5-7-2005

Autonomous Consolidation of Heterogeneous Record-Structured HTML Data in Chameleon

Philippe Chouvarine

AUTONOMOUS CONSOLIDATION OF HETEROGENEOUS RECORD-STRUCTURED HTML DATA IN CHAMELEON

By

Philippe Chouvarine

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

December 2004

AUTONOMOUS CONSOLIDATION OF HETEROGENEOUS

RECORD-STRUCTURED HTML DATA

IN CHAMELEON

By

Philippe Chouvarine

Approved:

Julia E. Hodges Professor of Computer Science and Engineering (Major Professor)	Hasan M. Jamil Associate Professor of Computer Science at Wayne State University (Thesis Director)
Susan M. Bridges Professor of Computer Science and Engineering (Committee Member)	Edward B. Allen Assistant Professor of Computer Science and Engineering (Graduate Coordinator)

Robert P. Taylor
Interim Dean of
the College of Engineering

Name: Philippe Chouvarine

Date of Degree: December 11, 2004

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Julia E. Hodges

Director of Thesis: Dr. Hasan M. Jamil

Title of Study: AUTONOMOUS CONSOLIDATION OF HETEROGENEOUS RECORD-STRUCTURED HTML DATA IN CHAMELEON

Pages in Study: 60

Candidate for Degree of Master of Science

While progress has been made in querying digital information contained in XML and HTML documents, success in retrieving information from the so called “hidden Web” (data behind Web forms) has been modest. There has been a nascent trend of developing autonomous tools for extracting information from the hidden Web. Automatic tools for ontology generation, wrapper generation, Web-form querying, response gathering, etc., have been reported in recent research. This thesis presents a system called Chameleon for automatic querying of and response gathering from the hidden Web. The approach to response gathering is based on automatic table structure identification, since most information repositories of the hidden Web are structured databases, and so the information returned in response to a query will have regularities. Information extraction from the identified record structures is performed based on domain knowledge corresponding to the

domain specified in a query. So called ”domain plug-ins” are used to make the dynamically generated wrappers domain-specific, rather than conventionally used document-specific.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Problem Specification

The Web can be viewed as an ever-growing large database having millions of users all over the world. Yet the Web has very little database functionality. An average user begins his/her search for information on the Web from the sites hosting the Internet search engines, such as Google or Alta Vista. They provide important automation for finding potentially relevant Web pages based on the user's keyword search – a process known as Information Retrieval (IR). However, after IR is applied, the user still needs to spend a significant amount of time and effort searching through relevant and irrelevant links. The biggest limitation of IR is that it cannot be applied to the hidden Web, the databases behind the firewall. Only pages with input forms for querying online databases can be searched, but not the dynamically generated result pages.

One specific type of Web query that is of practical interest to the users is querying vendors or organizations providing the same type of products or services. The users are interested in comparing prices on the same offers from different vendors, or comparing job offers satisfying the same criteria from different Web sites, etc. The overwhelming majority of such sites use online databases to efficiently present to the user their informa-

tional inventory showing up-to-the-minute changes. This introduces yet another task for the user: online forms must be filled out on each such site to query the online databases. Such a process, hereafter called the comparison search, adds another dimension to the already difficult problem of finding relevant information on the Web. The steps involved in a typical comparison search can be summarized as follows:

1. Retrieve links to potentially relevant sites by submitting a keyword query to a search engine (e.g., Yahoo and Alta Vista).
2. Follow the relevant links, which usually load a page with a form for querying an online database of a particular vendor.
3. Input the search parameters, submit the form, and wait for the query results to be fetched from the online database and encoded into the HTML format.
4. Write down the results in a table for future comparison.
5. Repeat steps 2 - 4 for each relevant link.

If the information retrieval phase generates a significant number of relevant links, this tedious process may take hours of the user's time. The need for efficient automation of this process has not been satisfied yet.

Practical solutions for the comparison search problem are limited to third-party commercial sites (e.g., Yahoo Shopping and MySimon). They usually provide a list of products of the same domain (mostly electronics or books) from various online vendors, which can be sorted by price, and allow the users to initiate a transaction on a remote vendor site with a click of a button. Two approaches are used to create such lists: real-time data collection (e.g., Jungo and MySimon) and storing the data locally using periodical updates (e.g., Junglee) [8]. Though these sites make life easier for the users who know about them, the

provided selection of online vendors is limited by the library of wrappers that these sites maintain. More importantly, these sites provide comparison shopping in only one or two domains.

1.2 The Key Idea Through a Motivating Example

When we think about the Internet, we think about the large repository of information where anything can be found if one puts enough effort and tenacity into the search. The level of effort required may also deter us from effectively using it. However, in the not so distant future the situation may change and the Web will become more and more database-like. The users may be able to pose complex queries that presently can only be performed by professional database systems. For example, a user living in California may decide to visit his/her family in Tennessee. The user may then pose a single query that would inquire about the cheapest airfare and car-rental options in Memphis with the condition that his/her stay in Tennessee would not overlap with the tornado season.

No resource on the Web can currently process this type of query that spans a number of domains. It would be unreasonable to expect third-party commercial Web sites to originate comparison search services that would span over such seemingly unrelated domains as weather and car rentals. Therefore, as we see it, the future is in client-side applications that can use domain knowledge plug-ins for querying a set of sites with online databases satisfying certain criteria in a given domain. Naturally such Web agent applications will support queries over any combination of domains that are supported. The Chameleon

agent that we developed follows this smart agent paradigm. The agent has an architecture that allows functionality extension by simply adding domain plug-ins (consisting of one DLL and two or more XML files) in specified directories. The appropriate DLL is linked based on the domain name specified in the query. The user also has flexibility in customizing extraction rules, editing domain-specific thesauri used in mapping of Web forms, and adding new search keywords.

The Information Extraction (IE) part of the project that we have implemented is fully compatible for use with a smart agent and nicely customizable by the user. The two algorithms that we developed are based on the assumption that records generated by remote scripts are based on HTML table structures (virtually all script-generated pages that we examined follow this format). This approach allows significantly improved processing time since we manipulate the table tree rather than the Document Object Model (DOM) tree that includes every tag of the page. For complex domains such as flight reservation, the result pages may contain a significant number of tags; for example, the page generated by www.orbitz.com contains roughly 30,000 tags.

The algorithms analyze the HTML structure of the result pages by finding structurally identical branches of the table tree in the case of the first algorithm or finding a table that most likely contains encoded records in the case of the second algorithm. Further structural analysis and domain knowledge-based validation precisely identify the encoded records. Information atoms of these records are then mapped to attributes of the target schema using analysis of the HTML structure and domain-knowledge-based extraction rules. This

approach allows data extraction at run time independent of structural or schema formats, which means that the system is not bound to static extraction rules previously identified for a particular site. On the contrary, it can potentially extract data from any site in a given domain. The target schema is also adaptable to the processed source schemas. It contains the attributes having the weight associated with the frequency of their occurrences and passing the threshold criterion set by the user. If the threshold value is set at 0% a target schema having the union of attributes from all source schemas is produced. Setting the threshold at 100% would produce the target schema with the intersection of the attributes.

1.3 Approach to Solving the Problem

Our approach to addressing the issue of automated comparison search in the hidden Web involves looking at a query from the domain of application. In other words, we contextualize the query by domains – for example, travel, car rental, job search, etc. Contexualization helps by adding relevant knowledge in the form of domain plug-ins. Typically the plug-ings contain domain knowledge in the form of key words for IR, thesaurus, mapping heuristics, information extraction rules, etc., relevant to a domain. A single logical Domain Plug-in Module (DPM) is physically implemented as XML and DLL files. In other words, DPMs contain the heuristics that are relevant for comparison searches of different domains. In this way, the query engine itself remains domain independent, and its functionality changes with plug-ins for a given set of data repositories.

Our system, Chameleon, also addresses a general limitation of the IR process. This process yields a list of the most successfully promoted links satisfying the keyword search criteria, whereas the user may desire to include in the comparison search his/her favorite site in the given domain and query the other sites having ontologies most closely resembling the ontology of this site. The system provides an option to specify the URL of such an example site and then creates the ontology of this site used for the query mapping. Processing of the next site uses this example ontology for mapping the query to the corresponding HTML form. This gives a better chance of successfully mapping sites with similar ontologies. Once the successful mapping is performed, the example ontology is combined with the ontology of the just processed site. As this process continues, the example ontology is refined to become common to all the processed sites. Once the system has a mapping of query parameters to Web form controls, it automatically fills out and submits the forms, and then automatically extracts the query result data from the pages generated by online databases on these sites.

Chameleon's functionality largely depends on its ability to consolidate heterogeneous records obtained from multiple sources so that a correct and coherent comparison can be performed by the front-end query engine. That leads to the main focus of this paper. In this paper we propose two algorithms for autonomous record structure identification and consolidation of records from multiple sources, and an architecture for a query engine for comparison search. We emphasize here that our approach relies on ad hoc integration of

Web sites. The sites to be integrated are a function of queries and query variables, and the integration happens at run time.

The two record identification algorithms discussed in this paper use different approaches which maximize the subset of sites covered by the system. Attribute values of the identified records are extracted using domain-knowledge-based IE techniques. Our implementation of the target schema, which combines attributes of the source schemas of the result pages from different sites, is adaptive to attribute frequency of the source schemas and can be customized to have attributes ranging from intersection to union of the source attributes.

1.4 Organization

The remaining chapters are organized as follows: Chapter II provides an overview of tools for automated filling and submitting of Web forms and information extraction tools. Chapter III contains an overview of the proposed tool (record identification algorithms, information extraction, and system implementation). Chapter IV covers information domains of Web sites used for testing, qualitative comparison analysis with related tools, and experimental results showing superiority of Chameleon compared with the closest ontology-driven IE tool. Chapter V provides a conclusion and an outline of the future research directions.

CHAPTER II

LITERATURE REVIEW

The project has two distinct phases: automatic filling out /submitting of forms to online databases and record identification / information extraction from the resulting HTML pages. Therefore the literature review covers the two areas of related research separately.

2.1 Automated Filling and Submitting of Web Forms

Research in this area has been driven by various end goals, the three most common of which are 1) to assist the user browsing the Internet in filling out redundant forms (e.g., Microsoft .NET Passport), 2) to design an automatic form-filling system that can be used in conjunction with a Web crawling agent to automatically pose queries to online databases (e.g., ShopBot, HiWE and OntoBuilder), and 3) to automatically transfer the entire content of an online database to a local database by interaction with the corresponding Web form (e.g., BYU).

One of the earliest research projects in this area is ShopBot [7]. This system tries to identify the correct forms for query submission by following all links found on the main page of a given site, the URL of which is provided as an input. If the system is confident that the forms do not have the correct fields, these forms are discarded. The remaining

forms are given several test queries and the corresponding result pages are analyzed to determine the correct form. This system works only with the shopping-domain forms having simple one-line search forms.

A more recent project whose goal is closely related to our research is the Hidden Web Exposer (HiWE) [21]. The project is aimed toward adopting a task-specific human-assisted approach to Web crawling; or, in other words, the user can customize the crawler to access the specific part of the hidden Web of interest to the user. The system includes Form Analyzer and Form Processor modules tackling the task of automatic filling out Web forms. The system uses a special Label Value Set (LVS) table for passing query values to forms. The LVS table uses fuzzy/graded weights associated with values showing their probability of being correctly used with a given label. Each Web form encountered by the HiWE crawler is processed by matching the labels on the form with the labels in LVS; based on the value weights, they are then assigned to the corresponding fields. The top-weighted value assignments are submitted separately and the result pages are validated to select the correct submission.

Another approach to automated form filling is proposed in [14] by Liddle et al. Their method begins with creating a parse tree of the target HTML form. Then a CGI GET request is sent using empty or default values of form attributes. This submission is followed by other submissions with varied values of certain attributes with the goal to extract all data behind this particular form. Statistical methods are used to minimize the number of samples (submissions). Obviously, this approach can be feasible only for relatively small

online databases where variation of a few default parameters can produce resulting pages covering the whole database. More importantly, this system, as well as Microsoft .NET Passport [17], which is used to assist the user in filling out redundant personal information forms widely used in e-commerce sites, have design goals inconsistent with their use as a part of Web-crawling agents.

One of the latest research projects in this area is the OntoBuilder project [18]. We use the product of this research as a part of our system. Manual ontology generation is a time consuming and tedious process; OntoBuilder automates this process. Creation of ontologies is an effective approach to mapping queries posed by the user to Web forms. OntoBuilder was designed to work specifically with HTML form ontologies. It has the ability to create an initial ontology and then merge this ontology with ontologies of the other sites in the same domain. The resulting ontology is common to the HTML forms of all the sites that were used in this process. The methodology consists of two phases: the training phase and the adaptation phase. In the training phase, the initial ontology is built, which is essentially a mapping of the user query to the HTML form. In the adaptation phase, ontologies of HTML forms from other sites are extracted and merged with the current iteration of the common ontology one at a time. This process requires minimal user supervision. OntoBuilder has high precision of ontology generation (more than 90% in narrow domains). There has been some research on creation of a fully automatic OntoBuilder agent that takes URLs with HTML forms as input and outputs their common ontology in an XML file. We use similar predefined XML files to simulate the use of

the OntoBuilder agent in our system. While mapping a user query to Web forms using OntoBuilder agent may not be 100% successful, it is a very powerful tool for finding and validating relevant Web forms. The discussion on how the OntoBuilder agent is used to perform these two tasks is given in section 3.2.3.

2.2 Record Identification and Information Extraction

This section provides an overview of IE approaches and applicability of tools designed using these approaches to the agent paradigm. The interested reader may find more information on IE in [8], [13], and [19].

Following the taxonomy of Web data extraction tools proposed by Laender et al. in [13], we will cover classes of such tools and identify the areas of their applicability. The taxonomy includes six groups:

1. Languages for wrapper development
2. HTML-aware tools
3. NLP-based tools
4. Wrapper induction tools
5. Modeling-based tools
6. Ontology-based tools

The taxonomy only identifies the approaches that can be taken in development of Web IE tools. In practice, a combination of such approaches can be used; e.g., the Chameleon tool that we developed relies on HTML-aware and ontology-based approaches.

2.2.1 Languages for Wrapper Development

One of the first approaches to wrapping Web-based data was the creation of languages for wrapper development. Such languages provide a higher level abstraction aimed to simplify wrapper creation compared with conventional programming languages such as Java or Perl that have traditionally been used for wrapper development.

Minerva [5] is a wrapper development language used in the Araneous system [16]. Minerva uses an EBNF-style grammar to define a set of productions for each target document. The irregularities of the processed Web pages are handled by an explicit procedural mechanism inside the grammar parser.

One of the latest efforts in this area is HTQL [4]. It is designed to perform SQL-like queries over semistructured HTML pages. Its operation is based on parsing a target HTML page into a DOM tree and locating parts of the tree specified in the query. One query can specify a number of repeated structures and filter based on the extracted values. This language has been used in conjunction with the PickUp system [3] that automatically generates HTQL queries of visually selected areas in the Web Browser window. These queries are in essence document-specific wrappers for Web IE.

Languages for wrapper development can potentially be used in conjunction with IE modules of smart agents to provide a higher abstraction for the IE task.

2.2.2 HTML-aware Tools

These tools rely on the HTML structure of the document by forming a parse tree consisting of HTML tags and finding regularities in such trees to identify objects for extraction. A good representative example of this type of tool is XWRAP by Liu et al. [15]. It uses a semi-automatic wrapper generation procedure where a user can try using one of the six predefined extraction heuristics to locate the target extraction objects (tuples). The user can vary the number of attributes per object to achieve the desired result. Once the result is satisfactory, the user can name each attribute and generate a document-specific static wrapper. (By static wrappers we mean that their extraction rules are coded at compile time, as opposed to dynamic wrappers that infer extraction rules at run time). This tool allows fast generation of static site-specific wrappers under user supervision. The tool that we want to integrate with a smart Web query agent should be fully automatic in its IE functionality and preferably generate extraction rules in run-time mode and cover a certain set of result pages generated by various online databases (including pages that will be designed in the future) in a given domain. XWRAP does not have these qualities and cannot be used for this purpose.

One of the most advanced HTML-aware tools is RoadRunner [6]. The essence of this tool is comparison of two or more pages generated by the same online database that allows generating of a schema common to the pages. After that the system infers a grammar that populates a table of this schema with the corresponding extracted data. Unlike XWRAP, RoadRunner is a fully automatic system that is capable of capturing complex record struc-

tures. However the system generates document-specific static wrappers. Though RoadRunner is potentially applicable for use with a smart Web query agent, it would dictate the need to maintain a library of static wrappers that should be verified every time a wrapper from this library is used (the change of layout or formatting can easily make the existing static wrappers useless) and generate a new static wrapper for a changed page or new page which does not have a corresponding wrapper in the library yet. This would add substantial time and resource overhead. An alternative solution would require generation of temporary wrappers for every target HTML page, which, given the methodology used by RoadRunner, would cause even greater time overhead.

2.2.3 NLP-based Tools

Natural language processing (NLP) techniques have been developed and successfully used on free text much earlier than there was a need for Web IE. NLP techniques work well with unstructured, grammatically rich free text by relying on part-of-speech tagging, lexical semantic tagging, finding relationships between phrases or parts of sentences, etc. Therefore, NLP techniques are only useful for Web IE from pages containing big chunks of relatively unstructured text, which is not the case with most hidden Web pages generated by online databases where the schema attribute values are populated with text usually having very little grammatical structure.

One of the representative tools of this class is RAPIER (Robust Automated Production of Information Extraction Rules) [2]. This system takes as input texts and filled tem-

plates indicating the information to be extracted. The learning algorithm creates pattern-matching rules to fill the slots in the template. The algorithm learns unbounded patterns that include constraints on the words and part-of-speech tags surrounding the data used to fill the slots. The extraction patterns are based on delimiter and content description. The IE rules consist of three parts: pre- and post- fillers (patterns that match the text preceding and following the target text) and a filler (the target text). The system extracts a single record from each input document, therefore only single-slot extraction is possible unless the text is divided into more than three fields [22].

One of the most advanced NLP tools capable of multislot extraction is WHISK [22]. The system includes extraction rules from a set of hand-tagged training example documents. The process begins with an empty set of extraction rules and a set of untagged instances (sentences). The user tags the case frames to be extracted from the presented instances followed by the system's learning phase. During the learning phase WHISK uses the tagged case frames to create rules and to test the accuracy of the proposed rules. The tagging and learning processes interleave and repeat iteratively. The algorithm learns the extraction rules by top-down induction beginning from the most general rule and gradually adding terms to reduce errors to zero. The extraction rules identify the context and delimiters of the target text. WHISK can handle variable permutations of slots if training examples of all possible orderings of items were used in the learning phase.

2.2.4 Wrapper Induction Tools

These tools are influenced by natural language processing techniques, but instead of identifying parts of speech using linguistic constraints, they rely on formatting features that allow induction of extraction rules from a training set of example HTML pages. The extraction rules use HTML tags or groups of tags as delimiters.

The first widely known tool of this class is WIEN [11]. Though this tool pioneered the idea of automatic wrapper induction, WIEN has some serious limitations. The user must have knowledge of the structural class of the page and then submit labeled training examples of pages that belong to the same structural class. The system then generates a wrapper that is common to all the training examples and potentially works with the other pages of the same domain and structural class. This capability of generation of domain-specific wrappers rather than document-specific ones is the major advantage of the system. However WIEN does not work with complex nested record structures. This together with the need to know the structural class of the target HTML page are drawbacks of this system.

One of the most advanced tools of this class is STALKER [20]. Like WIEN, the system takes a set of training examples with labeled relevant data and generates a sequence of tokens surrounding the target data. After all positive examples are learned the system outputs a wrapper that covers all of the learning examples. The main advantage of STALKER is that it works with nested hierarchical structures. However, this is achieved by supplying an Embedded Catalog Tree (ECT) that provides structural description of the

target page. This system is not applicable for use in a smart agent because the generated static wrappers rely on ECT, which makes them document-specific. Such wrappers require verification that the structure of the document has not been changed before their use. The methodology used in STALKER wrapper induction precludes attempts to make the wrappers dynamic by using temporary wrappers reinducted for every target HTML page. The primary reason for that is that STALKER requires human supervision.

2.2.5 Modeling-based Tools

Modeling-based tools rely on matching the target structure of data to be extracted with portions of input Web pages. The target structure is provided from a set of modeling primitives (tuples, lists, etc.) that conform to the model of the target data. Usually such tools require substantial user interaction and create relatively static (poorly adaptive) wrappers, so it would not be feasible to use this class of wrapper generation tools with smart agents.

NoDoSE (Northwestern Document Structure Extractor) [1] is an interactive tool that allows one to semi-automatically determine the structure of documents and then extract their data. Using a GUI, the user hierarchically decomposes the document, marking the regions of interest and then describing their semantics. Once the user has identified the interesting regions, the system is able to identify similar regions automatically. This task is performed by a mining component that attempts to infer the grammar of the document based on the information supplied by the user. The mining component supports free text and HTML documents.

DEByE (Data Extraction By Example) [12] is an interactive tool for extracting semistructured data from Web pages. The extraction process is based on the examples supplied by the user via a GUI that allows the user to specify nested tables with possible structure variations. The examples are then used to generate extraction patterns by identifying the example nested tables on the target pages. The DEByE extractor module uses the patterns to execute a bottom-up extraction algorithm that identifies information atoms on the target page and assembles complex objects using the structure of the extraction patterns.

2.2.6 Ontology-based Tools

Ontology-based tools rely on data semantics rather than on the HTML encoding for IE. This allows generation of domain-specific wrappers, which use ontology specification as the means for mapping objects for extraction.

As yet, there is only one widely known ontology-based IE tool; it was developed at Brigham Young University (BYU) [9]. The tool requires a carefully manually constructed ontology that covers a certain domain of sites. Having such an ontology, the tool becomes a dynamic wrapper that works in two distinct phases. First, the record boundaries are identified using a number of heuristics applied in probabilistic (weighted) manner and then the ontology is applied to extract target objects. This approach fits well in the framework of a smart Web query agent. Our IE implementation was influenced by this tool. However, the record boundary identification techniques used in the existing BYU tool rely on the condition of multiplicity of records (the number of records must exceed the number of

tags used to format attributes in the source schema). The real-life documents may not necessarily satisfy this criterion. For example, complex record structures such as those used in the flight-reservation domain are likely to have more tags used to format attributes of a single record than the number of records generated in response to a query.

Our system overcomes this limitation. Another potential drawback of the BYU tool is caused by its neglect of HTML tags in identifying of record attributes. The BYU tool uses an NLP approach to map the records stripped of HTML encoding to the target schema. Such blending of information atoms can potentially cause confusion of data previously grouped in a single atom with the data contained in an ontologically close information atom. Our system avoids this problem by mapping each information atom individually to the target attributes. Yet another shortcoming of the BYU tool is the use of a hard-coded target schema tied to the ontology specification. In the situation where a smart agent extracts information from sites that persistently lack a target schema attribute or have an extra attribute not included in the target schema, the IE performance is degraded. Our system avoids this pitfall by having an adaptive target schema that uses statistical analysis of attribute frequency in the extracted source schemas.

Having reviewed all six classes of Web IE tools, we can conclude that each represents tools having different design purposes from tools of other classes. Requirements analysis for design of a particular Web IE tool may dictate inclusion of features of a particular class of tools. E.g., if the domain of target pages is known to be poorly structured and has grammatically rich texts, the tool to be designed must include NLP features. The

other important issue to point out is that there is a relationship between the ease (degree of automation) of wrapper creation and resilience (adaptiveness) of the created wrappers. For example, there are a lot of HTML-aware tools that can create a static, document-specific wrapper in a matter of seconds. On the other hand, a well defined ontology-based wrapper may take hours of manual design, but the resulting wrapper is dynamically adaptive to any target page in a given domain. This relationship must be considered during the design process when features of certain classes of Web IE tools are selected.

The Chameleon tool developed in this thesis research uses HTML-aware and ontology-based approaches to IE. This combination allows successful extraction of information from HTML-encoded tuples having complex structure (e.g., most tuples in a flight reservation domain use irregularly nested HTML table structures having multiple ontologically close information atoms). The implementation details are given in Chapter III.

CHAPTER III

THE CHAMELEON SYSTEM

3.1 Hypothesis

The purpose of the thesis is to demonstrate the feasibility of a proposed implementation of a query agent for information domains of the hidden Web. We propose a system architecture for the Chameleon agent that allows completely autonomous querying of a selected domain of the hidden Web consisting of a number of Web sites identified by the agent at run time. We assume that the existing research in the field of finding and validating Web forms and mapping queries to these Web forms is sufficient for its successful integration with the proposed system architecture of the Chameleon agent, therefore our research concentrates on the rest of the implementation. The research coveres record identification and information extraction from semistructured HTML encoded records generated by online databases, and consolidation of the extracted data in a recordset that can be used for further conventional SQL-like queries. The proposed tool is able to perform these operations without any prior knowledge of format, structure, or layout of the result pages generated by online databases, as long as all of the result pages belong to the same information domain. The functionality of the Chameleon agent relies heavily on domain knowledge provided by domain plug-ins in conjunction with the proposed record identification algorithms and

information extraction technique that utilize information grouping of the HTML encoding and ontology representation of a given information domain.

3.2 System Implementation Covered in this Research

This thesis research concentrates on record identification theory, information extraction, and system architecture of the Chameleon system. It covers two novel record identification algorithms for semistructured HTML sources generated by online databases, a domain-knowledge driven information extraction technique, and details of system implementation. The implemented system covers the functionality subset related to form filling of Web forms given existing map to the query parameters, further information gathering from result pages and consolidation of this information in a single recordset for further filtering, sorting, etc.

3.2.1 System Architecture

As shown in figure 3.1, Chameleon's main input is a user query. The first line of the query is common for all domains; it contains the domain name. The Query Parser identifies the domain name and uses it to link to the corresponding domain plug-in, which returns expected domain-specific query parameters to the parser. Once the domain plug-in is linked to the system, it passes search keywords and validation criteria to the Information Retrieval Module, which acquires links to relevant Web forms and passes them to the OntoBuilder agent. If the user specified the example URL, the selection of relevant links is influenced

by this URL. The OntoBuilder agent analyzes the Web forms and creates a collection of temporary XML files containing their ontology and mapping to the query parameters. When it is done, each form is processed by the Form Filling Module, Web Browser, and Record Identifier. The Processing Scheduler is used to synchronize form loading, form filling and submission, and extraction of records from the result pages. When all online databases are processed, a string array of identified records from all sites is passed to the IE Module, which extracts individual attribute values from these records and adds them to the Record Set. The Record Set can be sorted or filtered as specified in the user query. Finally, the report having a dynamic schema is generated. The schema includes or excludes source attributes based on their frequency and the threshold value set by the user.

3.2.2 Query Language

The current iteration of our system has the following query language format

```
1.<query> -> select from <domain name>
[:<example URL>]
2.      where <condition> {(and|or)<condition>}
3.      [order by <attribute name>]
4.      [filter:<condition>]
<condition> -> <attribute name>(=|<|>|<=|>=)
               <value>
```

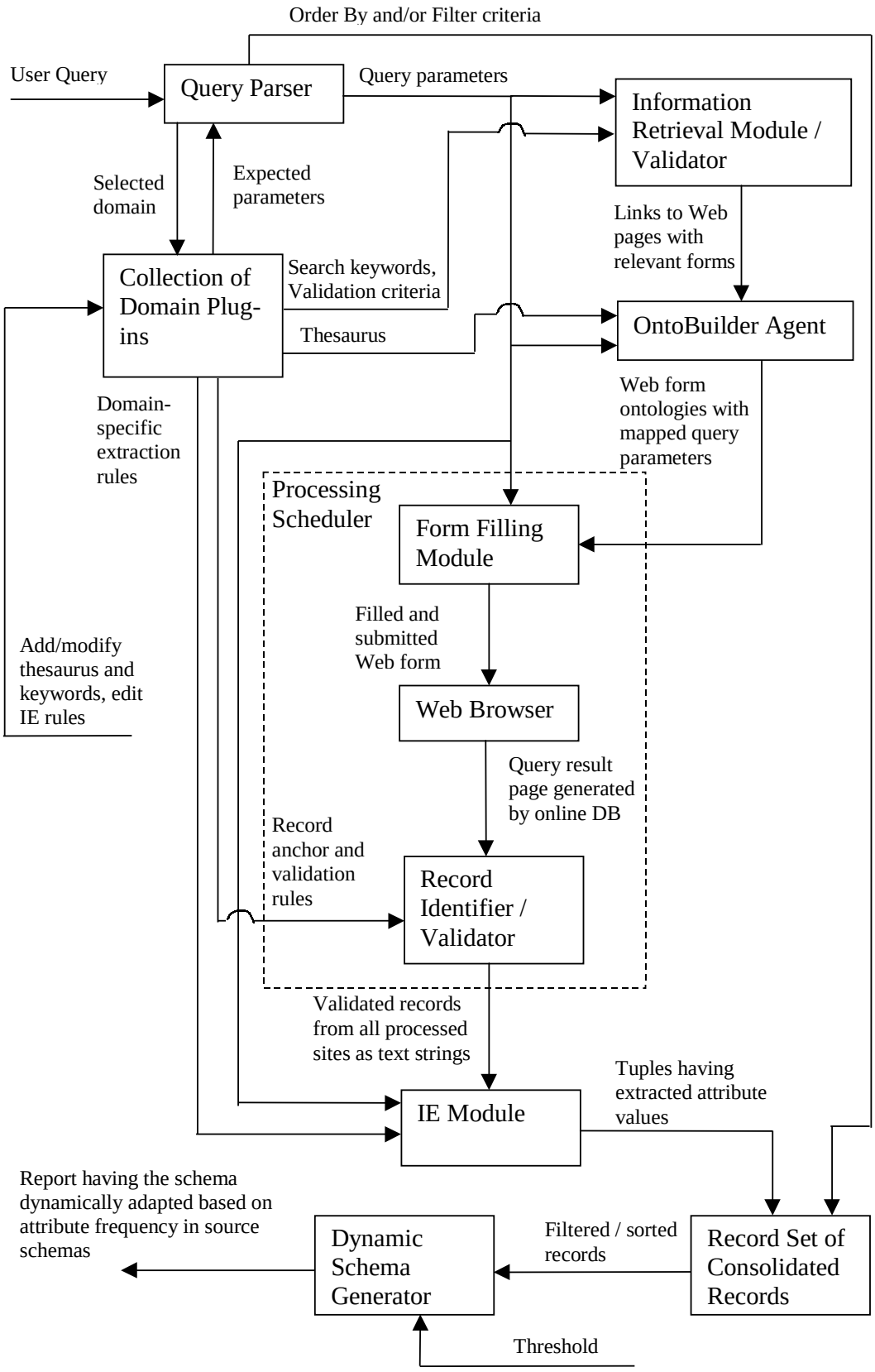


Figure 3.1 The System Architecture

Lines 1 and 2 of the query are used to define the recordset containing records of various online databases in a given domain. Lines 3 and 4 apply sorting and filtering conditions on the recordset. The current specification only works in a single domain. The future work on the system may extend its functionality to allow queries over multiple domains by creating recordsets for every domain and applying conventional SQL queries to the recordsets, much like SQL queries over multiple tables.

Each domain has a predefined set of query attributes that can be used in queries to form, sort and filter the recordset. A sample query for the flight reservation domain is shown below.

```
select from flight_reservation_domain
:www.orbitz.com
where airport1 = MEM and airport2 = SFO
and date1 = 09/20/03
and date2 = 09/30/03
order by URL
filter: Price < 500
```

3.2.3 Mapping of User Queries to Web Forms

Two essential system modules required for mapping a user query to Web forms of the specified domain are the Information Retrieval module and the OntoBuilder agent. Implementation of these modules is beyond the scope of this research as there already exist tools

with satisfactory functionality to perform the tasks of these modules. The information Retrieval module can rely on available search engines, such as Google to provide links to sites with potentially relevant Web forms. Very often the links may contain more than one Web form, so the next task is to identify the relevance of these forms. The OntoBuilder agent, which is the latest iteration of the OntoBuilder system [18], discussed in section 2.1 can successfully perform this task. Given an example URL that contains a Web form of a desired ontology the agent is able to locate semantically similar controls on the input form and the example form. If the number of such controls meets the threshold value used by the agent, then relevance of the input form is validated. OntoBuilder uses the merged ontology common to the example form and the previously processed form(s) for future processing, which enhances its performance. The forms validated as relevant are then processed by the OntoBuilder agent. The agent automatically generates XML files with the Web form ontologies having the query attributes mapped to the form controls. The agent uses a domain-specific thesaurus, which is packaged in a domain plug-in module in our system. The thesaurus is used to acquire the correct mapping of the query attributes to form controls by analyzing their names, select items, associated labels, etc. These activities are simulated in our system by supplying a set of predefined XML ontology files to the Form Filling Module. This module reads the ontology files using the Microsoft XML parser and then uses the WebBrowser ActiveX control to load the Web forms, fill them out based on the ontology mapping, and submit. This approach makes it possible to overcome

the complication of finding hidden attributes of the forms and their values required for direct CGI Get or Post submission.

3.2.4 Record Identification Algorithms

The system uses two different algorithms to identify records encoded into HTML format by remote scripts processing the outputs of online databases. The main test domain of the system was selected to be the flight reservation domain. The nature of this domain implies that, though the records have identical structure, some attributes should be multivalued, having a different number of values from record to record. For example, flight schedules may consist of flights of one or many airlines. To accommodate the need for nesting, which can be used not only for display of multivalued attributes (table 3 shown in Figure 3.2), but also for better record layout (table 2 shown in Figure 3.2 is used to introduce spaces between the records), the majority of travel sites use nested table structures. In some examined cases the nesting went up to 7 levels. All the examined sites, including sites from different domains, use some form of HTML table-based structures to encode the records of their online databases. Therefore the two algorithms that we have developed are oriented to identification of this type of records.

To illustrate this point, a three-level nested table structure is shown in Figure 3.2. All Expedia records have the same table structure; however, their HTML tag structure varies from record to record due to conditions such as different numbers of values in multivalued attributes. This observation made us exploit the table structure of the result pages to

Figure 3.2 HTML code of a single www.expedia.com result record

identify potential records as identical table structures in our HTML Table Tree Algorithm, which captures the hierarchy of nested tables in a tree structure. A simple example of a table tree is shown in Figure 3.3A. Figure 3.3B shows a fragment of the HTML table tree generated by www.expedia.com, where one of the identified identical branches shown in the rectangle corresponds to the record depicted in Figure 3.2. The same table tree structure is reused in our second algorithm.

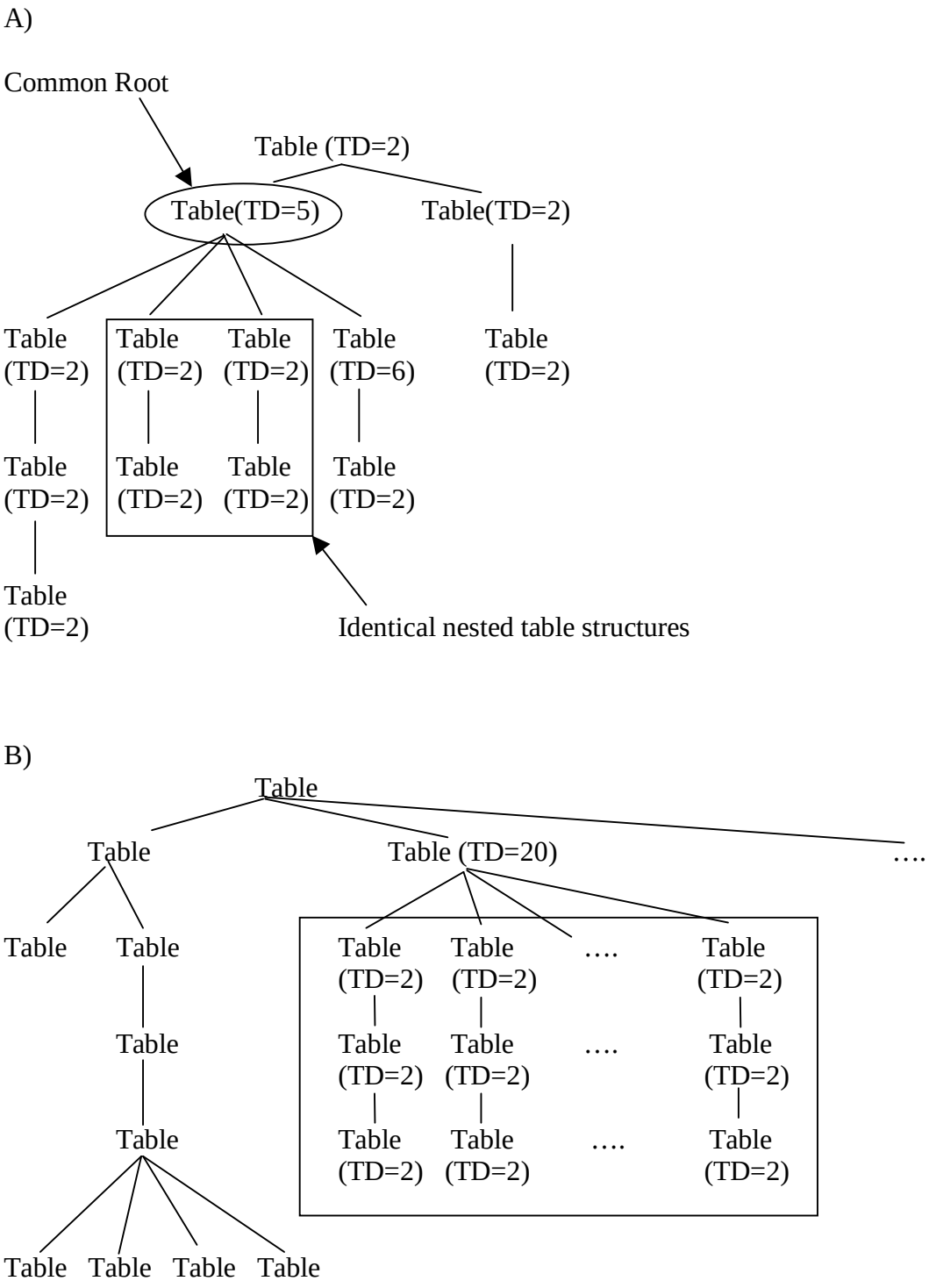


Figure 3.3 A) A generic HTML table tree example. B) www.expedia.com table tree

It is worth noting that the record structure used by www.expedia.com is relatively simple; it contains only one-way flight details. Many other sites use more complex record structures (for example, www.orbitz.com). A sample Orbitz record is shown in Figure 3.4. However, our system is capable of wrapping result pages of a given domain regardless of their structural differences as long as the record structures are table-based, which is the case for the overwhelming majority of sites.

$1815 airfare $5 service fee **$1820 trip cost**	**Delta Air Lines 690** Sun, Mar 16	**5:05a-7:17a** plane change	Memphis (MEM)	Atlanta (ATL)	1
	Delta Air Lines 523 Sun, Mar 16	**8:20a-10:46a**	Atlanta (ATL)	San Francisco (SFO) total duration: 7h 41min	
	American Airlines 566 Sun, Mar 23	**5:12p-10:31p** plane change	San Francisco (SFO)	Dallas/Fort Worth (DFW)	1
	American Airlines 514 Mon, Mar 24	**6:44a-8:11a**	Dallas/Fort Worth (DFW)	Memphis (MEM) total duration: 12h 59min	

Figure 3.4 A sample www.orbitz.com record

3.2.4.1 The Record Identifier Architecture

The architecture of the Record Identifier block is presented in Figure 3.5. The Alternative Algorithm block outlined with a dashed line in Figure 3.5 is not implemented; it is shown only to demonstrate the extensibility of the system architecture that may be useful for extension of the coverage of sites to include those that are not covered by the two implemented algorithms.

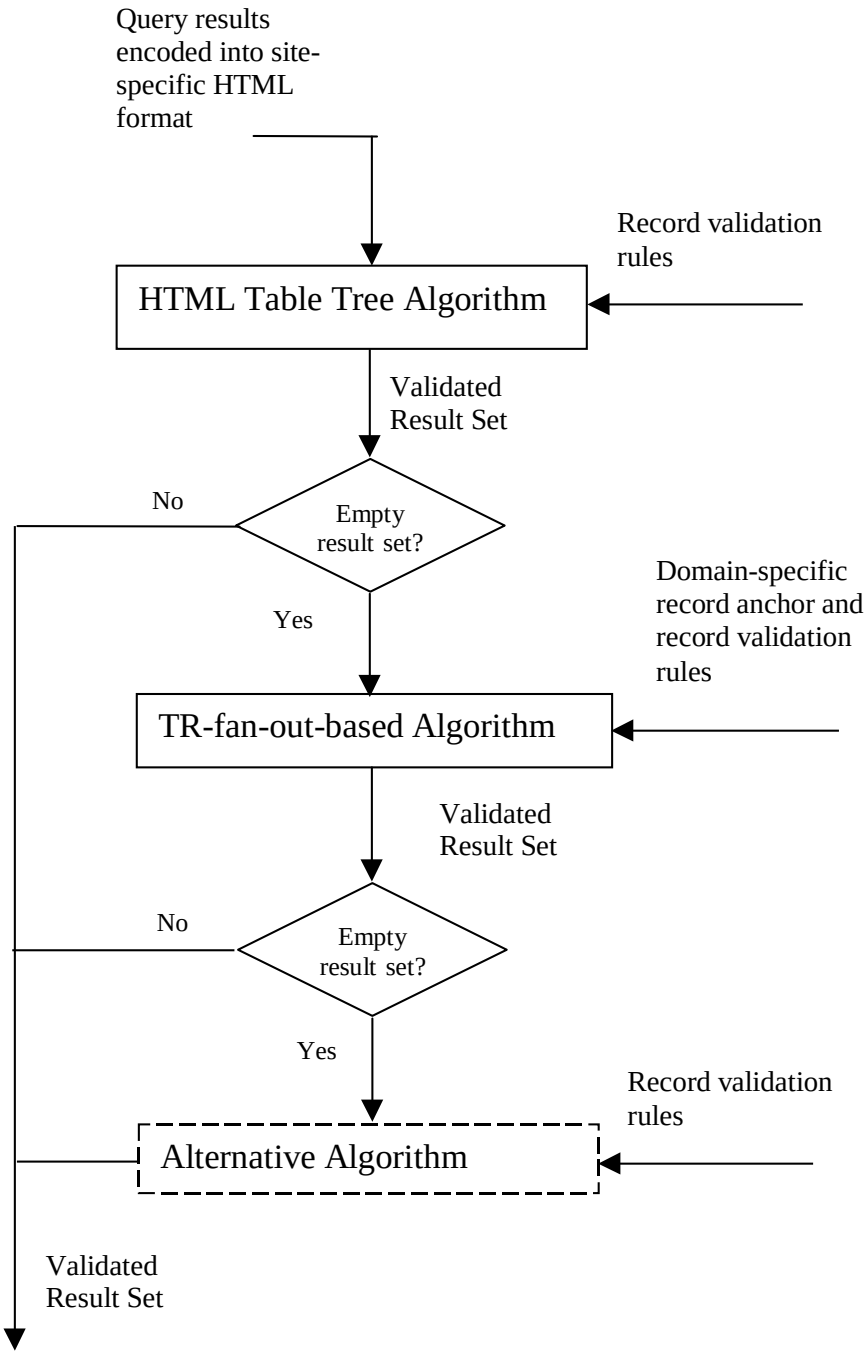


Figure 3.5 The Record Identifier Architecture

The two implemented algorithms cover a certain subset of the most commonly used HTML encodings. However, there are other possible ways to encode records, including non-HTML approaches such as tabulation between the <pre> and </pre> tags. This uncommon HTML encoding can be covered by the third algorithm that is not implemented in the system, shown as Alternative Algorithm in Figure 3.5. It may be based on record layout and ontological validation of potential records rather than on identifying record boundaries as HTML delimiters. For example, copying text from the browser window and pasting it into a text document will almost certainly produce line-feed characters in place of record separators, and though these characters may also appear in the middle of logical records, ontological validation rules may be applied to extract valid records. This approach neglects any HTML encoding and, thus, makes the IE phase based on purely natural language processing techniques. The satisfactory results achieved by the first two algorithms place the development of this NLP-based algorithm outside of the scope of this iteration of our system.

The system architecture is designed to minimize processing time. Therefore the fastest record identification algorithm performs its task first, and the second algorithm is activated only if the former produces an empty result set after the extraction rules are applied to the identified candidate records. The decision on the sequence of activation of algorithms should be based on precision and then on processing time. However, there is no difference in precision of the two implemented algorithms, so the second criterion is used. On

the other hand, the precision of the Alternative Record Identifier is expected to be worse compared with the first two algorithms, and therefore it should be used as a last resort.

3.2.4.2 The HTML Table Tree Algorithm

Following the table tree representation, the Expedia record shown in Figure 3.2 would appear as a straight line tree with table 1 as the root and table 3 as the leaf. Since this record itself is nested in a table containing all the other records (not shown in Figure 3.2) forming a branch of the tree, our goal is to go through all branches of the tree and find all of those having identical table structures. The identified potential branches can be then validated by semantic analysis of the HTML encoded data in these structures followed by data extraction from the positively validated HTML structures. We accomplish this task using the algorithm presented in Figure 3.6.

The algorithm constructs an HTML table tree for a target document and finds the count of <td> tags for every node. The purpose of counting <td> tags is to identify the number of attributes in the tables, which is used for identification of repeated record structures. For this purpose only the <td> tags of the first record of a given table are counted. Though the number of columns in the subsequent rows may vary (if their <td> tags use different colspan attributes), this count is sufficient to correctly identify potential records having the same structure. This approach also takes care of multivalued attributes, which may be encoded in nested tables having different numbers of rows, and thus, different numbers of <td> tags.

```
For each table T of a given page
        Add table hierarchy labels, thus constructing an
        HTML table tree H(T).
For each node T of the tree
        Add tag number attribute.
        Add <td>  - count attribute showing the number of <td>  tags in the first record
        of the table excluding subtables.
        If the node is a leaf then
                Add a Leaf label to the node.
For each leaf L
        If at least one of the neighboring leaves is on the same level and has the same
        <td>  count then
                Add L to a candidate group G(L).
For each group of candidate groups G(L)
        Find the common lowest parent.
        If <td>  counts of branches formed between the parent and the leaves are the
        same on each level then
                Trace back the branch boundaries to the corresponding tags in the HTML
                page.
                Store the corresponding pieces of HTML code S in a string array
                Temp(S).
For each string S
        If validated using domain keywords then
                Add the validated string to an array V
If V is not empty Go To IE
```

Figure 3.6 The HTML Table Tree algorithm

The leaves of the same level having the same <td> count are used to identify potential HTML encoded records, which are the branches of the tree ending at these leaves and beginning at the direct children of the lowest common root of these leaves. Such branches are compared using their <td> counts. The algorithm uses the assumption that the branches are straight, i.e., each level of a given branch contains only one table. This assumption uses the narrow coverage strategy, which can be useful to sift out uninformative pieces of HTML code that can randomly occur between the encoded records. The opposite wide

coverage strategy would use the direct children of the common root as record separators, which would include all subbranching in the potential record space. This strategy is used in the second algorithm covered below.

The identified records are then traced back to the original HTML source code and record delimiter tags are added to this code. The identified area of the source code is ontologically validated record by record, and in the case of positive validation forwarded for IE.

This algorithm is known to have problems identifying record structures if the table tree is low or the informative subbranching is present. The second algorithm covers some of the record structures not covered by the table tree algorithm.

3.2.4.3 The Highest <tr>-fan-out Algorithm

Creation of the Highest <tr>-fan-out algorithm presented in Figure 3.7 was influenced by the highest fan-out heuristic formulated in [10] by Embley et al. The original heuristic uses a DOM tree and finds a tag with the highest fan-out. Given the assumption of multiple encoded records outnumbering the number of attributes in any schema on the target HTML page, the direct children of such highest fan-out element are potential record separators. The original approach uses probabilistic methods to identify the record separator from the set of candidate tags. As mentioned above, we adopted the assumption of tabular-based record encoding. With this assumption in mind, a table with the highest <tr> fan-out is found. However the <tr> tags of this fan-out cannot be treated as the potential record

separators because there maybe multiple <tr> tags which are the direct children and yet not used as record boundaries.

For example, there may be a record structure using every other or every third <tr> tag to encapsulate the record. Moreover, the encoded record may be split by a pair of </tr><tr> tags a few times, where the <tr> tag is a direct child of the highest <tr>-fan-out table. However, the heuristic is useful for early identification of tables that are likely to contain the target records. At this point, an ontological approach is used for record identification. Each domain contains a unique ontological element that can be used as a linguistic anchor showing the approximate location of a record and, what is more important, it can be used to find the exact number of records.

```
For each node T of the table tree H(T) constructed in the HTML table tree algorithm
        Count the number of <tr> tags excluding the tags in subtables and add the <tr>
        tag count attribute.
Construct a sorted array A(TR Count, TableLabel) of the <tr> counts and the identifiers
of the corresponding nodes.
For each element in A from the highest <tr> count to the lowest
        Using table boundaries of the corresponding node trace back to the HTML code
        of this table.
        Find the number N of occurrences of linguistic anchors (such as "$" or "USD" for
        the flight-reservation domain) or groups of such anchors in the identified table
        code.
        Find the first tag RS in the identified code that repeats N times.
        Split the identified code into N+1 pieces using RS as the separator.
        Store the identified pieces of HTML code S, excluding the first one, in a string
        array Temp(S).
        For each string S
                If validated using domain keywords then
                        Add the validated string to an array V.
        If V is not empty Go To IE.
```

Figure 3.7 The Highest <tr>-fan-out algorithm

For most comparison shopping domains, including the flight reservation domain, such an ontological element is price. It can be easily spotted by locating either ”$” or ”USD”. The anchors are specified in the domain plug-in and passed to the Record Identifier at run time. To be considered a legitimate anchor they must be separated by a reasonable number of tags. Otherwise, they form one anchor, which can be the case if one record contains a price group, e.g., base price, fee, and total. After the exact number of records is identified, the record separator is located by finding the first tag that repeats the same number of times between the table boundaries of the candidate table.

If processing of the highest <tr>-fan-out table returns an empty result set, the process repeats for the second highest <tr>-fan-out table and so on until either the returned result set is non-empty or all tables of the page are processed.

Another difference is that this algorithm reuses the HTML table tree created by the previous algorithm instead of constructing a separate DOM tree. This approach optimizes processing time. Just like in the first algorithm, the identified records are then traced back to the original HTML source code and the identified area of the source code is ontologically validated record by record. The approach proved to work well with a number of sites. However the processing time is somewhat higher compared with the first algorithm, due to the need for counting <tr> tags of every table in the page and sorting the counts.

3.2.5 Information Extraction

Validated records output by either of the algorithms are used for extraction of attributes of the target domain-specific schema. The records consist of information atoms delimited by HTML tags or groups of tags. Each atom is tested separately to fit the extraction rules for each attribute of the schema. The extraction rules are based on the domain knowledge associated with each attribute. A decision tree exploring all possible ways to format a given attribute is applied to information atoms using regular expressions to check against a particular formatting. Information atoms may constitute a part of the target attribute, for example, values of month and day of the same date value may be located in two adjacent information atoms. The specified extraction rules are capable of detecting such adjacent atoms and mapping them to a corresponding target attribute. Multivalued attributes of source schemas are mapped to a single target attribute as a list of values. The generic domain independent IE algorithm used in the system is presented in Figure 3.8.

```
For each string in V
        For each information atom (data between tags)
        If mapping to the target schema attributes is successful then
                Add the information atom to the corresponding attribute of the target
                record being populated with values.
        Add new target record.
```

Figure 3.8 The IE algorithm

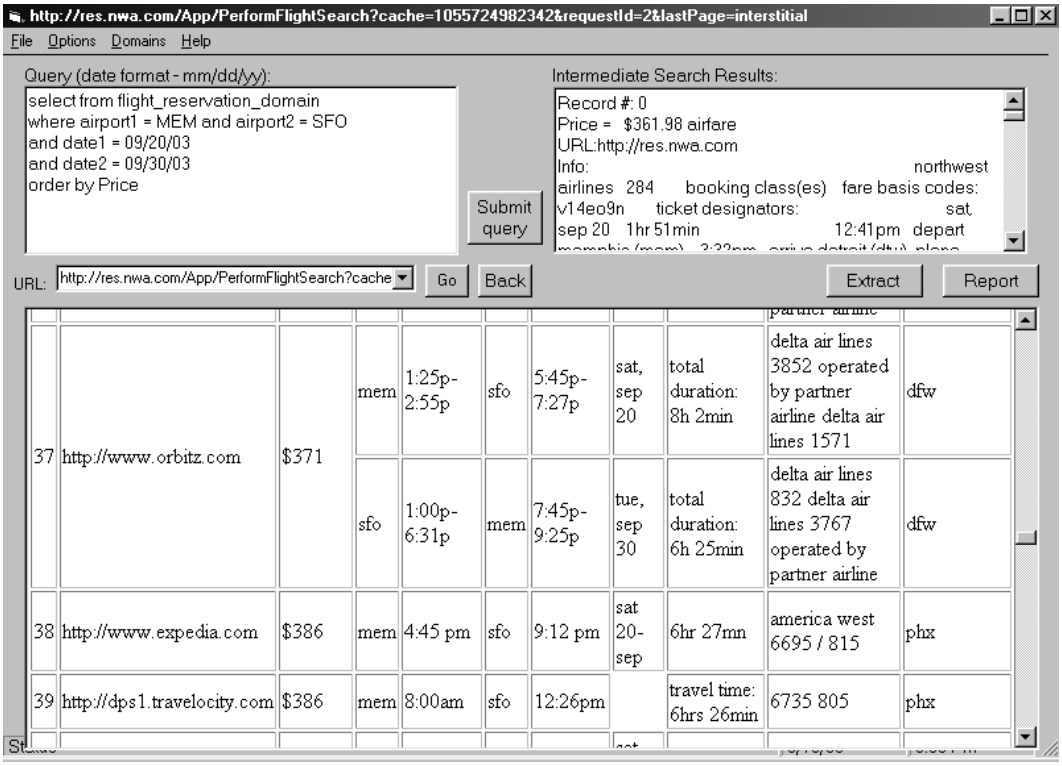
3.2.6 Data Consolidation and Output

After the IE phase is completed, the validated extracted records are stored in a Microsoft ADO recordset object, which allows any SQL manipulations. At this point, Order By and Filter criteria of the user query are applied to the recordset.

The sorted and filtered recordset is then used to generate a report in a dynamically generated HTML page. The target schema of the report is dynamically adaptable to the frequency of attributes in the processed source schemas. The user can customize the policy strictness on inclusion or exclusion of the source attributes by setting a threshold value anywhere between the union of all values (0%) and the intersection (100%).

An example of record consolidation generated by online databases on www.orbitz.com (Figure 3.4) and www.expedia.com (Fugure 3.2) is shown in Figure 3.9. Note that the system can consolidate records with substantially different structures. Orbitz records show two-way flight details, while Expedia records only show one-way flight information. As seen from Figure 3.9, the system also successfully maps differently formatted source attributes to the target schema. Note that the structure of the consolidated records varies to accommodate one-way and two-way flight details. In the case of this domain, the system expects either one of these presentations, which is a part of the domain knowledge used during the IE process, and adapts the output record structure accordingly.

Figure 3.10 shows the report headers with threshold value set to 0% (union) and 100% (intersection).



37	http://www.orbitz.com	$371	mem	1:25p-2:55p	sfo	5:45p-7:27p	sat, sep 20	total duration: 8h 2min	delta air lines 3852 operated by partner airline delta air lines 1571	dfw
			sfo	1:00p-6:31p	mem	7:45p-9:25p	tue, sep 30	total duration: 6h 25min	delta air lines 832 delta air lines 3767 operated by partner airline	dfw
38	http://www.expedia.com	$386	mem	4:45 pm	sfo	9:12 pm	sat 20-sep	6hr 27mn	america west 6695 / 815	phx
39	http://dps1.travelocity.com	$386	mem	8:00am	sfo	12:26pm		travel time: 6hrs 26min	6735 805	phx

Figure 3.9 Consolidated records of flight reservation domain

#	URL	Price	Depart		Arrive		Flight(s)	Connection Airports
			City	Time	City	Time		

#	URL	Price	Depart		Arrive		Date	Duration	Flight(s)	Connection Airports	Class
			City	Time	City	Time					

Figure 3.10 Dynamically adaptable target schema

3.2.7 User Interface

The system GUI (Figure 3.9) allows the user to specify a domain query in the upper left text box. Then after the Submit Query button is pressed the system does not require any human interaction. It automatically finds the Web forms in the specified domain, fills

and submits them, extracts data from the generated result pages, and outputs the query result in a dynamically generated HTML report table. The right upper text box displays identified records of the result page that has been processed prior to their submission to the IE Module. It is primarily used for testing. The system also allows manual navigation to the result pages using the URL box and the WebBrowser ActiveX control and data extraction from these pages by pressing the Extract button. The extracted data from all processed pages is kept in the system and displayed as a report once the Report button is pressed. The Report button can also be useful for regeneration of the report using a different target schema threshold, which can be set in the Options menu.

3.2.8 Domain Plug-in Modules

3.2.8.1 General Information

The system is based on the idea of isolation of domain knowledge in independent plug-in modules separate from the system, some of which can be added in the future to extend the functionality of the system according to the user's needs. Each module is implemented as a collection of a DLL file, which contains all domain-specific processing functions, and two or more XML files. One of them contains domain knowledge for identification of Web forms and their mapping to the user query, such as search keywords, thesaurus, etc. This XML file is not implemented in the current iteration of the system as integration of the Information Retrieval module and OntoBuilder Agent with the system is yet to be done. OntoBuilder output XML files covering ontologies of Web forms used for testing

are currently used as stubs for the missing modules. The second domain plug-in XML file contains a collection all possible attributes in a given domain and their extraction rules. The form mapping file can be modified by the user to add new entries or customize the existing items. The file with extraction rules can be modified to customize existing extraction rules. This may be useful for various reasons, for example a user in England may want to replace a dollar sign with a pound sign in the price extraction rule. Below is a partial XML file with extraction rules used in the flight-reservation domain for the US locale.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Record>
   <RecordAnchor>’$’ Or ’usd’</RecordAnchor>
   <AdditionalRecordValidation>[Time]
   </AdditionalRecordValidation>
   <Attribute>
      <Name>Price</Name>
      <ExtractionRule>Price(’$’ Or
  ’usd’)</ExtractionRule>
   </Attribute>
   <Attribute>
      <Name>Time</Name>
      <ExtractionRule>’*## [ap][m]*’ Or
```

```
  '*##[ap][m]*' Or '*## [ap] *' Or
  '*##[ap]*'</ExtractionRule>
   </Attribute>
   ... other attributes
</Record>
```

The record anchor is used for record identification. A combination of the record anchor and additional record validation rules specified by the first two entries is sufficient to correctly validate records of the flight-reservation domain as each schedule must contain pricing and flight time information. A price attribute by itself may be present in other non-record sections of the page, for example, a hotel add. The time attribute may also appear by itself, for example, in the section of the page showing the time of the report generation. However, there is a very high probability that no other non-record section of the result page would contain both of these attributes. The attribute entries of this XML file identify the attribute name and its extraction rule. The extraction rules can be of three types: 1) a regular expression in single quotes, 2) an attribute name defined in another attribute entry (showed in square brackets), or 3) an extraction function defined in the domain plug-in DLL. Multiple rules can be concatenated with ”And” or ”Or” and grouped with braces. The functions are used for complex attributes that may span over multiple adjacent information atoms and therefore cannot be extracted with a single regular expression. The extraction functions still let the user modify the essential part of extraction rules through their arguments. Multiple argument values can be submitted to the extraction functions as a list of single

quoted values separated by ”Or”, e.g. ’$’ Or ’usd’. In this case each of these values will be applied one at a time in attempt to map the information atoms to the target attribute of the function. The part of the program processing this XML file uses the extraction rules written as regular expressions to map them to the corresponding Boolean IsXXX functions and provide them with the arguments. During the information extraction phase the IsXXX functions are used for mapping information atoms to the target attributes.

Some other XML files may be used by creators of the domain plug-ins as long as they are referenced for the plug-in DLL file. For example, one-to-many relationships like Make - Model in the Car Sales domain can be conveniently represented in an XML file.

The domain plug-in files are expected to be created by the company that distributes the program and provides its maintenance. The initial distribution may come with a limited number of plug-ins while extra plug-ins may be downloaded from an ever-growing library located on the company’s Web site. Due to the complexity of DLL creation the end users will only be able to modify existing plug-ins by changing values in the corresponding XML files to tailor their functionality to the desired locale or other preferences. As mentioned above, the plug-in installation is extremely simple. The DLL and XML files must be copied to specified directories on the user’s machine running Windows. A domain name specified in the query will be used to dynamically link the corresponding DLL, which has the expected locations of the XML files. This way the functionality of the selected domain becomes current to the system until another query using a different domain name is submitted.

3.2.8.2 Creation of Domain Plug-ins

Domain plug-ins capture the target ontology of the corresponding domain of information as it exists on the World Wide Web. Therefore, the first step in creation of a domain plug-in is learning the target ontology by the creators of the plug-in. Various combinations of keywords should be tried to identify the ones that yield the highest recall. Ontological terms (classes), slots, and functions should be learned from studying a large number of Web forms and the corresponding result pages. Once the creators of the plug-in have this knowledge they can make decisions about the implementation. For example, studying of the result pages will answer such important questions as: What combination of linguistic anchors uniquely identifies (with a high degree of probability) target records? Will a single information atom span multiple target attributes? Will a single target attribute span multiple information atoms? Answering questions like these will create a unique set of implementation decisions.

Creation of a new plug-in by modification of an existing plug-in for a different domain is fairly straight forward. The XML files should be redefined with new keywords, anchors, extraction rules, etc. The DLL file should update the references to the new XML files. If there are new IE functions specified in the IE XML file, they should be implemented, while unused functions should be deleted. Likelihood of different relationships between information atoms and target attributes (determined from domain analysis) may also require modifications in the IsXXX functions for IE that can be carried over to the new plug-in. New IsXXX functions should be added to cover new target attributes and the

unused functions should be deleted. These major steps account for about 95% of the effort needed in creation of a new plug-in, the other 5% is other common sense modifications that are difficult to classify.

3.2.8.3 Benefits of Plug-in Architecture

The benefits of using the plug-in architecture are numerous. This architecture allows easy and seamless extension of program functionality by simple addition of new plug-in files that cover new ontologies. There is no need to recompile the program for this purpose.

Many ontologies require similar yet different features. For example, IE of a target attribute Time may need two different approaches in two different ontologies. Isolation of all domain-specific functionality in domain plug-ins prevents incorrect use of functions like IsTime from this example. From the software engineering point of view this architecture is beneficial as it increases cohesion and decreases coupling of program elements.

Finally, extending the number of ontologies covered by the system by creating a new domain plug-in is simpler than doing it any other way. One may just copy an existing plug-in for a different ontology and use it as a template for the new ontology by making all the required modifications. This task may be fairly simple if the two ontologies are not disparate.

It is important to note that the benefits of the plug-in architecture are strictly of the software engineering kind. Use of plug-ins does not improve the power of the ontology-driven data extraction functionality of the system. Yet, it provides a sound organization of

the system that minimizes programming errors and provides ease of functionality extension.

CHAPTER IV

EXPERIMENTAL RESULTS

4.1 Qualitative Comparison with Related Tools

The major contribution of the Chameleon project is the development of novel record identification and IE techniques that are applicable for use in smart agents automatically querying domains of the hidden Web rather than particular sites or pages. Therefore, this section provide a comparison of our system with the related Web IE tools. The developed system has been tested with flight reservation Web sites and produced good results. This domain utilizes very complex HTML structures for encoding records of the online flight-information databases which reflects complexity of the source schemas used in this domain. We have not seen any other tools in the literature that would work in this domain. Therefore, Table 4.2 provides the experimental results of the developed system only. The two last columns of Table 4.2 show how the two developed algorithms cover the source pages. Qualitative comparison with related IE tools is summarized in Table 4.1.

The qualities used in Table 4.1 for system comparison are key for an IE system that can work with a smart Web query agent for crawling the hidden Web. There are no widely used commercial products of this class of software, yet the need for these tools cannot be

Table 4.1 Qualitative comparison with related IE tools

	XWRAP	RoadRunner	WIEN	Stalker	BYU Tool	Proposed System
Capable of wrapper generation (in a given domain) without human supervision, regardless the HTML record structure.	No	Yes	No	No	Yes	Yes (*)
Supports nesting and complex variant record structures.	Yes	Yes	No	Yes	No	Yes
Capable of generation of dynamic domain-specific wrappers independent of formatting or layout changes.	No	No	No	No	Yes	Yes

* Current implementation only supports table-based record structures.

overestimated. Therefore these criteria were selected for the system comparison as they determine the modern-day scientific value of the IE tools being examined.

The first quality listed in the table is the most essential for functionality of the smart agent. The user interaction should be minimized to formulating a query, selecting a search domain and possibly setting some custom search conditions. After that the agent must perform the search process on all the sites that it finds automatically, which includes automatic IE from pages having different record structures within one session. The user should not be involved in structural analysis of the pages, trying various extraction heuristic, or any of that.

The second quality shows the robustness of the examined tools. Complexity of record structures should not deter the IE tool from wrapping the result pages.

The last quality shows if the tool is capable of generation of dynamic wrappers in run time, which is a big advantage for smart agents as it eliminates the need for maintenance of large libraries of static wrappers, which require verification before their use and regenera-

tion every time the formatting or layout of the source page is changed. It also eliminates potential IE errors caused by poorly performed wrapper verification.

The developed system has all of these qualities, while none of the other related IE tools has all of them. This makes the developed system superior and most fit for use with smart Web query agents.

4.2 Testing Domains

As discussed above, the system provides domain-specific information extraction rather than document-specific. The system architecture allows functionality extension by adding more domain plug-in modules implemented as XML and DLL files. Two domain plug-ins are currently implemented. The first plug-in covers form submission and IE in the flight-reservation domain and the second one covers IE in the used-car sales domain. The flight-reservation plug-in has been tested and shown to function correctly with various sites. Table 4.2 presents the results of the testing specifying the record identification algorithm used during the wrapper generation. Recall that the TR-Fan-out algorithm is invoked only when the HTML Table Tree algorithm fails to identify records correctly.

Table 4.2 Sites from which the system successfully extracts data

Flight Reservation Sites from which the System Performs Successful Data Extraction	HTML Table Tree Algorithm	TR-Fan-out Algorithm (activated to work with pages not covered by the Table Tree Algorithm)
www.expedia.com	Yes	--
www.orbitz.com	No	Yes
www.travelocity.com	No	Yes
www.cheaptickets.com	No	Yes
www.delta.com	Yes	--
www.nwa.com	Yes	--

To our best knowledge, there is no other tool performing IE from this domain. The used-car sales domain happens to be less complex and its ontology was implemented by creators of the BYU tool [9]. This tool is also capable of dynamically wrapping HTML content based on a manually constructed domain ontology. Since this tool is functionally the closest to Chameleon, their performances in the used-car sales domain are compared in table 4.3. The analysis of this comparison is provided in section 4.4.

4.3 Information Extraction Performance Metrics

Traditionally performance of an information extraction task is measured by precision, recall, and the F-measure that combines precision and recall [8]. Generally speaking, recall is a measure of how much information was correctly extracted and precision shows the reliability of the extracted information. The following formulae can be used to find precision and recall:

$$Precision = \frac{N_C}{N_P}$$

$$Recall = \frac{N_C}{N_T}$$

where N_C is the number of correct answers (true positives), N_P is the number of answers produced (true positives and false negatives), and N_T is the total of possible corrects (false positives and true positives). The formulae can produce real numbers between 0 and 1, which are often expressed as a percentage. The F-measure quantifies the performance in a

single value, which is often used to compare the performance of different systems. Below is the formula for the F-measure.

$$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

where P is precision, R is recall, and β is a parameter that can be set to favor either recall or precision. If $\beta = 1$ recall and precision have equal weights. In this case the F-measure is called the F1 score [8].

To quantify the experimental results presented in the next section we need to determine what constitutes correct answers, answers produced, and total of possible corrects. Counting of these values can be done on the basis of a single HTML source page. The total number of correct answers is the number of values in the intersection of the source table values and extracted values. The number of answers produced is the number of extracted values. The total of possible corrects is the number of values in the source table. This quantifies extraction quality from a single source, which is not enough to objectively represent extraction power of a given IE tool. To achieve the objective measure of extraction power the tool should be tested on a large number of randomly selected record-structured HTML sources given they belong to the information domain being tested. Then the average values of precision, recall, and the F1 score (or the F-measure) will identify the true information extraction power of the tool in study. The next section provides such metrics for Chameleon and the BYU tool.

4.4 Quantitative Comparison with the BYU Tool

This section presents results of comparative testing of the BYU tool and Chameleon. As explained in Sections 4.1 and 4.2, the BYU tool was selected because of its analogous ontology-driven IE implementation, which allows domain-specific IE as opposed to page-specific IE of other extraction tools. The comparison that is provided below reveals the superiority of Chameleon's record identification algorithms and IE implementation.

To perform a fair comparison of the tools the source pages were obtained from sites whose links were retrieved by "used cars" search keyword combination at www.yahoo.com and from an online listing of Mississippi car dealers. Different queries were performed on different selected sites to generate a minimally biased set of source pages. The only criterion used in selection of the source pages was that they should comply with the major elements of ontology (domain knowledge) specification, namely, the Price and Make attributes must be present in the record structures.

The Chameleon UsedCars domain plug-in supports extraction of a slightly larger set of target attributes (Price, Year, Make, Model, Mileage, Color, and VIN) compared with the set of attributes specified in the BYU Car ontology (Year, Make, Model, Mileage, Price, and Phone number). This slight difference does not preclude fair comparison between the performances of these tools.

The BYU tool was tested on January 11, 2004 using the online demo available at www.deg.byu.edu. The Car ontology was used for the test.

Table 4.3 provides the experimental data in terms of whether or not the extraction was successful, precision, recall, and the F1 score. Chameleon's success rate is almost twice as high as that of the BYU tool. Chameleon's precision and recall values are also higher. The precision rate is very high due to rigorous validation procedures implemented in Chameleon. The www.carsearch.com page was not extracted by Chameleon due to the presence of an ad that had all record identification anchors in it. As a result the ad was extracted instead of the records. The other two pages not extracted by Chameleon used a list instead of an HTML table to group records. As mentioned previously, Chameleon does not work with lists.

The important conclusion that can be drawn from this comparison is that the probabilistic approach to record identification used in the BYU tool is not very practical. The two record identification algorithms proposed in this thesis operate on much more sound assumptions, which results in the superior IE mechanism.

Table 4.3 Comparison of the BYU tool and Chameleon performances

No.	Web Site	BYU Tool				Chameleon			
		Extracted?	Precision	Recall	F1 score	Extracted?	Precision	Recall	F1 score
1	autos.yahoo.com	No				Yes	100%	100%	100%
2	www2.nada.org	Yes	100%	100%	100%	Yes	100%	100%	100%
3	www.arcardealers.com	Yes	100%	83%	91%	Yes	100%	100%	100%
4	www.autoselect.com	Yes	89%	100%	94%	Yes	100%	100%	100%
5	www.autotrader.com	Yes	43%	50%	46%	Yes	100%	100%	100%
6	www.bayside-ms.fivestardealers.com	Yes	100%	100%	100%	Yes	100%	100%	100%
7	www.bertallen.com	Yes	89%	100%	94%	Yes	100%	100%	100%
8	www.blackburn.fivestardealers.com	No				Yes	100%	100%	100%
9	www.carbuyer.com	No				Yes	100%	100%	100%
10	www.carlhogan.com	No				Yes	100%	100%	100%
11	www.carmax.com	No				Yes	100%	100%	100%
12	www.cars.com	No				Yes	100%	100%	100%
13	www.carsdirect.com	Yes	57%	13%	21%	Yes	91%	33%	48%
14	www.carsearch.com	No				No			
15	www.cartrackers.com	No				No			
16	www.clementscadillac.com	Yes	100%	100%	100%	Yes	100%	100%	100%
17	www.dealsonwheels.com	No				No			
18	www.eastbrooktoyota.com	No				Yes	100%	100%	100%
19	www.edmunds.com	No				Yes	100%	100%	100%
20	www.enterprise.com	Yes	100%	100%	100%	Yes	100%	100%	100%
21	www.graydaniels.com	No				Yes	100%	100%	100%
22	www.herrin-gear.com	No				Yes	100%	100%	100%
23	www.holmandealerships.com	Yes	93%	100%	96%	Yes	100%	100%	100%
24	www.hondadelercombrookhaven.com	Yes	98%	100%	99%	Yes	100%	100%	100%
25	www.jimrobinsongroup.com	Yes	100%	83%	91%	Yes	100%	100%	100%
26	www.motorzoo.com	No				Yes	100%	100%	100%
27	www.paulmoak.com	Yes	88%	100%	94%	Yes	100%	100%	100%
28	www.rogersusryhonda.com	No				Yes	100%	100%	100%
29	www.ropertoyota.com	No				Yes	100%	100%	100%
30	www.southernimports.com	Yes	87%	100%	93%	Yes	100%	100%	100%
31	www.turanfoley.com	Yes	100%	100%	100%	Yes	100%	100%	100%
32	www.tuscaloosatoyota.com	No				Yes	100%	100%	100%
33	www.usedcars.com	Yes	100%	100%	100%	Yes	100%	100%	100%
	Total number of sites extracted	16				30			
	Averages based on extracted sites		90.25%	89.31%	88.70%		99.70%	97.77%	98.28%

CHAPTER V

CONCLUSION

5.1 Summary

The hidden Web is beyond the reach of the Internet search engines, yet problems such as automated comparison search require mechanisms allowing autonomous querying of multiple sites of the hidden Web. To date, this problem has not been successfully solved. In this thesis we have developed a smart agent for querying domains of the hidden Web. We have shown feasibility of the proposed system architecture that allows completely autonomous querying of a selected domain of the hidden Web consisting of a number of Web sites identified by the agent at run time. The research presented in the thesis concentrated on record identification and information extraction from semistructured HTML encoded records generated by online databases, and consolidation of the extracted data in a record-set that can be used for further conventional SQL-like queries. The resulting tool is able to perform these operations without any prior knowledge of format, structure, or layout of the result pages generated by online databases, as long as all of the result pages belong to the same information domain. This is achieved by providing domain plug-ins to the main program that performs generic processing. The domain plug-ins contain the information about the ontology and IE heuristics of the corresponding domain. The plug-in architec-

ture allows easy extension of the program functionality by adding new plug-ins covering new domains of Web sites.

The record identification algorithms and the IE technique have been shown to perform substantially better than the closest ontology driven IE tool. The BYU tool selected for the comparison is the only widely known tool that is also capable of domain-specific IE. This shows the importance of the contribution of the thesis in this area of research.

5.2 Future Work

The current implementation of the system leaves out development and integration of the Information Retrieval Module / Validator and OntoBuilder Agent (Figure 3.1). While there have been a few research projects that can be utilized in development of each of these modules, the future research should be aimed toward tailoring their design for integration with the Chameleon system in a seamless fashion.

The future research should also concentrate on increasing the power of the Record Identifier / Validator module to increase the coverage of the source documents and possibly cover non-table-based records.

The organization of domain plug-ins can be improved for better domain coverage to ease their work with complex domains. Submission of multipage Web forms and IE from multiple result pages within one site are currently not supported. The future research should find the way to overcome these difficulties.

Finally, automation of domain plug-in creation is another area worthy of future research. The OntoBuilder agent is the first step in this area. The agent automatically creates and matches ontologies of single-page Web forms. However, the agent has an unacceptably high error rate. On the other hand, the OntoBuilder tool for semiautomatic ontology generation relies on human interaction and domain knowledge to correct the errors. A common ontology based on a high number of learning examples from a given domain can possess fairly high precision and recall. It seems that human interaction is inevitable for creation of a sound domain plug-in, but the human effort can be substantially decreased with development of computer aided design tools for this task.

REFERENCES

[1] B. Adelberg, "NoDoSE: A Tool for Semi-Automatically Extracting Structured and Semi-Structured Data from Text Documents," *Proceedings of SIGMOD Record 27*, 1998, vol. 2, pp. 283–294.

[2] M. Calif and R. Mooney, "Relational Learning of Pattern-Match Rules for Information Extraction," *Proceedings of the ACL Workshop on Natural Language Learning*, Spain, July 1997.

[3] L. Chen, H. Jamil, and N. Wang, "Automatic Wrapper Generation for Semi-Structured Biological Data Based on Table Structure Identification," *1st International Workshop on Biological Data Management - BIDM 03*, Prague, Czech Republic, 2003.

[4] L. Chen and H. M. Jamil, "On Using Remote User Defined Functions as Wrappers for Biological Database Interoperability," *International Journal of Cooperative Information Systems*, vol. 12, no. 2, March 2003, pp. 161–195, Special Issue on Biological Databases.

[5] V. Crescenzi and G. Mecca, "Grammars Have Exceptions," *Information Systems*, vol. 23, no. 8, 1998, pp. 539–565.

[6] V. Crescenzi, G. Mecca, and P. Merialdo, "RoadRunner: Towards Automatic Data Extraction from Large Web Sites," *Proceedings of VLDB*, 2001, pp. 109–118.

[7] R. B. Doorenbos, O. Etzioni, and D. S. Weld, *A Scalable Comparison-Shopping Agent for the World Wide Web*, Tech. Rep. UW-SCE-96-01-03, Department of Computer Science and Engineering, University of Washington, 1996.

[8] L. Eikvil, *Information Extraction from the World Wide Web: A Survey*, Tech. Rep. 945, Norwegian Computing Center, 1999.

[9] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y.-K. Ng, D. Quass, and R. D. Smith, "Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages," *Proceedings of DKE 31(3)*, 1999, pp. 227–251.

[10] D. W. Embley, Y. S. Jiang, and Y.-K. Ng, "Record-Boundary Discovery in Web Documents," *Proceedings of SIGMOD*, 1999, pp. 467–478.

[11] N. Kushmerick, D. S. Weld, and R. B. Doorenbos, "Wrapper Induction for Information Extraction," *Proceedings of IJCAI (1)*, 1997, pp. 729–737.

[12] A. H. F. Laender, B. Ribeiro-Neto, and A. S. da Silva., "DEByE - Data Extraction by Example," *Data and Knowledge Engineering*, vol. 40, no. 2, 2002, pp. 121–154.

[13] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. D. Silva, and J. S. Teixeira, "A Brief Survey of Web Data Extraction Tools," *SIGMOD Record*, June 2002, vol. 31, pp. 84–93.

[14] S. W. Liddle, S. H. Yau, and D. W. Embley, "On the Automatic Extraction of Data from the Hidden Web," *Proceedings of ER Workshops*, 2001, pp. 212–226.

[15] L. Liu, C. Pu, and W. Han, "XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources," *Proceedings of ICDE*, 2000, pp. 611–621.

[16] G. Mecca, P. Atzeni, A. Masci, P.Merialdo, and G. Sindoni, "The ARANEUS Web-Base Management System," *Proceedings of SIGMOD Record 27*, 1998, vol. 2, pp. 544–546.

[17] Microsoft, ".NET Passport," http://www.microsoft.com/net/services/passport/, (current 12 Sep. 2003).

[18] G. A. Modica, A. Gal, and H. M. Jamil, "The Use of Machine-Generated Ontologies in Dynamic Information Seeking," *Proceedings of CoopIS*, 2001, pp. 433–448.

[19] I. Muslea, "Extraction Patterns for Information Extraction Tasks: A Survey," *Proceedings of The AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[20] I. Muslea, S. Minton, and C. Knoblock, "Hierarchical Wrapper Induction for Semistructured Information Sources," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 4, no. 1/2, 2001, pp. 93–114.

[21] S. Raghavan and H. Garcia-Molina, *Crawling the Hidden Web*, Tech. Rep. 2000-36, Computer Science Department, Stanford University, December 2000.

[22] S. Soderland, "Learning Information Extraction Rules for Semi-Structured and Free Text," *Machine Learning*, vol. 34, no. 1-3, 1999, pp. 233–272.