Undergraduate Research Opportunity Program
(UROP) Project Report

User-centric Web Information Extraction

By

Shawn Tan

Department of Computer Science

School of Computing

National University of Singapore

2010/11

Undergraduate Research Opportunity Program
(UROP) Project Report

User-centric Web Information Extraction

By

Shawn Tan

Department of Computer Science

School of Computing

National University of Singapore

2010/11

Project No: U096883L
Advisor: A/P Kan Min-Yen
Deliverables:
Report: 1 Volume
Source Code: 1 DVD

Abstract

In this report, I present an algorithm providing an iterative method for users to generate a wrapper visually from a given web page, and also a machine learning solution for extraction of data after a site layout change. These were implemented as a system, and then evaluated both quantitatively and qualitatively.

Subject Descriptors:
H3.5 Web-based Services
H2.8 Data Mining
I5.2 Pattern Analysis

Keywords:
bookmarklet, XPath, alignment, information extraction

Implementation Software and Hardware:
Java, Ruby 1.9.2, Javascript

Acknowledgement

I would like to extend my gratitude to my supervisor, A/P Kan Min-Yen, for his patience when explaining certain concepts to me, and guidance throughout the project. I am grateful for this opportunity to work on this UROP project under his care. I would also like to thank the entire WING group, for their valuable advice, especially Jesse Prabawa, for answering so many of my questions.

List of Figures

List of Tables

Table of Contents

Chapter 1

Introduction

Information extraction deals with extracting data from input documents into a structured form. In contrast, information retrieval is concerned with retrieving relevant documents from a collection of documents. Traditionally, the task of information extraction is usually performed on unstructured free text and as such, makes use of certain Natural Language Processing techniques. Web IE however, is concerned with documents extracted from the web. Many of these documents today are in HTML, which are semi-structured, and increasingly, generated by server scripts and template-based. Thus, web information extraction is slightly different from information extraction from free text. In IE, procedures that extract certain information from structured or semi-structured documents like HTML are known as *wrappers*.

1.1 Motivation

While many systems for web information extraction have been developed over the years, many of these are used in corporate settings, and generally extract huge amounts of data from the web into a form more suited for search and retrieval. While this may be within the reach of corporations, a user who browses the web for leisure does not have access to such resources. Another drawback of these systems is that the process of getting them up and running usually involve many hours of labelling and training work. As a result, these systems are inaccessible to users who do not have the time to do this.

	RSS Feeds	Screen Scrapers
Availability	Lies with content provider	Anything that is displayed can be scraped
Content Extracted	Lies with content provider	Lies with user
Affected by Layout	Content provider provides content, no layout involved	Breaks on layout change
Format	Still requires parsing of RSS XML in order to maipulate data.	Can extract into any format
Technical Knowledge required	User has only to know how to use RSS.	Requires programming experience

Table 1.1: Pros & Cons of using RSS or wrappers (screen-scrapers)

One might argue that this is where RSS feeds come in, to provide users updates to constantly changing data on their pages. However, the data provided on RSS is usually controlled by the site in question, and may not provide the data that the user actually needs. One other alternative to tackle this problem would be to write "screen scrapers", as this would provide complete control over the data extracted. This approach faces another set of problems, the most glaring being that most users are not familiar with programming, let alone the many other technical issues faced when doing screen scraping.

1.1.1 User study

We conducted a survey comprising of 30 students studying in tertiary institutes, and asked them about their prior experiences with Web IE.

Surprisingly, 60% of the students have never heard of Web IE. In order to keep up to date with topics they are interested in, a little over 60% manually visit the sites of such topics, while 22% make use of RSS feeds. When asked if they knew what a screen-scraper or a wrapper was, only 1 out of the 30 have heard of them, and 96% have never used any form of Web IE tool before.

This shows that while companies have the capabilities and resources to harness the power of data mining and Web IE, many users, even those as tech-savvy as university students, are either unaware of such technology, or such tools are not as accessible to the average user.

When asked how they would go about getting specific data from a set of pages into a tabular form for some research or personal interest, many of them said they would approach the problem

manually, copying and pasting the required data into a spreadsheet or a text file.

As such, we see a gap here which needs to be filled by bringing web IE closer to the average user by making creation of wrappers more user friendly, and at the same time, creating wrappers that are robust and resistant to layout changes. In essence, a "super scraper" for user-centric web information extraction.

1.2 Goals & Challenges

In this report, we present a system that attempts to solve some of these problems by doing the following:

1. Provide an intuitive interface for labelling that is platform agnostic, and takes advantage of the many advancements in Javascript. This will provide the user with an immediate visual feedback as to the items that he/she will be extracting, and at the same time reduce the amount of labelling that needs to be done.

2. Create a more robust framework for extraction of the selected information using machine learning. The classifier will have less focus on the HTML structural information of the tags in order to be resistant to any layout changes made to the page.

We will describe our approach in the following chapters. In Chapter 2 we will look at some of the other work that has been done in the areas of visual annotation of training data, XPath generation and creation of robust wrappers. In Chapter 3, we will describe the various algorithms and methodologies involved in the creation of the system. In Chapter 4, we will look at the implementation of the system, while finally in Chapter 5 we will see the 2 main components of the system being evaluated on 2 fronts, a human evaluation for the selection interface, and an evaluation of the machine learning component.

Chapter 2

Related Work

Initially, web IE involved manually written wrappers that catered to specific needs and information sources (Chawathe, Garcia-Molina, Hammer, & K, 1994; Perkowitz & Etzioni, 1995). While this provided accuracy in the information extracted, it required users of such IE systems to have some background in programming. Later, many wrapper induction tools came about after Kushmerick's paper (Kushmerick, Weld, & Doorenbos, 1997), using several different approaches to the problem (e.g. machine learning) (Freitag, 1998; Soderland, 1999). Subsequently, from 2001, systems that required no user intervention were developed (Chang & Lui, 2001; Crescenzi, Mecca, & Merialdo, 2002; Arasu & Garcia-Molina, 2003). This suggests a trend toward information extraction systems with less human interaction.

2.1 Visual feedback for User annotation

IE systems cannot completely be free of user intervention: "Supervised approaches, although require annotation from users, extend well to non-template page extraction if proper features are selected for extraction rules" (Chang, Kayed, Girgis, & Shaalan, 2006). Since user annotation is crucial to web IE, it is important to improve current user interfaces for labelling. Attempts such as *Lixto*, which has some focus on the user interface aspect of labelling, have tried to make this process more visual (Baumgartner, Flesca, & Gottlob, 2001). Other ways to do this would be to reduce the amount of sample data required by the system, or to provide a way to give the

user immediate feedback as to what the result of the learnt classifier would be.

There are several commercial systems that also do this, like Mozenda and Needlebase. MIT's Solvent is part of a mash-up creation system that also provides users with a similar visual feedback system. Enabling the users to see what the system will extract helps give a sense of transparency from the system, and in this way, allows the user to make any required corrections immediately.

However, the abovementioned systems, with the exception of Needlebase, require users to perform the changes within a customised browser inside the system. In addition, Mozenda, Needlebase and Solvent require users to have had experience with programming, as part of their extraction process requires the user to define the "flow" of the extraction.

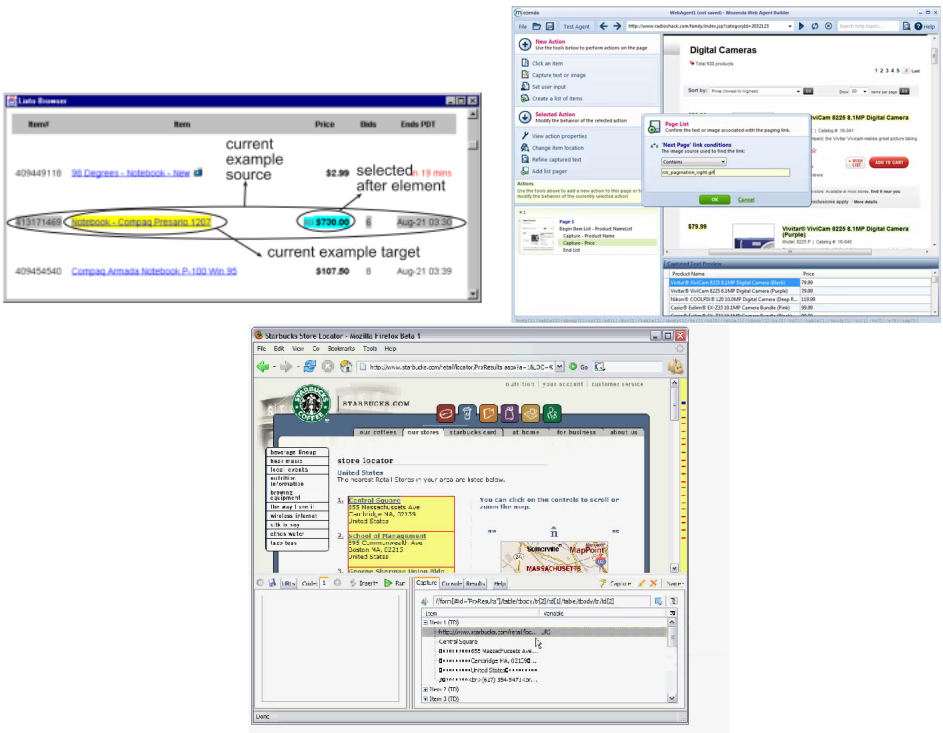

Figure 2.1: The *Lixto*, Mozenda and Solvent interfaces.

In our proposed system, grabsmart, we aim to provide a system which allows users to have a sense of what will be extracted, with the added advantage of keeping the installation process simple by introducing this to the user in the form of a bookmarklet, so that all that is needed is a drag-and-drop action.

We have identified XPath as a suitable format to query the selected pages for extraction, due to its widespread use[1], so by being able to generate XPath queries, we are also able to provide a way for writers of their own scrapers to do so more easily.

2.2 Generation of XPath for wrappers

XPath is a powerful query language commonly used for selecting particular elements in XML documents. Recently, many scrapers are using XPath as a method of retrieving the relevant element within an HTML page. This is because its representation of the selected items is somewhat similar to the way we access directories in our file systems, and also the ubiquity of XPath as a query language for many HTML parsers makes XPath viable as a tool to retrieve the relevant elements within a page. However, the process of coming up with the appropriate XPath lies with the programmer of the scraper. This requires prerequsite knowledge of DOM trees and XPath, and creating the XPath manually also introduces human error.

Methods are available to automate the creation of these XPath queries. A method for extracting XPaths is described in (Anton, 2005). The approach models the DOM Tree as a traversal graph, and uses this to try to create an XPath that gets the wanted elements for all pages it is given.

Later, Dalvi, Bohannon, & Sha (2009) explored the approach of developing a tree-edit model of HTML, modelling changes to a page as a stochastic process, using it to improve wrapper construction for robustness. The method generates a list of candidate XPaths for a certain selected region within the page. It used archived data of a given website to calculate its *compute transformation probability*. Using this, the most robust XPath candidate was selected from the list. This process, however, required access to archives of older versions of the site. Such methods of XPath generation do are not immediate, and, for the most part, does not involve a lot of input from the user, but the drawback of that would be that the user is unaware of what will be extracted until he/she sees the extracted results.

In contrast, our system trades the robustness of the XPath query with the speed of gener-

[1]Implementations of Javascript in most browsers have a native API for querying using XPath

ation. Our proposed system will be able to create, and display elements on the page that will be extracted as the user is selecting items.

2.3 Robust wrappers

Another problem faced when trying to automate web IE is having to regenerate or re-implement the wrapper every time there is a change in the site layout. This is an important problem given the ease in which templating systems allow pages to be "themed" and redesigned. Wrappers need to be robust enough to handle such frequent changes, and be less reliant on the underlying HTML structure of the page.

One method of approaching this was to make use of the visual features of the page. The PARCELS (Lee, Kan, & Lai, 2004) system is not strictly an information extraction system, but rather, is concerned with dividing up the page and classifying the resulting blocks correctly. As input, the user is required to input several example pages with labelled blocks as examples. The system then extracts features from 2 different aspects of the blocks: Its *lexical* features and its *stylistic* features. Using both of these the classifier uses a technique called co-training, machine classified blocks from either view is fed into the training data of the other view. The system was then improved to include usage of inter and intra similarities between pages as features(Lau, 2005). Gatterbauer also approached the problem of extracting data from web tables by looking directly at the rendered 2D output, extracting tables (or grids) within them, and then analysing these in order to extract relevant information (Gatterbauer, Bohunsky, Herzog, Krüpl, & Pollak, 2007).

2.4 Conclusion

After studying the drawbacks and advantages of some of the previous work in the area, I hope to make two main contributions with my system:

1. Provide an intuitive interface for labelling that is platform agnostic, and takes advantage of the many advancements in Javascript. This will provide the user with an immediate

visual feedback as to the items that he/she will be extracting, and at the same time reduce the amount of labelling that needs to be done.

2. Create a more robust framework for extraction of the selected information using machine learning. The classifier will have less focus on the HTML structural information of the tags in order to be resistant to any layout changes made to the page.

For the rest of the report, I will define users of the system as users who do not have access to resources required by current information extraction system. This includes not only the hardware and processing power required, but also the time needed for the labelling work done. Also, I will define robustness as the wrapper's resistance to changes in HTML structure from the time that the wrapper was derived.

Chapter 3

Method

In the following sections, we will elaborate on the contributions of this report. In the first section we will describe traversal path alignment algorithm, which we propose to overcome the issues of current methods of XPath generation by providing a way to create viable XPaths based on elements that users have selected or rejected, giving them a visual understanding of what will be extracted as they complete the process.

Next, we address the approach we take in trying to create a robust wrapper for the site. We propose a framework for learning models for classification of HTML tags or DOM nodes by learning from extracted instances using the XPath.

3.1 Path Traversal Alignment

An important aspect which describes tags which are visually similar is its traversal path from the `<BODY>` tag. Parents of selected tags with similar tag names or that have similar attributes are likely to be tags that look visually similar on the page. This also implies that the elements are probably items that that the user would also be looking for.

We propose using a modification of the Needleman-Wunsch (Needleman & Wunsch, 1970) global alignment algorithm for this purpose. Each traversal path could be seen as a sequence of symbols that need to be aligned based on their common attributes. The more attribute-values they have in common, the more likely they are ancestors of elements of the same type.

However, when reconstructing the aligned sequence, one needs to be aware that the symbols are not entirely equivalent, and that only the common attributes should be retained.

Hence, we make the following modifications to the Needleman-Wunsch algorithm:

1. The scoring function used is based on the number of attributes in common between the 2 elements.

2. The method for reconstructing the aligned path is given as a parameter. We will describe in the following sections why this is needed.

Algorithm 1 PTAlign(P, Q, λ)

1: **for** $i \leftarrow 1$ to $|P| + 1$ **do**
2: **for** $j \leftarrow 1$ to $|Q| + 1$ **do**
3: $M_{i,j} \leftarrow \max(M_{i-1,j}, M_{i-1,j-1}, M_{i,j-1}) + \text{Score}(P_i, Q_j)$
4: **end for**
5: **end for**
6: $R \leftarrow []$
7: **while** $i > 1$ and $j > 1$ **do**
8: **if** $M_{i,j} - \text{Score}(P_i, Q_j) = M_{i-1,j-1}$ **then**
9: $R \leftarrow \lambda(P_i, Q_j) + R$
10: $i, j \leftarrow i - 1, i - j$
11: **else if** $M_{i,j} - \text{Score}(P_i, Q_j) = M_{i-1,j}$ **then**
12: $R \leftarrow * + R$
13: $i, j \leftarrow i - 1, j$
14: **else if** $M_{i,j} - \text{Score}(P_i, Q_j) = M_{i,j-1}$ **then**
15: $R \leftarrow * + R$
16: $i, j \leftarrow i, j - 1$
17: **end if**
18: **end while**
19: **return** R

Figure 3.1: The *traversal path alignment* algorithm

Given the input of 2 arrays P and Q of elements, the scoring function is given as $\text{Score}(P_i, Q_j) = |P_i \cap Q_j|$, where P_i and Q_j are elements (sets of attributes) along P and Q. After the alignment, a new generalised traversal path is created, P', such that $P'_k = P_i \cap Q_j$, where P_i and Q_j are elements in the original traversal paths that were aligned. Figure 3.1 gives the pseudocode for the *Path Traversal Alignment* algorithm (PTA).

When selecting additional elements for generalising the XPath, the λ function would be

defined as $\lambda(P_i, Q_j) = P_i \cap Q_j$. This would reconstuct a generalised traversal path with common elements and their common attributes. Gaps in the path are represented as "*" in the output. During the serialisation of the XPath, groups of these are converted to `//` to represent any descendant.

This process is a bottom-up approach: Users start with a single element, and as more items are selected, the items captured by the XPath generated grows. For each element along the selected item's path, the similarities between the tag's attributes are kept, and the differences removed. This results in an extraction rule with fewer and fewer constraints as more elements are selected.

P	`div`	`div`	`div`	`ul`	`li`	`h3`	`a`
id	mainContent						
index	4	1	1	3	1	1	1
class		content	newsl, pagedNewsList	newslTop			

Q	`div`	`div`	`div`	`ul`	`li`	`h3`	`a`
id	mainContent						
index	4	1	1	3	2	1	1
class		content	newsl, pagedNewsList	newslTop			

After PTA:

Result	`div`	`div`	`div`	`ul`	`li`	`h3`	`a`
id	mainContent						
index	4	1	1	3		1	1
class		content	newsl, pagedNewsList	newslTop			

Figure 3.2: Example of fewer constraints after each iteration

Using alignment of the path elements, XPath rules can be generalised for elements at different levels of the DOM tree. Figure 3.3 is an example of the array generated when 2 `<a>` tags are compared, one at a depth of 5 from the `<body>` tag, and another at depth 3. The `<body>` tag is omitted in order to reduce the running time of the algorithm, as it is common in the paths for all visible elements on a given page. The algorithm is then able to generalise an XPath: `//div[@id='main']/div//a`. This XPath will include both selected elements, as well as other

anchor tags that the user may be interested in.

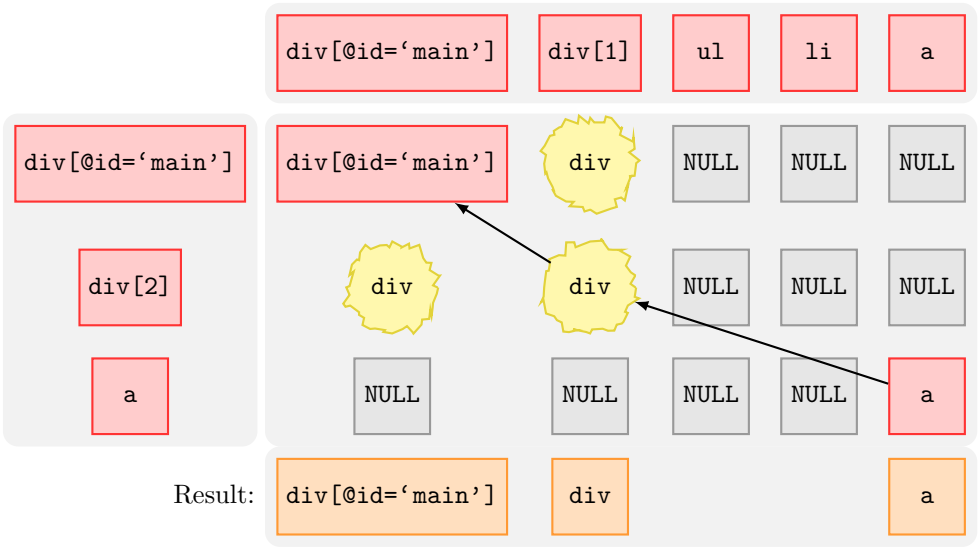

Figure 3.3: Table generated by the LCAS algorithm

It is important to note that serialising the generalised model to its XPath form would result in loss of information. As such the model in its unserialised form is kept, and used again when another item on the page is selected. Since this process has an associative property, the order in which items are selected on the page does not matter, and the user can then select 3 items or more in order to specify the items he wants.

3.1.1 Element rejection

Once the user has defined a region for extraction, he/she may find that the region includes items that are irrelevant. Our objective was to make this process part of the selection process as well. In order to achieve this, we used the same alignment algorithm, with P as the already generalised path, and Q as the new traversal path of the element that the user wishes to reject, with λ defined as $\lambda(P_i, Q_j) = Q_j - (P_i \cap Q_j)$.

3.1.2 Limitations

There are still several limitations to this method. When $Q_j = P_i \cap Q_j$ for all i, j in the aligned path, that rejection path cannot be used. This usually implies that the traversal paths of the

wanted elements and the rejected elements are indistinguishable using the considered attributes. In such a case, the user is not allowed to select elements that cause such an event. Once the first rejection element has been generalised with the PTA algorithm, additional rejection elements can then be generalised using the first algorithm.

Also, the mentioned method of generalising the XPath is a form of a multiple sequence alignment , but since it is approached using a sequence of pairwise alignments, the optimal alignment may not be achieved. However, the selection scopes of the generated XPaths are generally usable for scraping.

One final limitation that we intend to overcome in our project is that the XPath generated in this manner will not work after a layout change that affects the layout of the DOM tree. In the following section, we will describe how we overcome this problem.

3.2 Robust Extraction

The PTA algorithm enables us to generate a fixed rule (XPath) to extract data from the given pages. However, XPath is heavily reliant on the structure of the DOM page in its present form: Any element extracted is defined by its ancestor nodes and their attributes. Once this is changed, which is likely due to a layout change, the same XPath may no longer be applicable to the current page.

In order to be able to extract data even after a change in layout, another method of extraction without the use of XPath is required. Our approach for this system uses a machine learned model. The rest of the chapter elaborates on the details of the implementation.

In our framework, we use the provided XPaths together with the URLs of the pages to be extracted as annotated documents. Each node in the DOM is treated as a separate instance, and labelled based on the XPath it was extracted by. The XPaths are assumed to be mutually exclusive. We have chosen to use decision trees because the generated models are small, and can be easily interpreted. The generation of these models are also relatively faster than other classification models. Since CPU time and memory space are constrained resources, decision trees were suitable for our needs.

Looking at any web information extraction system using machine learning classification techniques, the workflow of a user can be generalised into the following steps:

1. The user finds a page or site that he/she would like information to be mined from.
2. Samples of the page presented to the user for labelling. Positive examples are selected and these are fed into the machine learner to create a model.
3. Using this model, data is extracted from similar pages, this data is then put into storage (e.g. database)
4. Some systems repeat steps 2 and 3 iteratively until an accurate model is achieved.
5. The user can then access the data storage for the mined information.

3.2.1 Features used

To keep the model more robust and resistant to layout changes on the original site, content features are extracted in order to use less of the HTML structure for extraction. In this system, a J48 decision tree classifier is used. Positive examples of HTML elements are those returned when the XPath is applied to the given page. The following features are extracted:

- Word occurrence count, or tokens (Numeric)
- Tag name (Discrete)
- Previous sibling, next sibling and parent tag name (Discrete)
- Number of words/tokens (Numeric)
- Ending with character (:,-,.) (Discrete)
- Header (Discrete)

For subsequent extractions, the server will first use the available XPath for extraction from the site. If the XPath rule fails to find elements for extraction, the learnt classification model is

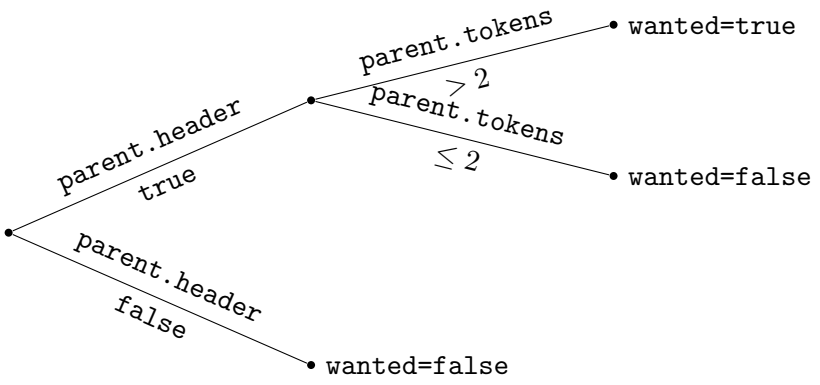

Figure 3.4: Decision tree generated when search entries on Google are selected.

used. Figure 3.4 shows a decision tree when the process is run on a Google search result with the result entries selected for extraction.

As we can see, the generated decision tree seems pretty reasonable given a Google search result. The `parent.` prefix is the equivalent to the statement "if the parent of the element". We can see that the generated tree is looking for elements with parents that are headers and contain 2 or more words.

3.2.2 Crawling for more instances

We found that many annotations capture very few elements per page when compared to the total number of elements for that page. By attempting to balance the number of elements that belong to labels with those that were not captured by any XPath, we hoped to improve the performance of the classifier. However, this still yielded relatively poor results. Eventually, we crawled the site for other pages which had elements captured by the same set of XPaths. This gave us more instances to train the classifier on, and gave better results.

The crawler starts with a given set of pages, and crawls links with URLs similar to the given pages. The similarity score is given by the edit distance between the link URL and the URL of the wanted pages. A heap is used to store the seen links, and the most similar URL selected at every round. The crawler stops when it has collected a fixed number of instances, which is now set at 500.

3.2.3 Limitations

In Weka's implementation of their machine learning algorithms, the serialised models created tend to store all attributes and their values. Due to this, some of the models are really large, with a lot of redundant data. This takes up much I/O time and storage space on the server. However, in comparison with the implementations of the other classification methods in the Weka library, the decision tree classifier does perform better than the other classifiers in terms of runtime and model size. A decision tree classifier optimised specifically for the system was not implemented due to time constraints.

Also, from our tests, the learnt model does better on heading-type results, while other fields tend to have a low precision score, we will discuss the evaluation results in detail in Chapter 4.

Chapter 4

The grab*smart* System

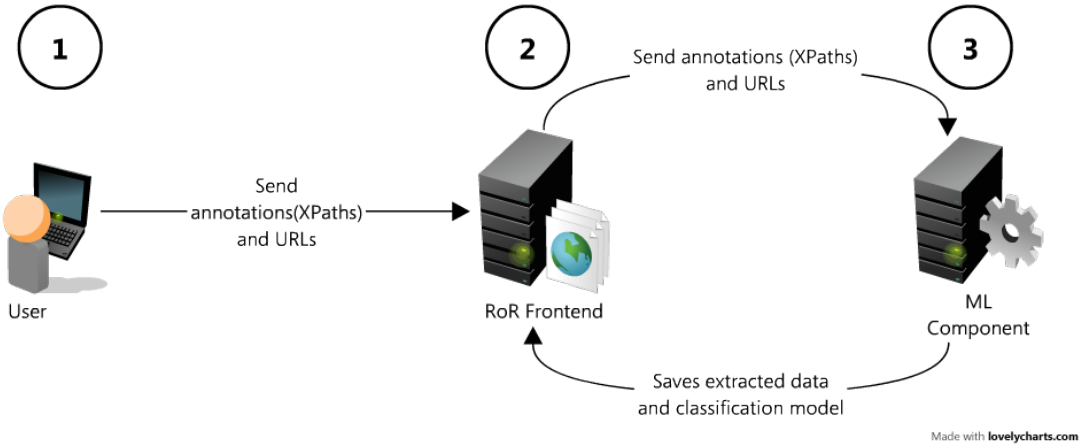

Figure 4.1: Overall system architecture

This chapter details the implementation of the methods mentioned before into a working system. The system consists of 3 main components, the architecture is depicted in Figure 4.1. The components shown are the following:

1. **Selection Interface** This is implemented as a bookmarklet which the user can simply drag into the toolbar of the browser.

2. **Web Application** This is the frontend of the system, allowing the user to create, update and delete their extractors, and also to be able to view their extracted data.

3. **Machine Learning Component** This is the component that creates a classification

model based on the generated XPath and the pages selected to be extracted from.

4.1 Selection Interface

The selection interface described in this section aims to provide visual feedback to the user when building a suitable wrapper for the page chosen by the user. The interface is implemented in the form of a bookmarklet which the user simply has to drag and drop into his/her browser toolbar. This bookmarklet can then be activated when the user reaches a page which he/she wants to have something extracted. Clicking on items will select them in green, and subsequent clicking will expand the scope of the extraction, with the items to be extracted highlighted in yellow. Figure 4.2 shows a screenshot of a search result in bookdepository.co.uk with the bookmarklet activated.

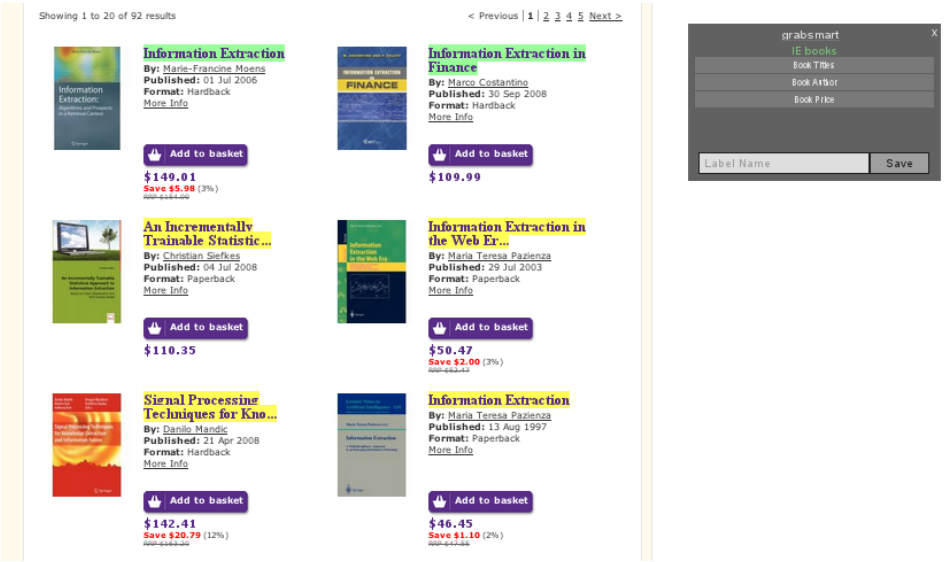

Figure 4.2: An example of the selection interface in action.

For greater automation, the bookmarklet interface attempts to reduce the amount of labelling work the user has to do by trying to predict what the user wants to extract from the page. More specifically, as the user selects the individual HTML elements on the page using the interface, the bookmarklet attempts to generalise an XPath that captures the selected elements, and also elements on the page with similar characteristics, like class names and position within the parent tag.

The process is an iterative one. When the user clicks on a new element on the page, the XPath is recalculated, and then used to highlight the captured items on the page. This way, the user understands the changes he/she has made to the extraction scope as he/she clicks on additional items. The user is then able to perform this task for any number of labels the user thinks is appropriate for the extractor he/she is creating.

4.1.1 Engineering challenges

One of the aims of the interface was to ensure that the item would not require the hassle of installation. This included not only standalone, specialised browsers for selection of items, but also extended to plugins – since a different plugin would have to be written for every browser. We made the decision to go with a bookmarklet because it was one of the few ways we could still run our code on top of another website. In general, a bookmarklet achieves this by injecting a `<SCRIPT>` tag into the host page, with the `src` attribute pointed to the main code hosted on the application's server. Since this meant downloading the code every time the bookmarklet is activated, one of our challenges was also to keep the code small.

The bookmarklet sends the created unserialised generalised traversal paths and XPaths back this by way of AJAX callbacks to the RoR frontend. One of the technical difficulties in implementing this was the fact that the standard XMLHttpRequest method of AJAX callbacks were not usable due to the cross-site security put into place by the Javascript API. The workaround was to use hidden form elements within the interface to send POST requests back to the server. Data was fetched by inserting script tags and callbacks to allow the JSON objects to be passed back into the small Javascript application.

4.2 Web Application

In our system, the user is aware of only creating **extractors**. In each of these **extractors**, there is a set of **labels** and a set of **pages** that the user has selected for extraction by activating the bookmarklet on the pages that he/she wanted. The web application we have developed aims to provide the user a way to modify and manipulate the concepts we have mentioned.

4.2.1 Extractor list

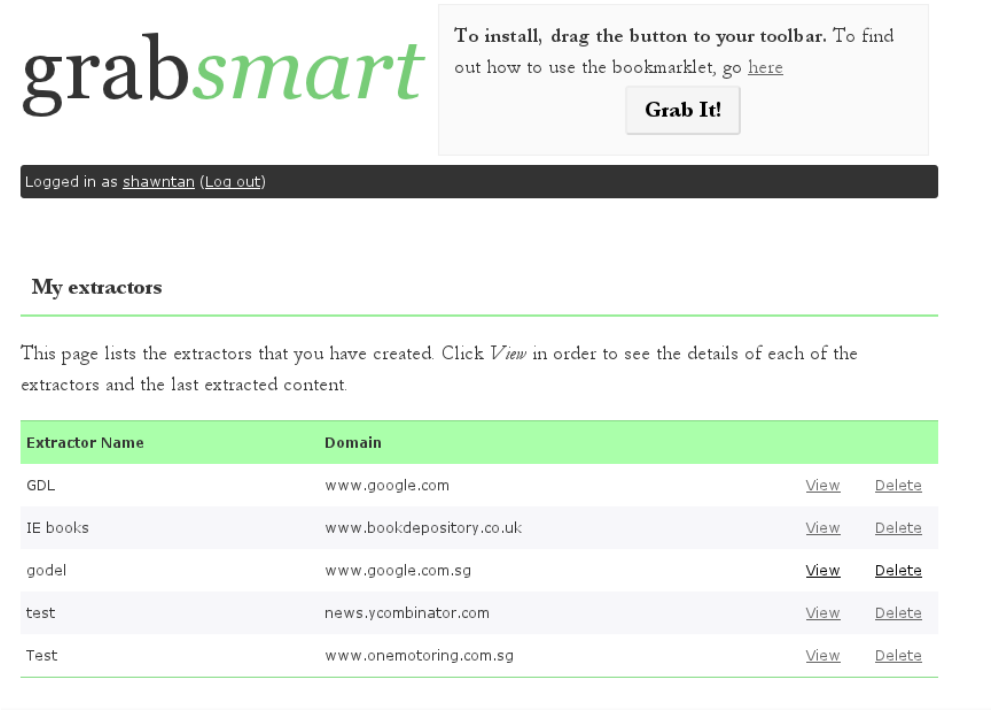

Figure 4.3: Dashboard screenshot

Figure 4.3 shows a screenshot of the user's dashboard. This is the first page the user sees after he or she has logged in. The page lists the various extractors that the user has created, and the user is able to delete them or view the extractors in greater detail by clicking "View".

4.2.2 Viewing the extractor

The detailed view of the extractor shows several items, which can be seen in Figure 4.4:

1. List of the labels, together with their corresponding XPath. We have chosen to supply the generated XPath in the page for the benefit of users who are familiar with writing screen-scrapers.

2. The list of pages that the labels will be applied on.

3. The latest list of data extracted from the data, if it exists.

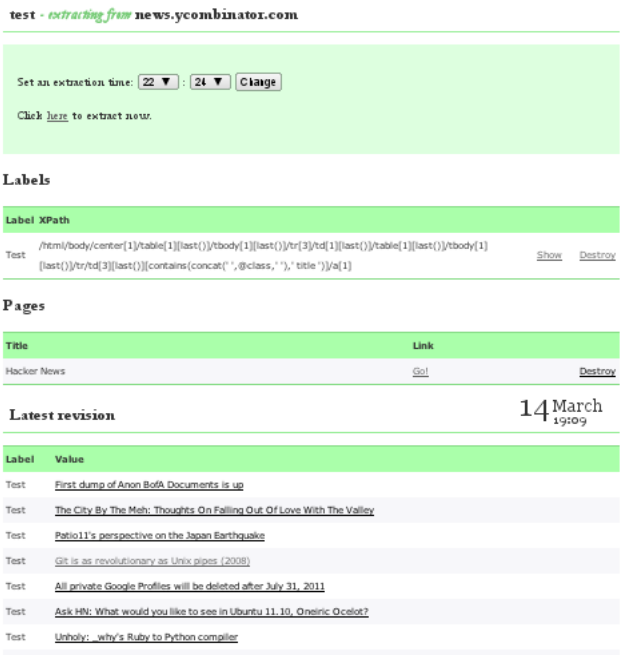

Figure 4.4: Extractor view

The user is, again, able to delete any labels or pages that he/she no longer wishes to extract using the extractor. Also, the page has a linked RSS feed that the user can subscribe to.

It is at this page that the user is able to do a test run of the extractor. Once satisfactory, the user can then set a daily time for the data to be extracted.

4.3 Machine Learning Component

After the user clicks on "Extract now" or when the extractor is scheduled to be executed, the machine learning component will retrieve the relevant pages and extract the data using the extractor's XPath. Using this set of data, it will then proceed to retrieve more similar pages with the method described in the previous chapter.

This process is meant to be transparent to the user, so the user should not have to make any changes to his configured extractor. The extractor should then work even though a layout change has occurred.

The extracted data is inserted back into the database, and then made available to the user. At the same time, features are extracted from this data in order to create a model for extraction

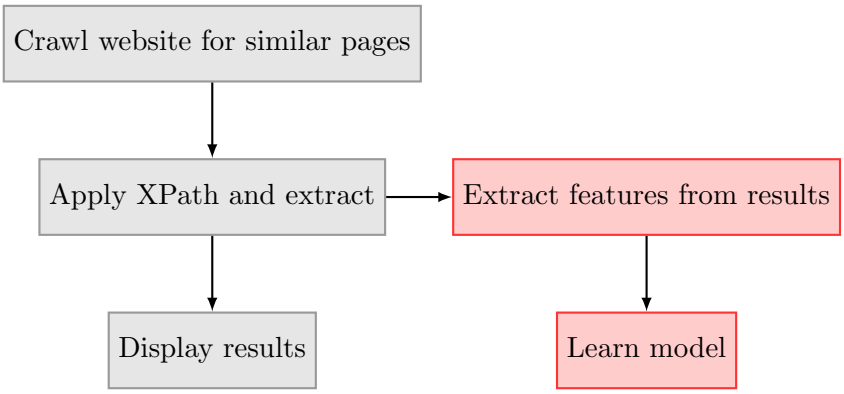

Figure 4.5: Workflow of the machine learning component

for future use.

Chapter 5

Evaluation

We have divided the evaluation of our system into 2 important sections. The selection interface deals with a more subjective topic, since its performance is dependent on how easy users find selecting items for extraction is. For this component of the system, we are doing a human evaluation on the qualitative performance of the system. In the second section, we will describe the evaluation of the machine learning technique we have used, and how well this performs when attempting to extract data from a webpage after a layout change.

5.1 Human evaluation of Selection Interface

For the evaluation of this part of the system, a group of 30 students were paid to do a simple 30 minute evaluation of the system. A simple tutorial was given to each of the students, and once completed, a simple extraction task from the bookdepository.co.uk website was to be carried out. Lastly, the students were asked to fill out a questionnaire on the following aspects of the system: Ease of installation of the bookmarklet, the ease of item selection, and the ease with which they could view the extracted items. The full content of the entire survey can be found in the appendix, along with the survey results.

The students were from various tertiary institutions in Singapore, and were given a small sum of money for their time.

"I think that should be the most simple way to install a bookmarklet. I think you should consider the case when the bookmark toolbar is hidden, and if it's possible, show a tutorial video for beginner."

5.1.1 Analysis

The participants were asked to rank the ease of use for the installation of the bookmarklet on a scale of 1 to 5. On the average, the score was .... The installation procedure was meant to be simple and easy: Dragging the button from the page to the toolbar. However, some of the participants had an issue with this since some browsers, for example Chrome, do not show their toolbars by default.

On the intuitiveness of the selection procedure, the average score was ..... Some of the participants found the interface for modifying existing labels confusing, while some had issues when trying to select elements "behind" its child elements.

About 75% of the participants found the visual feedback for the selection procedure useful in understanding what the extractor would be extracting.

Viewing the extracted data seemed to be a problem for many of the participants. The average score given for the ease of viewing extracted data was at ..... This may be due to the way in which the page is updated after the data has been extracted and inserted into the database.

In general, we find that the selection interface has been successful in its task: Aiding in the process of annotation by giving visual feedback.

5.2 Machine Learning evaluation

We want to be able to extract the same content from pages which have had a layout change. So our evaluation method involves attempting to extract data after a layout change by training a classifier for the previous.

5.2.1 Methodology

We have adopted a similar evaluation to that of (Dalvi, Bohannon, & Sha, 2009), with slight modifications. Since our system only uses the learnt models when the layout of the page changes (which is detected when the XPath does not retrieve any items), we have restricted the evaluation of the learnt models between layout changes. We have decided to use old pages from web.archive.org to simulate old pages that have not undergone a layout change. To define a layout change, we use the tree-edit distance (Zhang & Shasha, 1989) between 2 pages as a simple metric. In order to find layout changes within the list of pages we retrieved from Web Archive, we took the tree-edit distance of the DOM tree between every pair of consecutive pages. The graph of these values is shown in Figure...

The spikes visible in the graph were signals of a high tree-edit distance between the pairs of pages, and signals some sort of layout change. What we are looking for, however, are changes in which the XPaths fail to work. For each of the labels we are hoping to extract, we check the if the application of the XPath yields any resulting DOM nodes. If they do not, the page was considered a layout change, and the pages prior the change were randomly sampled for 10 instances for learning.

We created 5 sets of 10 pages from the pages retrieved from web then trained the classifier on each of these. The test results were then compared to the data extracted if an XPath was created manually for each of these sets.

5.2.2 Analysis

Chapter 6

Conclusion

When tackling the problem of Web Information Extraction in the context of the average user. Having looked at some of them in Chapter 1, we have selected a subset of these issues: Visual Feedback when performing annotations and creating Robust Wrappers for information extraction.

In our report, we have shown that a simple alignment algorithm can be applied iteratively to produce the desired effect to give users a visual feedback of what they will be selecting. We have also used machine learning to tackle the issue of creating robust wrappers in the form of a decision tree. These wrappers are then used as backups when layouts change and the original XPath fails to work. These solutions were then implemented in the *grabsmart* system.

6.1 Contributions

While *grabsmart* is still far from being a full-scale Web Information Extraction system, it has the following advantages when compared to other Web IE solutions:

1. It provides an immediate visual feedback when the user does the annotation process.

2. From a usability perspective, we have shown that it is: Easy to install, relatively simple to use, and the extracted data can easily be viewed.

3. It provides an alternative to the rigidness of the generated XPath with the decision tree

wrapper alternative.

4. The report also proposes a method for automating the detection layout changes given a chain of archived pages by using the Zhang-Shasha tree edit distance.

6.2 Future Work

With the *grabsmart* system only tackling a few of the problems in User-centric Web IE, it leaves much room for future work to be done:

6.2.1 User Interface

The selection interface, or the bookmarklet, can be made to consider more features of the selected elements. For example, the current implementation only takes into account the attribute values of the elements. Since the XPath syntax also allows for regular expression predicates, it is concievable for a method to be formulated that creates a regex pattern common to all selected items. This would give a more "content-based" XPath that may be more selective.

The web application, being less of the focus in this project, could have more functionality built in. Since the extracted data is already present on the database, one could create ways in which this data could be manipulated and displayed. Currently, only the data from the latest extraction is available. The data from the previous extractions could be used for comparisons with the current extracted data, examples of such data include Linux distribution download rankings or book prices.

6.2.2 Machine learning component

For this project, we have made use of a library from the Apache project known as HtmlUnit. This library provides us with a "headless browser" from which to access the annotated pages, but it has several drawbacks. Unlike a true browser, it is unable to parse the CSS associated with the page, as well as the Javascript. This deprives us of many usefull visual features that would be obvious to the user. An alternative to this would be the WebKit library, since it is a widely used rendering engine. A headless instance of WebKit could function like a browser,

allowing us to query the the page for visual features that would be useful for classifying HTML elements.

6.2.3 User-centric Web IE in General

A good next step forward for the system would be to be able to do some form of focussed crawling through the site for a "chain" of pages that the user might want.

Since the focus of many scraping systems are lists of structured data, many of these are paginated, and one of the limitations of our system is that the extractor does not go beyond the pages the user has annotated. Being able to infer a navigation pattern that goes through this paginated data, and then extracting this automatically would be useful for users who do not know how to write a scraper.

References

Anton, T. (2005). XPath-Wrapper Induction by generalizing tree traversal patterns. *Lernen, Wissensentdeckung und Adaptivitt (LWA)*, (3), 2005.

Arasu, A., & Garcia-Molina, H. (2003). Extracting structured data from Web pages. *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*, , 2003, 337.

Baumgartner, R., Flesca, S., & Gottlob, G. (2001). Visual Web Information Extraction with Lixto. *Proceedings of the 27th VLDB Conference*, , 2001.

Chang, C.-H., Kayed, M., Girgis, M. R., & Shaalan, K. F. (2006). A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, *18*(10), October, 2006, 1411–1428.

Chang, C.-h., & Lui, S.-C. (2001). IEPAD : Information Extraction Based on Pattern Discovery. *Proceedings of the 10th international conference on World Wide Web*, , 2001, 681–688.

Chawathe, S., Garcia-Molina, H., Hammer, J., & K (1994). The TSIMMIS project: Integration of heterogeneous information sources. *Proceedings of IPSJ*, , 1994, 1–12.

Crescenzi, V., Mecca, G., & Merialdo, P. (2002). RoadRunner : Towards Automatic Data Extraction from Large Web Sites. *Proceedings of the 27th VLDB Conference*, , 2002.

Dalvi, N., Bohannon, P., & Sha, F. (2009). Robust web extraction: an approach based on a probabilistic tree-edit model. *Proceedings of the 35th SIGMOD international conference on Management of data* (pp. 335–348), 2009: ACM.

Freitag, D. (1998). Information Extraction from HTML : Application of a General Machine Learning Approach Extraction as Text Classi cation. *Search*, , 1998.

Gatterbauer, W., Bohunsky, P., Herzog, M., Krüpl, B., & Pollak, B. (2007). Towards domain-independent information extraction from web tables (pp. 71–80), Banff, Alberta, Canada, 2007: ACM.

Kushmerick, N., Weld, D., & Doorenbos, R. (1997). Wrapper induction for information extraction.

Lau, A. M. (2005). Advancing PARCELS : PARser for Content Extraction and Logical Structure Using Inter- and Intra- Similarity Features Advancing PARCELS : PARser for Content Extraction and Logical Structure Using Inter- and Intra- Similarity Features. *2004/2005 Honours Year Project Report, National University of Singapore*, , 2005.

Lee, C. H., Kan, M.-Y., & Lai, S. (2004). Stylistic and lexical co-training for web block classification. *Proceedings of the 6th annual ACM international workshop on Web information and data management - WIDM '04*, , 2004, 136.

Needleman, S., & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, *48*(3), 1970, 443–453.

Perkowitz, M., & Etzioni, O. (1995). Category translation: Learning to understand information on the internet. *International Joint Conference on Artificial Intelligence*, Vol. 14 (pp. 930–938), 1995: Citeseer.

Soderland, S. (1999). Learning Information Extraction Rules for Semi-structured and Free Text. *World Wide Web Internet And Web Information Systems*, *44*, 1999, 1–44.

Zhang, K., & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, *18*(6), 1989, 1245–1262.

How do you use the Internet to keep up to date with topics you are interested in?

Response	Freq	%
I visit the related sites of such topics regularly.	17	63%
I use RSS feeds and a feed reader to keep up-to-date.	6	22%
The sites I frequent usually have a "Subscribe to newsletter" function.	0	0%
I don't use the web for this purpose.	1	4%
Other	3	11%

Table 1: Survey results for Question 1