Honours Project Report

Querying the Deep Web: Result interpretation

By John Muntunemuine

Supervised by: A/ Prof. Sonia Berman

Department of Computer Science
University of Cape Town
Private Bag, Rondebosch
7701, South Africa

03 November 2008
Abstract

With an ever-increasing amount of data on the Deep Web, the consumers and providers become more concerned about how efficiently the data can be accessed and managed. This project report describes the implemented Result interpretation tool that interprets data stores on the Deep Web, and helps the consumers and providers access and easily manages the data. This tool extracts data from various Web database result pages and then merges the results together under a global schema. The report also describes the performance of the system on the basis of the various tests run. From this testing, it results that 89% of HTML pages are fully extracted, 4% of HTML pages are partially extracted and 7% of HTML failed to be extracted due to the difficulty of accessing the data region.

The last part of the report gives an overall conclusion of the project and suggests future work to improve the Result interpretation on the Deep Web.
Acknowledgement

I would like to thank my supervisor, A/Prof. Sonia Berman for her support, guidance, valuable comments and knowledge and for giving me the opportunity to accomplish this Honours thesis.

My special thanks to Dr Anet Potgieter for her financial support. Without this it would be very tough year.

My special thanks to my brother Bopwa Kumesu for his financial support. Without this I would not study at University of Cape Town

My special thanks to my sweet wife Sifa Djuma for her great support and endless love during my study period. Without you it would be impossible.

My special thanks to my parent for their support.

I would also like to thank Martha Kamkuemah for her valuable input and suggestion.
Dedication

To my brother Bopwa Kumesu his valuable support and guidance throughout my graduate studies.

To my kids Daniella Bikumbane Muntunemuine and Jonathan Bopwa Muntunemuine.
# Table of Contents

Abstract .................................................................................................................. i
Acknowledgement ................................................................................................. ii
Dedication ................................................................................................................ iii
Table of Contents .................................................................................................... iv
List of Figures .......................................................................................................... vi
List of Tables ............................................................................................................ vii

1. **Introduction** ................................................................................................... 1
   1.1. Motivation....................................................................................................... 1
   1.2. Success Criteria............................................................................................... 2
   1.3. Report outline ............................................................................................... 2

2. **Theory / Background** .................................................................................... 3
   2.1. Introduction of Background ......................................................................... 3
   2.2. Overview ....................................................................................................... 3
   2.3. Related Work ............................................................................................... 3
       2.3.1. Response Page Processing from the Deep Web................................. 3
           2.3.1.1. Result displays by piece ......................................................... 4
           2.3.1.2. Retrieving all results with default query ................................. 4
           2.3.1.3. Query submitted with a field missing ..................................... 4
           2.3.1.4. Error Handling ..................................................................... 4
       2.3.2. Detection and removing of Result Duplication .................................... 5
   2.4. Result Processing .......................................................................................... 6
   2.5. Web Data Extraction ..................................................................................... 6
   2.6. Data integration ........................................................................................... 7
   2.7. Summary of Chapter .................................................................................... 8

3. **Problem Definitions and Goal** ...................................................................... 9
   3.1. Result interpretation ..................................................................................... 9
   3.2. Goal .............................................................................................................. 9

4. **Design** .......................................................................................................... 10
   4.1. Design specification ..................................................................................... 10
   4.2. Extraction the Data Structure from an Airline Web .................................. 10
       4.2.1. Find Target HTML pages ............................................................... 10
       4.2.2. Data extraction ................................................................................. 10
       4.2.3. Distillation and improving Data ....................................................... 11
       4.2.4. Data mapping ................................................................................... 12
       4.2.5. Data integration ............................................................................... 12
   4.3. System Overview .......................................................................................... 14
   4.4. Sequence of events in Result interpretation .............................................. 15
   4.5. Use case ...................................................................................................... 16
   4.6. Conceptual class diagram .......................................................................... 16

5. **Implementation** ............................................................................................ 19
   5.1. HTML Data Extraction .............................................................................. 19
       5.1.1. Overview ............................................................................................. 1
       5.1.2. Preprocessing HTML pages ............................................................ 19
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1.2.1. Fetching HTML Web page</td>
<td>20</td>
</tr>
<tr>
<td>5.1.2.2. Repair bad HTML syntax</td>
<td>20</td>
</tr>
<tr>
<td>5.1.2.3. Generating parse Tree</td>
<td>21</td>
</tr>
<tr>
<td>5.1.2.4. Page sent to Extractor</td>
<td>25</td>
</tr>
<tr>
<td>5.2. Data extraction methodology</td>
<td>26</td>
</tr>
<tr>
<td>5.2.1. Region extraction: indentifying import region</td>
<td>28</td>
</tr>
<tr>
<td>5.2.2. Data extraction: discovering important region</td>
<td>29</td>
</tr>
<tr>
<td>5.3. Data integration</td>
<td>31</td>
</tr>
<tr>
<td>6.1. System Testing</td>
<td>35</td>
</tr>
<tr>
<td>6.1.1. Testing HTML Parser</td>
<td>35</td>
</tr>
<tr>
<td>6.1.2. Testing Data Extractor</td>
<td>35</td>
</tr>
<tr>
<td>6.1.3. Testing Data integration</td>
<td>36</td>
</tr>
<tr>
<td>6.2. System Performance</td>
<td>36</td>
</tr>
<tr>
<td>6.3. Findings</td>
<td>36</td>
</tr>
<tr>
<td>7. Conclusion</td>
<td>40</td>
</tr>
<tr>
<td>7.1. Report Summary</td>
<td>40</td>
</tr>
<tr>
<td>7.2. Project Assessment</td>
<td>41</td>
</tr>
<tr>
<td>8. Future Work</td>
<td>43</td>
</tr>
<tr>
<td>References</td>
<td>44</td>
</tr>
<tr>
<td>Appendix A: Airline Names</td>
<td>47</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1: Data extraction from foreign database ......................................................11
Figure 2: Filtering data extracted ..........................................................................12
Figure 3: Integration of data on global interface ......................................................13
Figure 4: Overall view of Result interpretation from Deep Web .........................14
Figure 5: Sequence Diagram in Result interpretation ..........................................15
Figure 6: User case of Result interpretation system ..............................................16
Figure 7: Class diagram in Result interpretation ....................................................17
Figure 8: HTML document preprocessing .............................................................20
Figure 9: An example of SAA airline HTML result web page ..............................22
Figure 10: An HTML fragment of SAA airline result web page ..........................23
Figure 11: Fragment HTML Tree of SAA airline result page ...............................24
Figure 12: Snip of Java code extractor algorithm ................................................25
Figure 13: Overall of system design .....................................................................27
Figure 14: Illustration of <Span> tags enclosed in <TD> tags of table .................28
Figure 15: Example of pure data and data noise extracted from Mango ...............30
Figure 16: Data integration on global interface ......................................................31
Figure 17: Result extracted and displayed on unified interface ............................33
Figure 18: Relation between page numbers and extraction time .........................37
Figure 19: Relation between result Size in KB and extraction time ......................38
Figure 20: Pie chart of data extraction ..................................................................38
List of Tables
Table 1: Relation between numbers of page extracted and time ....................37
Table 2: Relation between result sizes and extraction time ..........................38
Table 3: Information on total number of HTML page tested for extraction...39
Table 4: Deep Web airline names used for system testing..........................46
1. Introduction

With the rapid development of the World Wide Web, there are more and more Web databases available for users to query and access. This rapid development of the World Wide Web has dramatically changed the way in which information is managed and accessed; and that information on the Web has covers all aspects of human activities. This opens the opportunity for users to benefit from the available data. The data information in the Web still continues to increase at striking speed. This requires an appropriate approach in data accessing which helps consumers and providers to manage the huge quantities of information on the web.

The Web can be split into two parts that are: the Surface Web and the Deep Web. Surface Web refers to the Web pages that are static and linked to other pages. Traditional search engines create their indices by crawling Surface Web pages. Deep Web refers to the Web pages created dynamically as the result of specific searches. Data in the Deep Web are hidden behind the Web interfaces and none of the major search engines index those pages [1]. However, data in the hidden Web are stored in the databases and are accessible by issuing queries.

This project focuses on querying the Deep Web, specifically on the airlines domain. The Deep Web refers to the accessibility of different Web databases through querying interfaces on the World Wide Web. The Deep Web query system presents a single interface to users in order to query multiple hidden databases via their separate front-end query interfaces. Such a system will hopefully reduce the amount of searching time and simplifies the way users might search for information hidden in multiple databases, it is freeing the users from query details of an individual data source.

A Deep Web system usually has two components: Query formulation and Result interpretation. This report concentrates on the Result interpretation component.

1.1 Motivation

The huge amount of information on the Web is not easily accessible due to the scale, heterogeneity and unstructured mature of the Web data. Thus, to overcome this problem, a feasible way is to write a parser that can parse an HTML page from Web sources and extract data. The extracted data can then be inserted into databases for further utilization.

Consider the domain of airline flights, users have difficulties in the first instance to find the right airline databases and then query these databases is very tedious and time consuming due to the large scale of Web databases in the Deep Web. To enable effective access to databases on the Web, this project developed a tool that helps users to send a query to multiple databases and
then extracts and integrates the results received from different data sources and presents the results to users in a single global interface.

1.2 Success Criteria
The following are the set of success criteria for this project. If all criteria are met, it would indicate the project is successful.

- Develop a unified interface that is able to send a user query to various Web databases and return responses pages to a parser system.
- The developed a result interpreter system should be able to parse those response pages from various Web databases, then extract the results and merge them into the global interface for users’ processing.

1.3 Report Outline

Chapter 2: Theory /Background
This chapter discusses the literature on querying the Deep Web; specifically, it focuses on data extraction from various HTML Web page and data integration.

Chapter 3: Problem definition
This chapter identifies the problem of result interpretation in the Deep Web and gives the goals of the result interpreter system.

Chapter 4: Design
The Design chapter outlines the design of the system and software engineering model of implementation.

Chapter 5: Implementation
The Implementation chapter discusses in depth how the system was implemented and the tool used.

Chapter 6: Evaluation and Testing
This chapter describes the tests performed on the system and discusses the result produces on these tests.

Chapter 7: Conclusion
This chapter presents a general summary of the system and draws conclusions on the strengths and weaknesses of system.

Chapter 8: Future Work
The chapter suggests future work in database systems and how the result interpretation in querying the Deep Web can be further improved.
2. **Theory / Background**

2.1. **Introduction of Background**

The Deep Web refers to the accessibility of different web databases through query interfaces on the Word Wide Web. A Deep Web query tool presents a single interface to users, and upon submission of a query via its interface, the tool submits equivalent queries to many hidden databases via front-end query interfaces and then extracts and merges the results received from different web-sources.

The advantage of this tool in the airline domain for example, is to prevent the users querying from each airline among many airline websites, which is time consuming. The second advantage is that the tool will present a simple and friendly query interface to users and collect data from hidden airline databases and then return a single interface of results for user-processing.

A Deep Web Tool usually has two components: Query formulation and Result interpretation.

*The Query formulation* part will involve schema integration, and formulating the query to be sent to the various web resources. The query formulation part can be developed separately from the result interpretation part.

*The Result interpretation* part extracts the results from pages returned by different web databases, and then merges them together into global interface for the utilization by users. This part requires the appropriate methods for data extractions and their merging into unified interface.

2.2. **Overview of Chapter**

The remainder of this chapter is organized as follows. Section 3 discusses the related work in the Deep Web. In section 4 we discuss the result processing approach in order to extract the relevant pieces of information out of returned pages. Section 5 presents methods used for data extraction in the Deep Web. Section 6 discusses data integration into a unified interface. Finally, section 7 concludes the chapter and gives an overall summary.

2.3. **Related Work**

2.3.1. **Response Page Processing from the Deep Web**

Once a query is sent to relevant websites, the next step is to retrieve information from those target sites. Several cases are possible.
2.3.1.1. Results display by pieces
Handling of results displayed piecemeal has been studied by Stephen W. Liddle, David W. Embley, T. Del Scott and Sai H. Yau [1]. In this case, the web site returns a bit at a time, showing perhaps 2 or 4 results per page. The system will provide a button or a link URL to get to next page until the last page is reached. One approach [1] treats all the consecutive next pages from the returned page as part of one single document by concatenating all the pages into one page. The system activates this process if the returned page contains a button or link indicating next or more. In this way, the system constructs a logical page containing all the data.

2.3.1.2. Retrieving all results with default query in the case of small database
In the default query approach, the system may have retrieved all or least a significant percentage of the data before submitting all queries; the reason behind this is, many forms have a default query that contains all data available from the website. Stephen W. Liddle, David W. Embley, Del T. Scott and Sai Ho Yau [1] discussed this issue in depth. The problem found in a default query (with a default query the user is not necessarily selecting or filling fields with information) is that, sometimes it does not retrieved all data and every set of data returned may be some particular subset of the overall database. In this case the problem is solved by sampling the database and finding the data not already returned by the initial default query, the user will continue the process of submitting the query until as much data as possible is retrieved. If the additional queries all return data that is equal to or subsumed by the data returned for the initial default query, we need not query with all combinations.

2.3.1.3 Query submitted with a field missing or No-Result Found
For a query submitted with a field missing, or for no-result found, the system must automatically detect the problem and solve it. In the case that a required field is missing, the system will search for a message such as “Required field is missing”. This kind of error requires the intervention of the user using the system. The user will be required to fill the relevant fields of the interface and again submit the query to the system. In the case that no result can be displayed to the user query, the system could search for message like “No matching result could be found”.

Both error cases have been discussed by Stephen W. Liddle, David W. Embley, Del T. Scott and Sai Ho Yau [1]. It is more reliable to observe that the size of the information returned after removing miscellaneous header and footer information is normally very small if there was an error- usually a constant small value for all queries that return no result [1].

2.3.1.4 Error Handling
During the response pages processing from the Deep Web, the following errors may be encountered:
In the case of network failure, a server down, or HTTP errors, the system will notify the user by an error message and the type of error and then abort the current operation.

The errors that might be in an HTML page result might be easily recognized automatically, like HTTP 404. Other error messages are hard to recognize. These may be embedded within a series of tables, frames or other types of HTML division. Users can sometimes understand the messages, but automated understanding is very hard.

The results coming from an HTML page may contain duplication of information, which we should discard. Section 3.2 discusses how the system detects and solves the duplication error.

In certain circumstances, the server may require authorization information for logging on to the system.

2.3.2. Detection and removing of Result Duplication

Once the query is sent to the relevant web-sources, the data retrieved is placed into a repository discussed in [1, 3]. Data retrieved for multiple submissions of a form may contain duplication; the system eliminates this duplication of data before placing the result in the repository by using the detection mechanism described in [2], which is highly effective for finding duplicate sentences over a large set of textual documents. The system systematically analyses the data returned from a Deep Web query, then calculates the hash value for each result and then removes the duplication [1].

The data retrieved from behind web forms is usually displayed as paragraphs separated by the HTML paragraph tag <p>, as rows in a table separated by <tr> </tr> tags, or as blocks of data separated by the <hr> horizontal rule tag. Stephen W. Liddle, David W. Embley, Del T. Scott and Sai Ho Yau in [1] proposed a way of dealing with a special tag called the sentence boundary separator tag in order to adapt the copy detection system for collection of records.

During the duplication detection process, the system inserts this special tag into a retrieved web document around certain HTML tags that most likely delimit the duplicate record. The tags chosen for this treatment include </tr>, <hr>, <p>, </table>, </blockquote> and </html>. If none of the above tags except </html> appears in the document, the whole document is considered to be a single record. The idea above of handling duplicate recognition and elimination has been discussed in more detail in [1].
2.4. Results Processing

This section deals with how the results are being processed from a web form once the query has been submitted by the user and HTML pages returned. A parser [6] will analyze different formats of each html page returned by web databases in order to extract the relevant pieces of information out of forms. Once extraction of the data from different web-sites is done; the next step is to merge those data into a single response page called a global interface; this idea is detailed in the next section (2.5) of this chapter.

Result processing can be split into three components, which are:

- **Result extraction**: this component will identify and extract the relevant results from the response pages returned by web databases.
- **Result annotation**: this component will append the proper semantics for the extracted result.
- **Result integration**: this part integrates the results extracted from different web databases into a single response page (unified page).

2.5. Web data Extraction

Web data extraction from the Deep Web has been tackled by many people in the literature e.g. [5]; it seems that more work must still be done in this area. Raghavan, S. and Garcia-Molina at Stanford University [3, 4] developed the “Hidden Web Exposer (HiWe)” system that builds a Deep Web crawler that automatically parses, processes and interacts with form-based search interfaces. Because of the formidable challenges to a fully automatic process, HiWE assumes that crawls will be domain specific and human assisted. Although HiWE must start with a user filling in the form for a search task, HiWE learns from successfully extracting information, and updates the task description database as it crawls. Besides an operational model of a Hidden Web Crawler, other more interesting contributions are:

- The label matching component used for matching labels entered on the form to those labels in the Label Value Set table.

- Internal form representation: the crawler breaks up a query form into several information pieces. The form is represented by F= ((E1... En), S, M) where [E1... En] represents a set of n elements. S is the submission information associated with the form, and M is metadata information about the form. Each element of the set E has two pieces of information called domain Dom (Ei) and Label (Ei). Domain refers to a set of values Ei can take on and Label is the description associated with a domain value.

- Task-specific database, the HiWE crawler uses a task-specific database. This database stores all relevant information that helps the crawler to formulate search queries relevant to a particular task.
• Response analysis: this component stores the page result in the crawler’s repository.

Similar work has been discussed by Stephen W. Liddle et al [1]. They proposed a way to extract the data behind web forms. Their main contribution reveals how to retrieve the data behind a particular HTML form; how to process a result page returned by a form submission. This includes, for example, error detection.

2.6. Data Integration

Data integration is the problem of combining data from various web databases sources, and providing users with a unified view of data [15, 16, and 17]. One of the main tasks in designing a data integration system is to establish the mapping or relation between the web database sources and a global schema, which must be taken into account in formalizing a data integration system.

Others have explored this point of Deep Web data integration and many solutions are discussed. However they state that the challenging part remains “schema matching” for discovering semantic correspondences of attributes across heterogeneous sources. Bin He and Kevin Chen-Chuan Chang [7] addressed the “problem of automatic matching process” which integrated the DCM (Dual Correlation Mining Algorithm) framework with an automatic interface extractor. Such system integration turns out to be nontrivial as automatic interface extraction cannot be perfect, it will introduce erroneous extraction, which challenges the performance of the subsequent matching algorithm.

However, Stephen W. Liddle, David W. Embley, Del T. Scott and Sai Ho Yau [1] proposed that “it is necessary to automate extraction and integrate information data from different web databases” [1]. Related work of data integration has been discussed by S.Raghavan and H. Garcia-Molina [3, 4]; their Crawling the Hidden Web gives more significant contribution to the data integration in the development of Deep Web data integration. Their tool automatically traverses the Web graph, retrieves pages and builds a local repository.

Maurizio Lenzerini [15] discussed in his “theory of data integration” the main components of a data integration system that are a Global Schema, web databases Sources and mapping. He formalizes a data integration system I in a triple (G, S, M) where G is the global schema, expressed in a language LG over an alphabet AG; S is the database sources, expressed in a language LS over an alphabet AS; and M is the mapping between G and S, constituted by a set of assertions of the forms {qS, qG} or {qG; qS} where qS and qG represent two queries respectively over the source S and over the global schema G.
2.7. Summary of Chapter

In this chapter, the literature aspect of the Deep Web tool has been discussed. Such a tool contains two components: a Query formulation and a Result interpretation.

Query formulation: involves schema integration and formulates the query to be sent to the various web-databases sources.

Result interpretation: deals with parsing, data Extraction and data Integration of response HTML pages returned by a Deep Web.

The discussion in this chapter was concentrated on the result component of the Deep Web query. Once the query to the Deep Web is submitted by a user, the system must find the relevant fields of records and match them to fields of the global schema, then extract field values into a repository and then display this as an integrated result. In addition, problems to handle include errors in the results; the system must provide a mechanism of handling these errors before integrating the result into global consolidated result. In the case of a missing field error, user intervention is required. With a missing field, the system won’t be able to retrieve the information from the Deep Web, so the default query case discussed in section 3 of this chapter above, is inferior approach compares to other approaches.
3. **Problem definition and Goal**

The World Wide Web has become a significant place where a huge amount of data is stored. To help consumers and providers manage these huge quantities of data, it is useful to extract these data from various Web sources and either store them in local databases where the users can query and access them or display them into a data unified interface. This technique is referred as querying the Deep Web. We have mentioned in the Introduction chapter that the Deep Web tool is made up of a Query formulation component and a Result interpretation component. This chapter defines the problem and gives an overview of the goal in the Result interpretation component.

### 3.1. Result interpretation

The Result interpretation, also called Result processing component, extracts the query results from various Web databases and integrates the results together under a global schema. This component is described as follows:

- **Result extraction**: identifies and extracts the pure results from the response pages returned by Web databases.
- **Parser**: parses the response pages returned by Web databases in order to extract the results.
- **Result integration**: integrates the results extracted from various Web databases together under a global schema or a unified data interface.

The result interpretation problem in the airline domain has been tackled in [22] by Travel Start Website. This Website allows a user to send a query to many airline Web databases, however the results presented to user are inconsistence and incomplete. Our system interprets the data as given in the Deep Web and presents it to user without alteration.

### 3.2. Goal

Result interpretation plays a crucial role in the Deep Web tool. Its role is to parse the response pages returned by the Deep Web and then interpret the results extracted in each response page and map the result to a defined template.

Without the Result interpretation, the data in the Deep Web would not be easy to understand due to their huge amount, heterogeneity and lack structure. Result interpretation presents data with well a single, easily understandable structure and helps users to access and manage them in the right order.
4. Design

4.1. Design Specification
The Deep Web tool is broken down into two components: Query formulation and Result interpretation. Martha Kamkuemah was responsible for designing the Query formulation system. This report considers only the Result interpretation system.

This chapter presents an overview of the system design for results interpretation in the Deep Web. Once a query is sent to various databases sources using the unified interface, and the HTML pages returned. The block of data needs to be extracted after parsing HTML pages with a parser and then integrated and displayed on a global schema for further processing by the user. This section gives an overview of data extraction and merging data into a global schema.

4.2. Extracting the Data Structure from an Airline Web
In order to extract the data structure from a web-site, we need to identify the relevant fields of information or targets; to achieve that, there are five problems that need to be solved: localizing of HTML pages from various web databases sources; extracting the relevant pieces of data from these pages; distilling the data and improving its structure; ensuring data homogeneity (data mapping); and merging data from various HTML pages into a consolidated schema (data integration problem).

4.2.1. Find target HTML pages
The websites contains two types of HTML pages: target HTML pages that contain the data that must be extracted and navigational HTML pages that contain hyperlinks pointing to target pages or other navigational pages. An automated system such as an HTML parser can be used to retrieve target pages from a Web site.

4.2.2. Data extraction
Once the relevant HTML pages are returned by various web databases, the system needs to localize the specific region having the block of data that must be extracted without concern for the extraneous information. This can be accomplished by parsing each HTML page returned from different databases. Java has a parser method that parses HTML pages, so this method was used for parsing HTML pages. Figure 1 below explains how the data extractions are processed from an HTML document.
Figure 1: Data extraction from foreign database

The HTML pages returned by foreign databases are fed one by one to the HTML parser that parses and builds a tree structure. This tree is submitted to the extractor system, which generates the HTML page extracted and then transfers the extracted page to the local database system for storage and later processing.

4.2.3. Distillation and improving Data

The data extracted from the response pages returned by various database sources might be duplicated and unstructured. To handle these problems of duplication and lack of data structured, the data extracted needs to be filtered in order to remove duplication and unstructured elements, and then the data result must be presented in structured format. This is accomplished by passing each HTML data extracted through a filtering system; this processes the data and discards data that is duplicated. Figure 2 describes how the structure of data can be improved.
The HTML data extracted and stored in local database is transferred to an HTML filter for removal of duplication in the data. The resulting blocks of data are either sent again to the local database or displayed on the integrated interfaces (global interface) for further processing by users.

4.2.4. Data Mapping
The data fields from various airline database sources may be named differently. A mapping function is required for a standard format and improves the quality of HTML data extraction. This mapping function takes data fields as parameters and feeds them into a unified interface.

4.2.5. Data integration
The final step in the problem of data extraction is data integration. Data integration is the process of presenting data from various Web database sources on a unified interface. Data integration is very crucial to the users for the reasons that users want to access as much information as possible in less time. The process of querying one airline at a time is time consuming and sometimes very hard; especially when the users do not have enough information about airline sources. Figure 3 below gives an overview of data integration of the global interface system.
The data extracted from each airline database source is sent to a single interface as shown in Figure 3 above. The single interface presents unified data from various sources to the users for utilization.
4.3. System overview

Figure 4 below illustrates an overview of result interpretation in the airline domain on the Deep Web. Each HTML page returned in the Deep Web as shown in Figure 4 is submitted to the HTML parser in order to be parsed. The HTML parsed is then transferred to the extractor for data extraction. Data extracted is then merged and sent to a filter system for removing data duplication and extracting structure. The data structured is now displayed on the resulting unified interface for utilization by users.

![Diagram of System Overview]

**Figure 4: Overall view of Result interpretation from Deep Web**
4.4. Sequence of events in Result interpretation

Figure 5 below illustrates the sequence of events in Result interpretation. At the start of result interpretation events, the run time system (Martha Kamkuemah’s part) produces the sequence of HTML response pages returned by various foreigner databases as shown in the sequence diagram below.

![Sequence Diagram in Result interpretation](image)

This sequence of HTML response pages returned is then transferred to the HTML parser for parsing. The output HTML pages parsed are then fed to the Extractor to extract the data. The extracted data is sent to integrator system that integrate it a on unified interface.

The communication between the Run time system and the HTML parser is asynchronous in the sense that the Run time system produces the HTML pages that are transferred to the parser system.
4.5 Use case
The Run time subsystem from the query formulation component processes a user query sent to the Deep Web and obtains HTML result pages. The Result interpretation system fetches these pages from a location specified by the Run time subsystem and parses these pages, and then extracts data. This scenario is illustrated in use case diagram shown in Figure 6 below.

![Use case diagram](image)

**Figure 6: User Case of Result interpretation System**

The Run time subsystem is a user that interacts with the Result interpretation system because it supplies the information to the Result interpretation system.

4.6 Conceptual class diagram
The UML class diagram shown in Figure 7 below illustrates the design of the Result interpretation system. The system has a generic Parser where Kulula, Mango, One Time, SAA, Kenya and British classes are inherited from the Parser class. The Parser class is the generalization class that parses each HTML document from the Deep Web of airline domain. The HTML Document parsed is then used to extract the data information. The data extracted from HTML pages is then sent to the unified result interface for instantaneous access and to the local database for later access. Each sub-class in the system is made of behaviors from the parent class (Parser class), and its own behaviors.
Figure 7: Class Diagram in Result interpretation
The integrator class contains the following methods:

```java
protected void doGet(HttpServletRequest request, HttpServletResponse response)
protected void doPost(HttpServletRequest request, HttpServletResponse response)
public String getServletInfo()
static Reader getReader(String fileName)
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
```

The “doGet” method gets the extracted results from the Extractor. The “do Post” method posts extracted data to the HTML unified interface. The “getServletInfo” method returns a short description of the unified interface. The “get Reader” method reads the response pages returned by Deep Web of airlines. The “process Request” method is used by the doGet method and the doPost method in order to process “get” data extracted and then post these data to the unified interface.
5. Implementation

5.1. HTML Data Extraction

5.1.1. Overview
The implementation of an HTML Data extractor describes the techniques used to parse, and extract data information from HTML pages returned by different airline web databases sources, once a query is sent from a global or unified interface to the Deep Web.

The contents of HTML airline result pages contain semi-structured data information. This semi-structured information makes the HTML pages more challenging to parse and to extract the specific data from a specific region of an HTML page. The second challenge comes with different approach in designing HTML pages that vary from one airline web page to another, in terms of tags and structures of data storage. The blocks of data found in HTML are pure data and data noise.

Our HTML data extractor system takes semi-structured data from HTML pages returned by airline web databases and converts these data into structured records in order to parse. The system parses each HTML page returned by foreign Web databases, and then extracts the require data information from the region of that page containing the data records. The HTML page results returned by the Deep Web of airlines contain great number of table tags, where the needed data information is stored. For this reason, the extractor system processes the rows-columns region of tables in an HTML document in order to pull out the block of data between table-columns i.e. the flight number, departure time, arrival time, and the prices.

5.1.2. Preprocessing HTML pages
Once a query is sent by a user to an airline’s Deep Web databases, the HTML pages are returned and passed to the parser system in order to analyze and repair the bad syntax structure of HTML documents and then extract the required data on the page. The process is performed in three steps as shown in Figure 8 below. The first step consists of the retrieval of an HTML document once this page is returned by the foreign Web airline database and possibly its syntax reparation. The HTML Editor Kit, in the second step, is responsible for generating a syntactic token parse tree of the repaired source document; and the third and last step is responsible for sending the HTML document to an extractor system for extracting the information that matches the user need.
5.1.2.1. Fetching HTML Web page
The Query formulation part (Martha Kamkuemah side) is responsible for generating a set of rules that describe the list of interface functions and parameters as well as how they are used to fetch a remote HTML document from a given airline Web source. The list of interface functions includes the declaration to the standard library routines for establishing the network connection, issuing an HTTP request to the remote Web server through an HTTP Get or HTTP Post method, and fetching the corresponding Web page.

Once the set of rules for fetching remote HTML document is set up and the HTML document is retrieved, the next step is to feed this HTML document to the HTML Editor Kit function in order to repair bad syntax and generate the HTML parsing tree.

5.1.2.2 Repair Bad HTML Syntax
As soon as the HTML document is fetched and passed to the HTML Editor Kit function [21], the syntax repairing thread begins. It repairs bad HTML syntax of the document by inserting missing tags, and removing useless tags, such as a tag that either starts with < Pr which is an end tag that has no corresponding start-tag. It also repairs end tags in the wrong order or illegal nesting of elements. It describes each type of HTML error in a normalization rule. The same set of normalization rules can be applied to all HTML documents. The HTML Editor Kit can clean up most of the errors listed in an HTML document.
5.1.2.3. Generating Parse Tree

Once the HTML document is free of errors and bad formatting, the clean HTML document is submitted to the parser, which parses the block of tags in the HTML page containing the pure data information to be extracted and data noise that are irrelevant data. The airline source HTML documents that have been parsed contain many tags with tables nested inside of other tables, making the tree structure complex. Each tag table contains a sequence of elements or units, called syntactic tokens. Each token identified represents a sequence of characters that can be treated as a single syntactic entity. The tree structure generated in this step has each node representing a syntactic token, and each tag node such as TR represents a pair of HTML tags: a beginning tag <TR> and an end tag </TR>. Different languages may define what is called a token differently. For HTML pages, the usual tokens are paired HTML tags (e.g., <TR>, </TR>), singular HTML tags (e.g., <BR>, <P>), semantic token names, and semantic token values.

**Example 1:** Consider the South African Airline response page returned from their website [18] (see Figure 9 below), and a fragment of HTML document represent in Figure 10 below. Figure 11 below shows a portion of the HTML tree structure, corresponding to the fragment of HTML document in Figure 10, which is generated by running the HTML tree parser on the South African Airline response page returned from the website.

This portion of the tree structure contains the main Table that nests three other Tables called sub-tables. The third sub-table in the main Table embodies a new sub-table that records the pure data information and data noises. The structure of the tree has the following types of tag nodes: TABLE, TR, TD, SPAN, and a number of semantic token nodes at leaf node level, such as flight number, departure time, arrival time, aircraft, saver (in ZAR), classic (in ZAR), select (in ZAR) and premium (in ZAR). Examples of these token in Figure 4 are: SA378, 05:45, 07:45, 738, ZAR 684, ZAR 1151, ZAR 1835, and ZAR 2405 etc...

Important to note is that for each syntactic token, the parse tree is organized as follows. All non-leaf nodes are tags and all leaf nodes are text strings, each localized between a pair of tags. We have defined a method that extracts these text strings between pairs of tags.

The result page in Figure 9 below has been extracted from South African airways website, and shows the flights details from Cape Town to OR Tambo international airport in Johannesburg on 30th September 2008 [18].
Figure 9: An example of SAA airline HTML result page, extracted from [18]

The piece of HTML code (Figure 10) below has been extracted from South Africa Airways result page above (Figure 9), and shows how the bocks of data (flight details) are recorded in the row-columns of table tags.

```html
<tr>
    <td class="booking_content_light" background="/images/tables/booking_tbl_lt.gif" width="10">\n
    <td class="booking_content_light"><span id="0.0_flightNumber">SA 378</span></td>
    <td class="booking_white_spacer" width="1"></td>
    <td class="booking_content_light"><span id="0.0_departureTime">05:45</span></td>
    <td class="booking_white_spacer" width="1"></td>
    <td class="booking_content_light"><span id="0.0_arrivalTime">07:45</span></td>
    <td class="booking_white_spacer" width="1"></td>
    <td class="booking_content_light">738</td>
</tr>
```

(Figure continues …)
Figure 10: An HTML fragment of SAA airline html result Web page.
Figure: 11 Fragment HTML Tree for SAA Airline result Web page.
5.1.2.4. Page Sent to Extractor

Once an HTML document has been successfully parsed, the next step is to transfer the page to an extractor system (method) so that the block of pure data can be successfully extracted from the HTML page and data noise discarded for the reason that they are irrelevant to the users need and to our data global interface. The snip of java code below in Figure 12 shows the implementation of the extractor method:

```java
ElementIterator it = new ElementIterator(doc); // use iterator class to iterate in html doc
ejavax.swing.text.Element element; // use swing text element library
int count = 0;
while ((element = it.next()) != null) // check if html document still has element
{
    AttributeSet attributes = element.getAttributes(); // get element attribute in html doc
    Object name = attributes.getAttribute(StyleConstants.NameAttribute); // get name attrib
    // identify Tag in html document and catch TD tag
    if ((name instanceof HTML.Tag) && (name == HTML.Tag.TD))
    {
        element = it.next(); // iterate in html document and search next element
        StringBuffer text = new StringBuffer();
        int startOffset = element.getStartOffset(); // get start of element
        int endOffset = element.getEndOffset(); // get end of element
        int length = endOffset - startOffset; // length of element
        text.append(doc.getText(startOffset, length)); // Extract data
        char ch = text.charAt(C); // filtering pure data and discarded data noise
        String str = text.substring(0);
        if (text.length() <= 9 && ch == 'f' && ch == 'f' && str.contains("<"))
        {
            int check = str.length();
            if (check > 5) // Further Removal of data noises
                myVector.addElement(String.valueOf(text)); // add pure data to vector further processing
        }
    }
    count++;
}
```

Figure 12: Snip of Java code extractor algorithm.

The snip of Java code above takes a parsed airline HTML document, processes it and then extracts pure data and discards data noise. The Element-Iterator class iterates over the elements in the HTML document until end of the list. Now, each element found in HTML is submitted to Attribute-Set class in order to get the attributes name of that element. Once the name of the element is established, the next is to verify if this name is a tag and if it is a TD tag. If the last condition is met, the next () method in the Iterator-class is called to catch that element having a TD tag and assign it to the object of the Javax swing text class. This object of the Javax swing text is used with the getStartOffset () method and the getEndOffset () method to fetch the offset from the beginning of document that this element began with or from the beginning of document that this element ended. The last
step in the code is filtration of the block of data in the document; this is performed by the getText (startOffset, length) method. The inner -if statements in the code removed data noises from set of data block extracted in the HTML document.

5.2. Data Extraction Methodology

The crucial part in data extraction is to explore and specify the structure of the retrieved document in a declarative extraction algorithm. In processing an HTML document, the data to be extracted consists of analyzing the HTML DOM trees and tags of the response pages returned by airline web databases. The system interacts with the user to supplier the semantic token for querying the airline deep web and hierarchical structure from the global interface.

Once an HTML page is returned from airline web databases and parsed, as shown in Figure 13 below, the data extraction process involves three steps. These are: identification of the region having blocks of data; identification of blocks of data from the discovery region; and removal of data noise and data duplication from the block of data.

Step 1: Identifying the region that records blocks of data (semantic token) to be extracted from an HTML page

Though different airline web pages are designed by different people, these designers have a common consideration in placing the data region. The data records are the contents in focus on response pages, and they are usually centered on an HTML page and stored in row -column of tables (&lt;TR&gt; &lt;TD&gt; ... &lt;/TD&gt; &lt;/TR&gt;).

By investigating a large number of airline response pages returned by different airline web databases such as South Africa airways, British airways, Mango, Kulula, One Time and Kenya airways, two interesting facts were found. First, the flights data records are located inside of rows-columns of table region situated in the middle of an HTML page. Second the data records are localized either between open TD tags and close TD tags of table columns for some airlines (Example: Mango, Kulula, British airways, One Time, etc…) or between other open tags and closed tags that are localized between open TD tags and closed TD tags for others airlines (Example: SPAN tags in South Africa airways HTML page, etc…).
Step 2: Identifying the blocks data or semantic token of interested from a discovery region of an HTML page

The identification of blocks of data in the discovery region of an HTML page is carried out by walking on the tree structure generated by HTML DOM tree parser, specifically the TD tags of columns table because the data records are stored directly or indirectly between open TD tags and closed TD tags. Data records to be identified and extracted from the HTML source document are a set of semantic tokens that are combination of pure data of interest (i.e. flight number, departure time, arrival time, prices and places) and data noise (data irrelevant to user’s need).

Figure 13: overall of system design.
Step 3: Removal of data noise and data duplication

By investigating a large number of response pages returned by South Africa airways, Kulula, Mango, British airways, One Time and Kenya airways on the semantic token, a common fact has been discovered. The length of strings of data noise is generally larger than that of pure data. The pure data of different airline websites are almost the same and varies generally in the same proportion of length. This condition helps in filtering required data from data noise. Data duplications are handled by the removal of duplication in the data blocks.

5.2.1 Region extraction: identifying important region

Once a user submits a query to the Deep Web, the HTML pages returned are explored in order to identify the region that stores the data needed. By referring to the investigation that was made in step 1, this showed that the flights data are recorded in the table region of HTML; the extractor method searches for table region in an HTML page and walks on the TD tags of a Table where the blocks of data are recorded. The extractor computes the type and number of sub-region, and describes the structural layout of the region. For each type of region, such as the table region, the paragraph region, the text section region, the extractor pulls up the data information. From the response pages returned by seven airlines websites, the analysis shows that five of these airlines have their pages having the data strings enclosed between beginning tags <TD> and closing tags </TD> in the table region whilst two of these airlines have their pages having data string between beginning tags <span/> and closing tags </span/> and these tags <span/> / </span> are enclosed between opening tags <TD> and closing tags </TD> of table region. The last situation can be seen in Figure 14 below.

```html
<TD class='booking_content_light'><span id='0_2_flightNumber'>SA 304</span></TD>
<TD class='booking_white_spacer' width='1'/></TD>
<TD class='booking_content_light'><span id='0_2_flightDepTime'>06:20</span></TD>
<TD class='booking_white_spacer' width='1'/></TD>
<TD class='booking_content_light'><span id='0_2_flightDestTime'>08:20</span></TD>
<TD class='booking_white_spacer' width='1'/></TD>
```

Figure 14: Illustration of <Span> tags enclosed in<TD> tags of HTML table.

The Figure 14 above has been extracted from South Africa airways result page [18], and shows a piece of code in the column tag Table of HTML, recorded indirectly the flight details (flight number, departure time, arrival time).
5.2.2 Data extraction: discovering important data
The previous section identified important regions where the blocks of data are stored; this section concentrates on data extraction from the discovery regions. After the response pages were returned from the deep web and parsed, and then the data regions discovered, the data needs to be extracted from these regions. Two facts must be considered in order to extract data accurately. First, the blocks of data to be extracted are located in the nested table of the main table region in the middle of an HTML page for some airline sites (e.g. in the South African Airways case) or in other airlines sites, the blocks of data are located in the main table (e.g. One Time). The nested HTML table contains many rows, each enclosed in `<TR> ... </TR>`. The number of rows varies from one airline html page to another. In order to reach all number of rows in the table, the parser method walks on each tag TD enclosed between `<TR> </TR>` of the table and then pulls out the texts string until the last TD is reached in the last row of the table. At this point the proper data are not the only data pulled out but the data noise as well.

Second, data being extracted contains pure data and data noise (unnecessary data) that need to be removed from the set of data. Figure 15 below shows an example of pure data and data noise from the mango HTML page [19]. Because the length of data noise is longer than that of pure data, the parser restricts the string length condition to remove the data noise from the set of data. For the few instance of noise having the same length as pure data, the parser removes them by string matching and string pattern recognition. These conditions have been tested on many airline pages returned from different airline websites and the results produced were successfully.

Figure 15 below shows the distinction between data and noise. As shown by this figure, the length of flight number (JE122), prices (619.00), time (05h45 or 07h50) is shorter than data noise (Unlimited free changes, R40 meal voucher, 30 kg checked in luggage and free).
Figure 15: Example of pure data and data noise extracted from Mango HTML page [19].
5.3. Data Integration

Data integration is the process of merging the data from many airline sources into a global interface. This is a result of users frequently needing to access multiple airline sources in order to find the desired airline information, potentially a very time-consuming and labor-expensive process. To address this problem, an effective solution is to build a virtual integrated global interface over these airlines sources. Such an interface presents users with unified data for processing and provides uniform accesses to the sources, thus freeing them (the users) from the detail of individual airline searches.

This section presents the data integration mechanism for different airline sources once the data are extracted by the HTML DOM tree parser. Figure 16 shows the layout structure of data integration on the global interface. The data extracted from One Time airline, South Africa Airways, Mango, Kulula, Kenya Airways and British Airways are sent one by one to the data integrator system, which merges the result into a global interface and then displays them into a new HTML page. This Figure 16 presented six airlines, however it can be easily extended to iterate thus more airlines.

Figure 16: Data integration on global interface.
Figure 16 above shows how the data extracted from each airline source is integrated into the global interface. The integrator system displays these data on a global interface according to the order of arrival i.e. first come, first served.

The NetBeans 6.0.1 application that has been used to develop the system has a powerful tool that automatically generates an HTML page once data are extracted from different HTML pages, integrated and then sent to global interface page for user processing.

Figure 17 below shows the global interface with flight details of six airlines sources. For each airline, departure time, arrival time and the prices of trip are shown in the table. At bottom of each table, a link is provided for the booking process.

Welcome to Deep Web airline Site

Kulula Airline

<table>
<thead>
<tr>
<th>Departure</th>
<th>Arrival</th>
<th>Flight</th>
<th>Price</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>06:30</td>
<td>08:25</td>
<td>MN6301</td>
<td>R1156</td>
<td>R1209</td>
</tr>
<tr>
<td>11:30</td>
<td>13:25</td>
<td>MN6311</td>
<td>R886</td>
<td>R939</td>
</tr>
<tr>
<td>13:45</td>
<td>15:40</td>
<td>MN701</td>
<td>R766</td>
<td>R819</td>
</tr>
</tbody>
</table>

Booking with Kulula

Mango Airline

<table>
<thead>
<tr>
<th>Departure</th>
<th>Arrival</th>
<th>Flight</th>
<th>Total Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>05h45</td>
<td>07h50</td>
<td>JE122</td>
<td>ZAR 619.00</td>
</tr>
<tr>
<td>11h10</td>
<td>13h15</td>
<td>JE134</td>
<td>ZAR 659.00</td>
</tr>
<tr>
<td>12h35</td>
<td>14h35</td>
<td>JE146</td>
<td>ZAR 720.00</td>
</tr>
<tr>
<td>16h10</td>
<td>18h15</td>
<td>JE154</td>
<td>ZAR 659.00</td>
</tr>
<tr>
<td>21h40</td>
<td>23h45</td>
<td>JE172</td>
<td>ZAR 1499.00</td>
</tr>
<tr>
<td>18h05</td>
<td>20h10</td>
<td>JE176</td>
<td>ZAR 720.00</td>
</tr>
</tbody>
</table>

Booking with Mango

Figure Continues....
### Figure 17: Results extracted and displayed on unified interface.

<table>
<thead>
<tr>
<th>One Time Airline</th>
<th>Departure</th>
<th>Arrival</th>
<th>Flight</th>
<th>Price (ZAR)</th>
<th>Total Price (ZAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>06:45</td>
<td>08:45</td>
<td>IT 100</td>
<td>639</td>
<td></td>
<td>639</td>
</tr>
<tr>
<td>10:00</td>
<td>12:00</td>
<td>IT 102</td>
<td>639</td>
<td></td>
<td>639</td>
</tr>
<tr>
<td>12:20</td>
<td>14:20</td>
<td>IT 104</td>
<td>732</td>
<td></td>
<td>732</td>
</tr>
<tr>
<td>15:40</td>
<td>17:40</td>
<td>IT 110</td>
<td>732</td>
<td></td>
<td>732</td>
</tr>
<tr>
<td>18:00</td>
<td>20:00</td>
<td>IT 112</td>
<td>639</td>
<td></td>
<td>639</td>
</tr>
<tr>
<td>19:15</td>
<td>21:15</td>
<td>IT 132</td>
<td>732</td>
<td></td>
<td>732</td>
</tr>
<tr>
<td>20:15</td>
<td>22:15</td>
<td>IT 140</td>
<td>639</td>
<td></td>
<td>639</td>
</tr>
</tbody>
</table>

**Booking with 1 Time**

<table>
<thead>
<tr>
<th>South Africa Airline</th>
<th>Departure</th>
<th>Arrival</th>
<th>Flight</th>
<th>Total Price (ZAR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>05:45</td>
<td>07:45</td>
<td>SA 378</td>
<td>ZAR 684</td>
<td></td>
</tr>
<tr>
<td>06:00</td>
<td>08:00</td>
<td>SA 302</td>
<td>Sold Out</td>
<td></td>
</tr>
<tr>
<td>06:20</td>
<td>08:20</td>
<td>SA 304</td>
<td>Sold Out</td>
<td></td>
</tr>
<tr>
<td>06:50</td>
<td>08:50</td>
<td>SA 306</td>
<td>ZAR 809</td>
<td></td>
</tr>
<tr>
<td>07:30</td>
<td>09:30</td>
<td>SA 308</td>
<td>ZAR 559</td>
<td></td>
</tr>
<tr>
<td>08:00</td>
<td>10:00</td>
<td>SA 314</td>
<td>ZAR 559</td>
<td></td>
</tr>
<tr>
<td>08:40</td>
<td>10:40</td>
<td>SA 316</td>
<td>ZAR 559</td>
<td></td>
</tr>
<tr>
<td>09:50</td>
<td>11:50</td>
<td>SA 322</td>
<td>ZAR 559</td>
<td></td>
</tr>
<tr>
<td>10:15</td>
<td>12:15</td>
<td>SA 324</td>
<td>ZAR 559</td>
<td></td>
</tr>
<tr>
<td>20:55</td>
<td>22:55</td>
<td>SA 374</td>
<td>ZAR 559</td>
<td></td>
</tr>
</tbody>
</table>

**Booking with South Africa**

<table>
<thead>
<tr>
<th>Kenya Airline</th>
<th>Departure</th>
<th>Arrival</th>
<th>Flight</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:30 am</td>
<td>10:35 am</td>
<td>Flight 805</td>
<td>$89.7</td>
<td></td>
</tr>
</tbody>
</table>

**Booking with Kenya airline**

<table>
<thead>
<tr>
<th>British Airline</th>
<th>Departure</th>
<th>Arrival</th>
<th>Flight</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>06:15</td>
<td>08:15</td>
<td>BA 600 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
<tr>
<td>07:00</td>
<td>09:00</td>
<td>BA 6424 Comair</td>
<td>1470ZAR</td>
<td></td>
</tr>
<tr>
<td>08:40</td>
<td>10:40</td>
<td>BA 6406 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
<tr>
<td>10:00</td>
<td>12:00</td>
<td>BA 6410 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
<tr>
<td>11:15</td>
<td>13:15</td>
<td>BA 6408 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
<tr>
<td>14:00</td>
<td>16:00</td>
<td>BA 6416 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
<tr>
<td>15:45</td>
<td>17:45</td>
<td>BA 6418 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
<tr>
<td>17:00</td>
<td>19:00</td>
<td>BA 6422 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
<tr>
<td>18:00</td>
<td>20:00</td>
<td>BA 6426 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
<tr>
<td>19:45</td>
<td>21:45</td>
<td>BA 6438 Comair</td>
<td>600ZAR</td>
<td></td>
</tr>
</tbody>
</table>

**Booking with British Airway**
The global interface presented in Figure 17 above contains unified data interfaces from the deep web airlines. The pure data from each airline website are stored in separate tables. Each table contains four fields; which are departure time, arrival time, flight number and prices. These fields come in different orders from the Deep Web; however the integrator system re-orders the fields to be in the same format for every airline on the data global interface. The conventions for notation of the values for these four fields are different from one airline to another; for example departure time and flight number notation in Kulula differs from the Mango notation. The survey conducted on forty (40) airlines proved that the convention for notation of the values for these four fields is not standard. Each airline uses the convention of its choice. Among these four fields found in the deep web, there are other fields (e.g. aircraft in SAA, class of traveler in British airline, etc…) that have been excluded from the global interface for the reason that they are not sufficiently common and not of interest to the user.

When the HTML DOM tree parser processes the HTML document, the pure data is extracted and then sent to the system integrator. The integrator displays or prints the data result on the global interface in order of one airline after another airline, as shown in Figure 17. At the beginning, the integrator displays one row with four columns containing departure time, arrival time, flight number and price; at the next iteration, the integrator again inserts new row splits with four columns where the values of departure time, arrival time, flight number and price are printed. This process continues as long as the integrator still has the values that corresponded to these four fields in the table of that particular airline. Once the process of one airline is accomplished, the integrator proceeds with the next airline, until all airlines have been processed. Each value in the result page is recognized from string pattern.

As we can see the result on the global interface in Figure 17, some airlines come with extra column of price field (e.g. Kulula, One Time). This extra column is based on the policy of the airline where the first field of price contains the unit and the second field contains total including the government tax.
6. System Testing and Performance

The implementation of the system in this project concentrated on the Result interpretation in the Deep Web, so this chapter also concentrates on the Result interpretation i.e. HTML data extraction and integration.

Various tests were performed to identify errors and to ensure that the Result interpretation system runs smoothly in HTML parsing, data extraction and then data integration on the global interface. This chapter is broken down into two sections that are: system testing piece by piece and overall system performance.

6.1. System Testing

Our system was tested with fifty-five response HTML pages. These HTML pages were rendered by fifty five airline databases. The names of these airline databases can be found in appendix A at the end of this report. These airline databases were chosen randomly among the various airlines that exist in the Deep Web. The system was tested on three aspects, which are HTML Parsing, Data extraction and Data integration.

6.1.1. Testing HTML Parser

The first component of the Result interpretation system is the HTML parser. This component of the system has been tested with each of fifty five airlines. The result produced in parsing each HTML page rendered by each database, was one hundred percent successful. After the HTML Parser accomplished its job, the HTML document was moved to the Data Extractor component for extracting the data from the page.

6.1.2. Testing Data Extractor

The second component of the Result interpretation is the Data Extractor. The job of the data Extractor is to pull out of an HTML page the data information needed. The robustness of our Data Extractor method depends specifically on where the block of data are stored in an HTML page. Our Data extractor works perfectly if the blocks of data are recorded in the table region or tag of an HTML page. Our data Extractor performs poorly if the block of data is recorded outside of the table region of an HTML page or if the blocks of data are embedded inside of the Java script or if the blocks of data are read straight from the database and then embedded inside any function.

In the test run on our Data Extractor method to extract the data in fifty five HTML airline pages, the Extractor succeeded fully to extract data in forty nine pages; and partially to extract data in two pages, and then completely failed to extract data in four pages. In the case of data fully extracted from forty nine pages, the blocks of data were stored inside a Table region of the HTML page, so our Extractor extracted the data at a hundred percent. In the case of data partially extracted from two pages, some blocks of data were stored inside of the Table region of the HTML page; these blocks of data were successfully extracted by our Extractor. Other blocks of data
stored in the HTML page were retrieved from the database by the Java script function. These kinds of data blocks could not be extracted by our Extractor. The last case is where our extractor completely failed to extract data from four HTML pages of fifty five HTML pages returned; in this case, all blocks of data are not in the table region but are read from databases and embedded in the Java script function. The Extractor component could not process the data for the reason that the data was isolated in the data region.

6.1.3. Testing Data Integrator
The last component to be tested in the Result Interpretation System is the Data integrator component. The task of this component is to integrate different data results from the Deep Web of the airline into a unified interface. These data results are returned by the Data Extractor component. The data extracted from each HTML response page returned by various database sources were sent to the integrator mechanism in order to integrate these data in a unified data interface; the test performed on this Integrator component showed that the data results were successfully integrated (see Chapter 5, Figure 17). The integrator component uses the Do-Get method and Do-Post method provided by NetBeans IDE 6.0.1 application for getting data results from the Data Extractor component and then posts these data results to a data unified interface.

6.2. System Performance
This section discusses the performance of the Result Interpretation System based on the requirements in section 1.2 of the project.

The system was first tested with six HTML response pages returned by six airlines, which are: Kulula, Mango, One Time, South Africa Airways, Kenya Airways and British Airways. The result produced by the system was successful. This result can be seen on the data unified interface in Figure 17 of the Implementation chapter.

A second test was run on the system with the forty nine HTML response pages returned by forty nine airlines chosen randomly from various airlines on the Deep Web. In testing, each of forty nine pages was submitted to the system one after another. The system succeeded completely to interpret the result in forty three pages; interpreted partially in two pages, and then failed to interpret four pages. Even though our system failed to interpret the result in some pages, its performance is still very good when compared with the percentage of success and failure of result interpretation from HTML pages rendered by various airline database sources.

6.3. Findings
This section discusses the results obtained from two tests that were run and presents the results on the figures and tables.

The result performance produced in the first test case run is illustrated in the Figure 18. In this test, the system succeeded to interpret all six HTML pages where used. This Figure shows the relation between HTML page number and time in the
The system spent less than ten seconds for extracting data up to four HTML pages; spent ten seconds for extracting data in five HTML pages and spent one hundred and five seconds for extracting data in six HTML pages. The great variation in amount of time in the last case is due to the reason that the system waited for long time to get the response HTML pages from the Deep Web.

Figure 18: Relation between page number and extraction time in second.

The amount of time for data extraction can continue to increase if the response HTML pages returned by the Deep Web are supplied with big delay to the Result Interpreter System. Table 1 below records information regarding data extraction presented in Figure 18 above.

<table>
<thead>
<tr>
<th>Number of pages</th>
<th>Extraction Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>105</td>
</tr>
</tbody>
</table>

The amount of time in Table above was recorded in each case of run by the NetBeans application that used for developing our system.
Figure 19 below illustrated the correlation between a set of HTML page in KB to be extracted and amount of time in second spent on each set. This Figure shows that the system spent much amount of time in processing a set containing six HTML pages than other sets.

![Correlation between sizes of HTML pages and time extraction](image)

**Figure 19: Relation between result sizes in KB and Extraction time in second.**

Table 2 below records the information of set of HTML page in Kilobyte and amount of time in second that system spent on each set. The set that has size of 279.9 KB took time of 105 seconds to be processed by the system.

**Table 2: Relation between result sizes and Time**

<table>
<thead>
<tr>
<th>Set of HTML page Sizes (KB)</th>
<th>Extraction Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24.4</td>
<td>5</td>
</tr>
<tr>
<td>54.9</td>
<td>5</td>
</tr>
<tr>
<td>130.2</td>
<td>4</td>
</tr>
<tr>
<td>151.2</td>
<td>6</td>
</tr>
<tr>
<td>244.2</td>
<td>10</td>
</tr>
<tr>
<td>279.9</td>
<td>105</td>
</tr>
</tbody>
</table>

The amount of time was recorded by the NetBeans application in each case of run.

Table 3 below records the information on the total number of pages tested for the system overall performance. From the table we can see that the total number of pages tested was fifty five. From this number, 49 pages were successfully extracted; 2
pages were partially extracted and 4 pages failed to be extracted for the reason that the data were embedded in the script function.

Table 3: Information on total number of HTML pages tested for extraction

<table>
<thead>
<tr>
<th>Number of pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total pages tested</td>
</tr>
<tr>
<td>Pages Successfully Extracted</td>
</tr>
<tr>
<td>Pages Partially Extracted</td>
</tr>
<tr>
<td>Pages Failed to be Extracted</td>
</tr>
</tbody>
</table>

The pie chart in the Figure 20 below illustrates clearly data extraction and the system’s overall performance.

Figure 20: Pie chart of Data Extraction

From this pie chart, we can see that 89% of HTML pages were perfectly processed, 4% partially processed and 7% were not processed because of the difficulty of accessing the data region.

The results produced on the tests run were successfully interpreted in the case where the blocks of data in the HTML response pages are stored in the table region. Therefore, our system can be used in the Deep Web airlines to extract the block data in the table tag of HTML response pages returned. However, our system does not handle blocks of data that are recorded outside of table region in the HTML response pages.

The amount of time that the system spent to interpret the result is depend on how fast the HTML pages are returned by the Deep Web and the number of pages to be extracted. If the system interpreter waits for long time to get the HTML pages input from the Deep Web, the amount of time in the data extraction will be high.
7. Conclusion
This final chapter of the project summarizes our achievement, then reiterates some of the issues encountered throughout the development process and assesses the project.

7.1. Report Summary

The incredibly speed development of World Wide Web has brought many changes in the management of information. Users of Web become more concerned how easily to access and manage the information on the World Wide Web. To help consumers and providers to manage this information, it is useful to use databases to generate Web pages dynamically. Often, dynamically generated pages are accessible only through an HTML form that invokes a common gateway interaction request to a Web server. The Web is composed of the Surface Web and the Deep Web.

Surface Web refers to the Web pages that are static and linked to other pages, while Deep Web refers to the Web pages created dynamically as the result of search of specific search. This project concentrated on the Deep Web specifically on airlines domain. The Deep Web tool is divided into two parts, which are: Query formulation and Result interpretation. Query formulation was developed by Martha Kamkuekah. This project report focused on the Result Interpretation part.

The Result Interpretation extracts the query results achieved from response HTML pages returned by various Web database sources, and merges the results together under a unified interface. There are three components in this part that are: HTML Parser, Data Extractor and Data Integrator.

**HTML Parser:** parses a response HTML page returned by the Deep Web databases.

**Data Extractor:** received the parsed HTML pages from HTML parser, identifies and extracts the data. This data contains the noise; our Data Extractor removes the noise on the basis of string matching and string pattern.

Our implemented Extractor works perfectly if the blocks data to be extracted are located in the table region of HTML page. And our Extractor performs poorly if the blocks of data are embedded in a script function. The tests run with fifty five response HTML pages on our data Extractor produced 89% of pages were completely extracted, 4% of pages were partially extracted and 7% of pages failed to be extracted because of difficulty to access data region. Even if our Data Extractor failed to extract data in some pages, it’s still performing well if compares to the number of pages that were successfully extracted.

**Data Integrator:** gets the data extracted from Data Extractor and then merges these data together under a global schema (unified interface). The data received from the Deep Web airline, comes in any order. The Data Integrator system re-orders and displays them on the global interface.
The primary goal of this project is to interpret the results of response HTML pages returned by various Web databases. This goal is achieved and it was successfully to interpret the result from the Deep Web airlines.

Although the project is successfully, an HTML page continues to evolve and new features may add to this language. These features might bring the new way the data should be stored in an HTML. This situation might cause to review our Result Interpretation System. If in the case where all the blocks of data are embedded in the script functions of all response HTML pages, our system needs to be re-written.

In the development of Result Interpretation System, we faced two problems. The first problem affected our project time and the second problem affected the design of our system.

The first problem encountered during the development of the system was the use of the UCT proxy network that could not allow our Java code to access external websites. The connection to the UCT network each time timed out when a query to external site is sent via the Java code. This problem was solved by the use of a Vodacom modem that allowed us to access the needed external websites.

The second problem encountered during the development of the project was the computer that has been allocated to us. The machine used was very slow and was unable to handle the workload. We re-installed the operating system to improve the performance. This re-installation did not solve the problem completely. Due to the memory constraint of the machine, we did not implement a local database for storing the large amount of results extracted from the Deep Web. The results extracted were displayed as an HTML page for each query sent.

7.2. Project Assessment
This section discusses the project assessment with reference to the project proposal and project plan.

The development of our system has conformed to our project proposal presented because the Result Interpretation System that we have developed, takes the result pages returned by the Query Formulation System then extracts the data and merges the results together under a unified interface. The only constraint in data extraction is that our system only works if the block of data is localized in the within the HTML page. If the block of data is stored within a java script, like some websites do, our system is unable to cope. Due to the time constraint, we were not able to solve this problem.

The project development did not exactly follow the proposed project plan. We were late for two weeks due to the proxy connection problem on the UCT network.
During the development of the project, we learnt so much about how to develop a system capable of parsing an HTML page, then extract the information and merge this information on a global schema.
8. Future Work

The Result Interpretation System implemented, extracts the query results achieved from the Deep Web airline, and integrated the results together under global interface. The system successfully interprets the results, if these results are in the Table region of HTML pages. In the case where the results to be interpreted are embedded inside of the script function of HTML page, the system won’t be able to interpret the results. In order to resolve this problem an extra work must be done to build a new system called a “Generic Interpreter” that can interpret the results from the Table region of an HTML page and the script function of an HTML.

Testing
The system should be tested with at least two hundred HTML pages to ensure its good performance and to see how the amount of time varies against number of page used to extract data. But due to time constraint, it was tested with fifty fives HTML pages in total.
References


[References]


[22] Travel Start Website. *Result page*. Available at: [http://www.travelstart.co.za/search.jsp?homeLeaveMonth=200812&paxInfants=0&paxTeens=0&extPartner=&paxKids=0&search.y=13&isReturn=false&isLocSearch=true&numberOfRooms=1&search.x=72&homeLeaveDay=30&view=air&destCity=CPT&](http://www.travelstart.co.za/search.jsp?homeLeaveMonth=200812&paxInfants=0&paxTeens=0&extPartner=&paxKids=0&search.y=13&isReturn=false&isLocSearch=true&numberOfRooms=1&search.x=72&homeLeaveDay=30&view=air&destCity=CPT&).
# Appendix A: Airline Names

## Table 4: Deep web Name used for system testing

<table>
<thead>
<tr>
<th>Airline Name</th>
<th>Data fully Extracted</th>
<th>Data partially Extracted</th>
<th>Data Failed to be Extracted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kulula</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mango</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>One Time</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA Airways</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>British Airways</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kenya Airways</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qatar Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Emirates Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seair Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Namibia Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kenya airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lufthansa Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vueling.com Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gulf Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zest-Airs</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spain Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singapore Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portugal Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brussels Airlines</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Air Malta</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Air France</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virgin Atlantic Air</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ariana Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kuwait Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Royal Jordanian Air</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saudi Arabia Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pakistan Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>India airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mexico Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Malaysia Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Australia Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thomsonfly.com</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canada Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Senegal Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Air Zambia</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Air Algeria</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expedia Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>US Airways</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maps World Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ghana Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aer Arann Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Italia Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lot Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alexandria</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qantas Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vaussiaia Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Taca Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Croatia Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WIZZAIR.COM Air</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Surinam Airways</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Zealand</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBA Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iceland Express Air</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jordanian Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Egypt Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turkish Airline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>