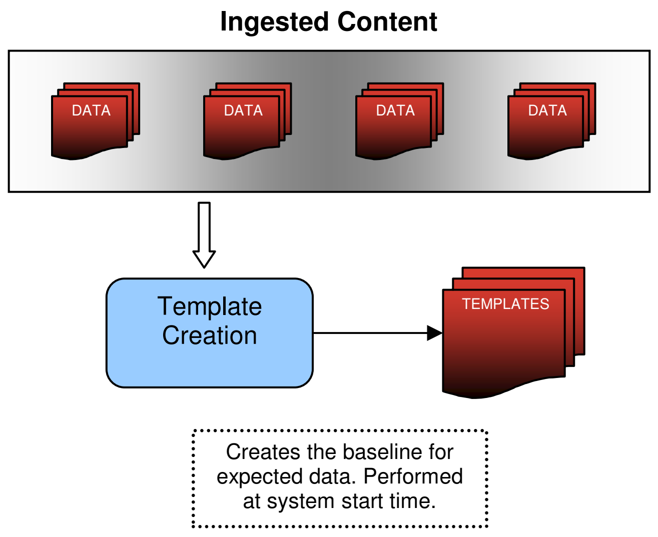
Ingested Content
DATA
DATA
DATA
DATA
Template Creation
TEMPLATES
Creates the baseline for expected data. Performed at system start time.

Figure 1. The Creation Of Base Templates.

The methodology for change detection (Figures 1 & 2) includes the following steps:

1) The creation of a template for expected data (Figure 1). This step is typically the output of the web scrapers (e.g., a key value pair containing 'artist->pageviews', or 'user->comments'). This template can be composed of:
   a. metadata (e.g., key-value pairs of the type 'artist->pageviews' or 'user->comments')
   b. actuals (e.g., key-value pairs of the type 'total-artists = 100' or 'comments-per-page = 50')
   c. rule-based (e.g., key-value pairs of the type 'artist->pageviews', where 'pageviews' is an integer)

2) Anomaly detection. When the expected output differs from the actual output, an anomaly is detected. The specific anomaly that gets detected (e.g., 'no comments found on user's page') is recorded in this step.
3) Correlation & Feedback. In this step, all of the anomalies recorded in the ongoing run are collected and correlated. For instance, if there were no comments found on any user's page, it can indicate a major update on the website. Alternatively, if only a small subset of user pages changed, it can indicate a phased rollout of new features/changes.
4) Notification. In this step, the results from the prior step are sent to:
    a. The automatic adaptation method, which attempts to modify the extraction code of the web scraper, such that the problem is resolved.
    b. The operator. Our approach employs the open source tool Nagios [4], but any other notification application can serve the purpose. If step 4a fails, the operator is notified of a critical error, if 4a succeeds, the operator is merely made aware that the system adaptation has occurred.

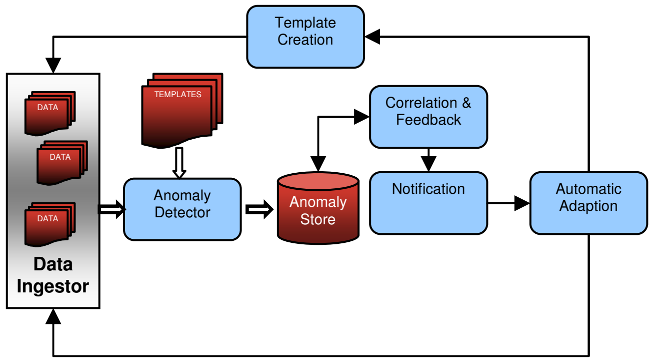

Figure 2. Framework for Change Detection and Correction Facilitation.

The methodology for automatic adaptation includes the following steps:

1. Determining "content fix points" (CFPs)

CFPs are web site data that doesn't change over time. CFPs can occur either within the extracted data or close to the extracted data. An example for CFPs within the extracted data would be a list of possible stock symbols that are always being extracted from a specific page. This list can be used to find these symbols after a layout change. An example of CFPs close to the desired data would be surrounding text such as "Today's Stock Market" in the header of the stock table.

2. Search.

Finding CFPs in the current version of the web page. Each occurrence of a CFP becomes a candidate for future extraction. The correct location is determined by finding clusters of CFPs in close proximity and the web scraper extraction information is modified accordingly.

3. Verification.

Applying the modified screen scraper to the web page and judging the quality of the newly extracted data as presented in "methodology for change detection". Moreover, if the nature of the content is temporal, e.g. a discussion board, older data can be re-crawled to receive a version of old data in the new layout. The modified screen scraper then needs to be able to extract these historic entries properly as determined by comparison with existing system data.

4. Partial Adaptation.

In instances where fully automatic adaptation is not possible or applicable (e.g., the website doesn't contain CFPs, or not all desired data on the website is made up of CFPs), the system provides a technique for machine-assisted change identification. This involves keeping an old copy of the content (e.g., user comments), and upon detecting change, locating the old content on the new version of the webpage. This is then used to create a 'change record' (essentially a diff output) that can be accessed by the operator/user to determine what changes need to be made to conform to the new website layout.

In applying the above techniques in the Sound Index system [3], we successfully performed change detection in all cases when data source changes occurred, and drastically reduced the fix turnaround times through Partial Adaptation for major changes. Our future work includes conducting a very thorough evaluation of our approach.

4. Acknowledgements

We would like to thank the BBC, specifically Geoff Goodwin (Head of BBC Switch), for their support and encouragement.

5. References

1. http://en.wikipedia.org/wiki/Web_scraping
2. Alfredo Alba, Varun Bhagwan, Tyrone Grandison. Accessing The Deep Web: When Good Ideas Go Bad". The Proceedings of the ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). Nashville, Tennessee. October 2008
3. BBC Sound Index project, http://www.almaden.ibm.com/cs/projects/iis/sound/
4. Nagios, http://www.nagios.org