WebCat – Information Integration for Web Catalogs

by

Attila Barta

Department of Computer Science
University of Toronto
Toronto, Canada
September 1998

A thesis submitted in conformity with the requirements
For the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Copyright © - 1998 by Attila Barta

This thesis is dedicated to:

Tibor, Tinu, Daniel, Corina, Laura.

Abstract

An important part of the World Wide Web (Web) is organized as collections of HTML pages, interrelated by hyperlinks, and arranged in some kind of hierarchy. The problem that we address is that of automating the process of searching through several collections of HTML pages, in which pages are interrelated by hyperlinks and the collections have a common domain. In order to automate the search we integrate these collections. We are interested in only those collections for which the common domain can be organized according to some kind of hierarchy. In this thesis, we define such collections as being *Web catalogs* and we present WebCat, a paradigm to integrate such Web catalogs. In addition, we present an implementation for this paradigm for the domain of clothing catalogs.

Integrating information sources is a problem addressed by Information Integration Systems (IIS). In this thesis we argue that the present IIS have difficulties in integrating Web catalogs. The difficulties are created by the lack of query capability of Web catalogs and by the ownership of the Web catalogs. We overcome these problems by extracting the schema of the Web catalogs augmented with the attributes that we are interested in, then we integrate them and store them locally. Once the data is stored locally, it can be queried. Consequently a query works as follows: first we retrieve the records from the local database that satisfy the query criteria; then, for each record, we retrieve the associated HTML pages from the Web.

The WebCat methodology can be divided into two distinct methods: first, a method for wrapper building for Web catalogs, and second, a method of integrating this information. We call this method Information Integration with Semantics since it relies on our knowledge of the domain to which the Web catalogs belong. For building wrappers for Web catalogs, we use a novel technique; specifically, we use the WebOQL query language in the core of our wrappers.

Acknowledgments

First of all, I would like to thank Professor Alberto O. Mendelzon, my supervisor, for suggesting the topic of this thesis and for his continuos support, advice, and feedback. Without his guidance, this thesis would have not existed. I would also like to thank Professor Ken Sevcik, my second reader, not only for his valuable recommendations, but also for his availability and kindness.

I owe a lot of thanks to Dr. Daniel Marcu, for the time that he put into reading early drafts of this thesis and for suggesting its structural organization. I own a lot of thanks to my colleague student Daniel Marcu for his advice, guidance, and support in my early days of the graduate program, when I was totally naïve about the new world that I was entering.

Many thanks to my colleagues from the Database group, Gustavo (Gus) Arocena and George (Georges) Mihăilă, for the criticism and suggestions. A special thanks also to Gus for his endless patience in explaining me the "secrets" of WebOQL, on which I partially based my work.

I am grateful for the financial support I have received from the University of Toronto, the Center for Advanced Studies of IBM Canada, and Professor Alberto O. Mendelzon.

Special thanks I owe also to some of the people who have help me get here in the first place: Professor Tövisy, whom I have never had the opportunity to thank for trusting my academic potential in the very beginning; Professor Ion Lungu, who has offered me the great opportunity to co-author a book with him, and who has offered constant advice and support; Professor Gheorghe Sabău, who supported and adviced my previous academic "life"; and finally Professor Viorel Marinescu, for his friendships and generous heart.

On the non-scientific side of my life, my family never ceased to be my most faithful supporter. My cousins Emma and Eva always trusted me even when I was out of trust. I owe them a big: "Köszönöm szépen!" for everything. My "ex-father in law" "Boss" Vaida always was a support for me, to an extent matched only by a real father. For him: "Mulțumesc mult pentru tot!". And finally, my parents, who never ceased to be my most

faithful and devoted friends in spite of being thousands of miles away. Their support and teaching was invaluable. They knew nothing about science, but what they taught me proved to me more essential and fundamental than all my University courses put together. There are no words to thank them for that. “Hálás köszönet Drága Szűleim”.

Almost part of the family were Diana and Radu, whom I have to thank for their support and not only, especially during the last part of this work.

Aside from my parents, teachers, and professors, I believe that those who I owe the most are my friends. I include in this category also the persons with whom, for a glimpse of time, we shared our life. Their presence and personality enriched my life and influenced me to become what I am today, in good or bad. Because of them, I consider myself to be one of the most fortunate persons in the world. This thesis is dedicated to them.

CONTENTS

LIST OF FIGURES 8

CHAPTER 1: INTRODUCTION 9

1.1 MOTIVATION 9
1.2 THE PROBLEM 10
1.2.1 Information Integration Systems 11
1.2.2 Limitations of the IIS approach 12
1.2.3 Overview of the WebCat approach 13

CHAPTER 2: RELATED WORK 16

2.1 INTRODUCTION 16
2.2 EXTRACTING INFORMATION FROM THE WEB USING WRAPPER ARCHITECTURES 16
2.3 AUTOMATING WRAPPER GENERATION 17
2.4 SOFTWARE AGENTS 18
2.5 LANGUAGES TO QUERY THE WEB 19
2.6 WEB SITE MANAGEMENT SYSTEMS 20
2.7 COMMERCIAL PRODUCTS 21

CHAPTER 3: BUILDING WRAPPERS USING WEBOQL 23

3.1 MOTIVATION 23
3.2 WEBOQL 24
3.2.1 Overview 24
3.2.2 Operators 25
3.2.3 Navigational Patterns 28

CHAPTER 4: WEB CATALOGS 30

4.1 WEB CATALOGS, AN EXAMPLE 30
4.2 DATA MODEL 31
4.2.1 Vertices and Arcs 32
4.2.2 Building Record IDs 33
4.2.3 An Instance of the Data Model 33
4.3 MAPPING REAL CATALOGS TO WEB CATALOG 35
4.4 CLASSIFICATION OF WEB CATALOGS 36

CHAPTER 5: THE WEBCAT METHOD 39

5.1 LOCATION OF THE INFORMATION 39
5.2 INFORMATION INTEGRATION WITH SEMANTICS 40
5.2.1 Building the ontology 41
5.2.2 Mapping a Web catalog to an ontology 42
5.3 THE WEBCAT METHOD 44

CHAPTER 6: THE WEBCAT ARCHITECTURE 46

6.1 “REAL WORLD” WEB CATALOGS 46
6.2 THE WEBCAT ARCHITECTURE 47
6.3 INTEGRATOR COMPONENTS 49

6.3.1 Wrappers ... *49*
6.3.2 StruViz ... *50*
6.3.3 Domain Knowledge-base ... *52*
6.3.4 The Integrator ... *52*
6.4 USER COMPONENTS ... 53
6.4.1 Apparel WebCat ... *53*
6.4.2 The Query Engine ... *54*
6.4.3 The User Interface ... *55*
CHAPTER 7: WEBCAT WRAPPERS ... 58
7.1 BACKGROUND ... 58
7.2 WEBCAT WRAPPERS ... 58
7.2.1 First Level ... *59*
7.2.2 Sub-tree Group ... *64*
7.2.3 Second Level ... *65*
7.2.4 Fourth Level ... *70*
7.3 THE WRAPPER ... 74
CHAPTER 8: CONCLUSION ... 76
8.1 CONTRIBUTION ... 76
8.2 SHORTCOMINGS OF THE DESIGN ... 77
8.3 FUTURE WORK ... 78
8.3.1 AST Extractor ... *78*
8.3.2 AST Compare ... *80*
BIBLIOGRAPHY: ... 81
APPENDIX 1 ... 86
APPENDIX 2 ... 91

LIST OF FIGURES

FIGURE 1: A COMPUTER SCIENCE TEXTBOOK DATABASE 25
FIGURE 2: OUTPUT OF QUERY Q1 26
FIGURE 3: OUTPUT OF QUERY Q2 27
FIGURE 4: QUERY Q3, Q4, Q5, Q6 ANT THEIR OUTPUT 27
FIGURE 5: WEBCAT DATA MODEL 32
FIGURE 6: INSTANCE OF WEBCAT DATA MODEL, THE DATA SCHEMA OF EDDIE-BAUER CATALOG 34
FIGURE 7: ALGORITHM TO COMPUTE THE LEAF'S NEW POSITION 43
FIGURE 8: THE WEBCAT ARCHITECTURE 48
FIGURE 9: THE EDDIE-BAUER STRUCTURE VISUALIZED BY STRUVIZ 51
FIGURE 10: THE APPAREL WEBCAT USER INTERFACE 56
FIGURE 11: THE APPAREL WEBCAT RESULT FILE 57
FIGURE 12: EDDIE-BAUER CATALOG HOMEPAGE (LEVEL 1) 60
FIGURE 13: FRAMES IN THE CATALOG HOMEPAGE 61
FIGURE 14: AST FOR LEVEL 1 63
FIGURE 15: WEBOQL QUERY TO EXTRACT THE LINKS TO LEVEL 2 64
FIGURE 16: EDDIE-BAUER CATALOG, LEVEL 2 66
FIGURE 17: WEBOQL QUERY TO EXTRACT THE LINKS TO LEVEL 3 67
FIGURE 18: AST FOR LEVEL 3 AND LEVEL 4 (PRODUCT) 68
FIGURE 19: WEBOQL QUERY TO EXTRACT THE LINKS TO PRODUCTS 69
FIGURE 20: WEBOQL QUERY TO EXTRACT THE "SALE" ATTRIBUTE 69
FIGURE 21: WEBOQL QUERY TO EXTRACT THE "NEW" ATTRIBUTE 69
FIGURE 22: EDDIE-BAUER CATALOG, LEVEL 4 (PRODUCT LEVEL) 70
FIGURE 23: WEBOQL QUERY TO EXTRACT THE PRICE FOR THE PRODUCTS 71
FIGURE 24: WEBOQL QUERY TO EXTRACT THE DESCRIPTION FOR THE PRODUCTS 71
FIGURE 25: AST FOR LEVEL 4 (PRODUCT) 72
FIGURE 26: WEBOQL QUERY TO EXTRACT THE SIZE TYPE FOR THE PRODUCTS 73
FIGURE 27: WEBOQL QUERY TO EXTRACT THE SIZE AND COLOR FOR THE PRODUCTS 73

Chapter 1: Introduction

1.1 Motivation

Suppose, for example, that you want to buy a new pair of gloves and suppose that you are unhappy with the choice at the closest department store. If you are an Internet enthusiast who has experience purchasing over the Web, you probably know the URLs of a few department store chains and clothing product catalogs. If this were the case, you would go to one or two known sites and spend some time browsing in order to find a pair that suits you. It will take you a while because there is no consistent way for vendors to classify the products they sell. In some catalogs, you may find gloves in the "winter clothes" section, while in others, in the "accessories" section. But even if you take the pain to browse through all sections, how many web sites do you expect to visit? How many do you already know? Obviously, not too many...

It is very likely, therefore, that you would end up doing a more extensive search. Unfortunately, current search engines will help you find not only the list of vendors that sell gloves, but also web pages that are completely irrelevant to your goal. For example, when you search for "glove" on AltaVista, the page that is ranked as being the most relevant contains the text shown below. The text is presented against a photograph of a baseball player who wears a baseball glove on the top of his cap.

“Cincinnati Reds' Barry Larkin, dressed like teammate Deion Sanders, in background, wears a batting helmet with a glove attached to the top to kid around with Sanders prior to the game against the St. Louis Cardinals Saturday, June 28, 1997, in Cincinnati. Sanders was hit on top of the head by a fly foul ball in left field during Friday night's game.”

It is not only that search engines are traditionally bad at returning web pages that are relevant, but it is also the case that they are incomplete. A recent study has shown that current search engines cover only 40% of the Web, with the best engine covering only 34% [LG]. Unfortunately, even if you spend your time browsing through 10 catalogs, there is still no guarantee that the gloves that you buy will be the best choice given all the price, color, material, and design constraints that you initially had in mind.

In this thesis, we explore a different paradigm for information integration over the Web: in our approach, we build a system that integrates in virtual catalogs information sources that pertain to a common domain and that enables querying according to complex search criteria. If such a virtual catalog is built beforehand, you will not only be able to find the gloves of your dreams very quickly, but you will also be guaranteed that they satisfy the criteria that you have specified such as price, color, size, etc..

1.2 The Problem

An important part of the Web, is organized as collections of HTML pages, interrelated by hyperlinks, and arranged in some kind of hierarchy. These hierarchies could reflect the structure of an organization, if we refer to the home page of an organization; could reflect knowledge of human nature, if we refer to the home page of a library or bookstore, for example; and could reflect the product categories, if we refer to a product catalog.

Given this diversity, it is natural to ask ourselves, what happens if we want to find some information that resides in one of the pages of such a collection? In this case, we have to follow some navigational path from the main page of the collection until we find the page containing the desired information. Usually this is a simple task if the hierarchy reflects the structure of an organization, but it becomes difficult if the hierarchy reflects a product catalog. This difficulty is induced by the different classification used by vendors. Using the example above, gloves can be found in some catalogs at the “accessories” section, and also can be found at the “leather products” section. Furthermore, if we want to consult several catalogs presenting the same products, we have to locate the catalogs on the Web, and we have to “navigate” through the pages until we find the data of interest.

The problem that we address is that of automating the process of searching through several collections of HTML pages, in which pages are interrelated by hyperlinks and the collections have a common domain. We are interested in only those collections for which the common domain can be ordered according to some kind of hierarchy.

In this work, we define such collections as being *Web catalogs* and we present WebCat, a paradigm to integrate such Web catalogs. In addition, we present an implementation for this paradigm for the domain of clothing catalogs. For the rest of the

thesis, for simplicity, we refer to such domains, which have a hierarchical structure, as Web catalogs or catalogs. We refer to a collection of integrated catalogs as a *virtual catalog*. A virtual catalog acts like a collection of real catalogs; for the user the location of real catalogs, and the classification criteria used by each individual catalog are totally transparent.

The original idea of WebCat comes from the product catalogs that can be found on the Web. We studied a method to integrate several product catalogs. As soon as we succeeded, we learned that several other sources on the Web have a similar structure with product catalogs, and are therefore suitable for integration using the WebCat method.

Although integrating several catalogs in a virtual catalog is a problem of Information Integration Systems (IIS), this task pushes existing IIS to their limits. In order to present the need for our system, we present first the capabilities of IIS, and next we underline why these systems are not appropriate for our problem.

1.2.1 Information Integration Systems

The problem of integrating data from multiple sources is by no means a new one. It originated at the same time with the distribution problem for the first database systems. All these systems were capable of integrating data distributed over a wide area and furthermore, the federated systems were capable of managing data from heterogeneous database management systems. All these systems were fully functional database systems; they could perform queries as well as transactions.

The advances in storage devices made possible the accumulation of data as never before. The emergence of the Internet, as well as the intranets, made possible the access to this huge collection of stored data and posed a new challenge, the querying of data kept in these multitude of formats. Information Integration Systems (IIS) address precisely this problem: IIS are capable of querying data from multiple databases, semi-structured sources, and totally unstructured sources, such as file systems. Moreover, the database management systems, integrated by IIS, can have various degrees of heterogeneity, including different data models. In addition, the existence of a uniform schema that mediates among different data sources, which was a requirement for the federated databases, is no longer assumed.

The IIS are built by means of wrapper/mediator architectures. Wrappers encapsulate data sources and mediate between them and the mediators. For each new information source a new wrapper has to be built. That is, each source has its own wrapper. Wrappers take advantage of the query possibility on the underlying sources and can handle large amounts of data. The mediators are responsible for the schema integration and query related activities such as: query planing, query optimization and query execution.

The sources wrapped by current IIS range from relational databases, object-oriented databases, and image databases to bibliographic databases, plain text files, and legacy systems. The emergence of the World Wide Web (Web) imposes a new challenge to IIS, namely: the integration of Web based information sources. Our problem, integrating Web catalogs, represents a instance of this problem.

1.2.2 Limitations of the IIS approach

The existing IIS systems (we give references to them in the next chapter) are rather inappropriate to integrate Web sources. There are two main reasons. The first one relates to how IIS wrappers are built and the second one relates to the ownership of the information.

The first reason, the inadequacy of the IIS wrappers, resides in the following:

- The IIS wrappers take advantage of the query capability of the information sources that they wrap. In the case of Web catalogs, these query capabilities do not exist.
- The size of the data collection is another reason. In the existing IIS the wrapped sources are usually full-capability databases with possibly millions of records; Web catalogs are of a relatively small size on the order of thousands of records, thus it is not justified to build a complex wrapper for such small sources.
- Catalogs are navigational sources. That is, in order to retrieve some information, we have to follow a certain path. The existing IIS are ill suited for this task.

- Because we know the domain of definition of the Web catalogs, we can perform in an automated fashion some parts of the integration.
- The IIS use a “flat” schema, compared with a hierarchical one for the catalogs.
- Web catalogs have an important feature, which make them different from the regular IIS: unique ID for the records across the collection of information sources. This is due to the URL that is associated with each record. We use these URLs as record IDs. This approach is different from the traditional IIS approach in which the system builds the IDs.

The second reason resides in the ownership of the information. The IIS were primarily built to integrate information from large corporations that used data kept in a multitude of formats. Because of this characteristic, data ownership was not an issue for the IIS. The situation changes with the Web catalogs that have different owners. While in the case of IIS the data sources were totally accessible and any information related to sources, such as data schema, were available, in the case of the Web catalogs such information about the sources does not exist. Furthermore the Web catalog owners are rather reticent in any cooperation. Moreover, even if the owners agree to make any information available, a common exchange format is needed.

This shortcoming of the exchange format is long time known, and new emerging standards, such as Extensible Markup Language (XML), are proposed to improve the data exchange, and to allow the information extraction without cooperation of owners. Despite all these advances, the XML standard, as any new standard, will need time to mature and gain acceptance. Consequently, it will have to be implemented by each information provider. But often, it is the case that the information providers are not willing to share their data. An example is the clothing catalog owners, who are not willing to merge their data in order to allow comparison-shopping.

WebCat overcomes this problem since it does not require any cooperation from the information owner. Once the information is accessible for the public, WebCat extracts the schema of the source augmented with the several other pieces of information that are needed for the later integration.

1.2.3 Overview of the WebCat approach

We propose a new approach for information integration that we call WebCat. The main ideas from WebCat are consequences of the difficulties of using the IIS to build wrappers for Web catalogs. Due to the lack of query capability of the information source, we extract the attributes of interest from each record, and we store and query them locally. Two additional factors make this approach possible: the modest numbers of records in each catalog and the unique identifier for each record.

Once the data is stored locally, it can be queried. Consequently a query works as follows: first we retrieve the records from the local database that satisfy the query criteria; then, for each record, we retrieve the associated HTML pages. This approach is a tradeoff between the time required by the integration and a fast query response time.

In order to extract all this information, each HTML page from the collection has to be visited. This is equivalent to the extraction of the structure of the source, or what we call the *catalog schema.*

The WebCat methodology can be divided in two distinct methods: first, a method for wrapper building for Web catalogs, and second, a method of integrating this information, a method that we call Information Integration with Semantics (IISem) and which encompasses the approach stated above. For building wrappers for Web catalogs we use a novel technique, specifically we use the WebOQL query language in the core of our wrappers.

All these considerations suggest the following steps for building a virtual catalog using the WebCat method:

- First, we build the wrappers for each catalog using WebOQL queries.
- Second, we establish the ontology of the domain, which is equivalent with establishing the virtual catalog schema.
- Third, for each new catalog, we extract the catalog schema and we map it into the ontology.
- Fourth, for each new catalog, we extract the attributes of interest, according to the mapping, and store them in the virtual catalog.
- Finally, we build the query interface to the virtual catalog.

The most difficult part is the deduction of the catalog schema, which is part of the wrapper's task. The wrapper not only extracts the data but also deduces the catalog schema. That is, the deduction of the catalog schema is part of the wrapper building

process. The difficulty is due not only to the large number of HTML pages (e.g. more than a thousand in a clothing catalog), but also to the hierarchy in which the files are organized, which is complex and not always straightforward. Also, describing information sources from the Web is not an easy task; until recently, the best known approach was to integrate collections of HTML that have a simplistic structure, usually HTML tables generated from databases (see [AK97-1], [AK97-2], [HG-MC+97], [HKL+98], [LSS96]).

The rest of this thesis is organized as follows: in the next chapter we present several related efforts in the area of IIS and information extraction over the Web. In Chapter 3 we present how we use WebOQL to build our wrappers. In Chapter 4 we introduce the Web catalog concepts and terminology. In Chapter 5, we present the basis of our method. In Chapter 6, we present the architecture of WebCat in detail. In Chapter 7, we present in detail the WebCat wrappers. In the last chapter, we conclude our work, we highlight some of the shortcomings of the design, and we present a design specification for tools needed to overcome the shortcomings.

Chapter 2: Related Work

2.1 Introduction

As mentioned earlier, our goal is to integrate hierarchical collections of HTML pages, in which pages are interrelated by hyperlinks, and the collections have a common domain. This process requires two steps: first, extracting the information of interest (from the Web, in this case), and second, integrating this information.

Several research efforts address specifically or partially our goal. Some of these works address more than our problem; others address partially our problem. In the first category are the IIS that were built to integrate information from various sources. In the second category we include several projects that extract information from the Web. We include these efforts because extracting information from the Web is a crucial part of our system. These projects include agents, learning methods, pattern-matching methods, and languages to query the Web. All these efforts extract information from Web sources for various purposes. We argue that these projects deal only with less complex Web sources, such as HTML forms, or simple HTML pages in which the information of interest is located either in tables or in some sort of “easy to recognize” markup.

Also we present efforts from a relatively new area of research, the Web site management systems. Some of these projects match our requirements and could be used as an alternative for extracting data from Web catalogs.

Next we present these projects, and we compare each of these approaches with our goals in order to emphasize the advantages and drawbacks of each method.

2.2 Extracting Information from the Web using Wrapper Architectures

The wrapper/mediator architecture is the most common architecture for Information Integration Systems (IIS). Systems like TSIMMIS [G-MHI+95], [HG-MN+97], [TSIMMIS], Garlic [TrS97-1], [TrS97-2], [HKWY97], [Garlic], and Disco [KTV97], [TRV95], [Disco] are

built using this paradigm. In these systems, the wrappers encapsulate the data sources and mediate between them and the mediators. Wrappers convert system queries into native queries or commands that are understandable by the underlying sources and transform native results into a format that is understandable by the system application. Usually wrappers include query engines and query plan generators.

Wrappers are hard to build, due to their complexity. Although one approach is to use wrapper templates, generating a new wrapper is not an easy task; and generating wrappers for a large number of sources could be very time consuming.

For wrapping Web sources, query planners, and query optimizers are useless, because most Web sources have no query capability. This observation leads to the idea of building “light” wrappers for the Web. An implementation of this idea is provided by Hammer et. al. [HG-MC+97] which is part of the TSIMMIS project. In this approach, data is extracted from a Web page according to a custom-built *extractor specification file*, which is based on text patterns that identify the beginning and end of relevant data. In the domain they address, the authors extract weather data from HTML tables. The specification file describes the mapping between the data enclosed between HTML markers, and the extraction variables.

The drawback of the method is found in its simplicity. The text patterns, which identify the beginning and the end of data, may not always be available. In addition, real catalogs on the Web have a far more complicated structure than an HTML table, abounding especially in hyperlinks, which are not very well handled, by the extractor specification file.

2.3 Automating Wrapper Generation

Considering that wrappers are hard to build, a tempting idea is to automate the process. In the following, we present two efforts: first, a semi-automated wrapper generator for the Web and second, an induction method for Web wrapper generation. In these approaches wrappers are reduced to the sole function of information extractions.

Knoblock et. al., as part of the Ariadne project present an extraction method for obtaining information related to countries from the *CIA World Fact Book* [AK97-1], [AK97-2], [KMA+98]. Their method is based on the observation that some known words and phrases, and some particular HTML tags identify the section headings, constituting

what they call *tokens of interest*. Another observation is that these tokens of interest are nested in some hierarchies within sections.

Based on this observation, they developed an algorithm that, given a page with all sections and headings correctly identified, outputs a hierarchy of sections. The algorithm is based on the heuristic that font size indentation in HTML pages indicates that one section is a subsection of the other.

The drawback of this method resides in the fact that the algorithm works successfully only when HTML pages are well organized, and when the collection of pages is "flat". That is, the method assumes that there are no hyperlinks to other documents. But, this is not the case for catalogs, which have a complicated hyperlink structure, in which pages are not always "nicely" arranged.

A different approach is provided by Kushmerick et. al. [KG98], [Kus97], [KWD97]. The authors use wrapper induction for generating wrappers. They rely on the basis of learning, by generalizing from examples. In order to generate wrappers, they build an oracle that labels examples, and then, by inductive generalization, they generate a wrapper for the information sources.

The drawback of the method resides in the simplicity of the documents that can be learned by the system. They rely only on head-left-right-tail (HRLT) documents, which are HTML tables with a title and an ending. This is an oversimplified case for our purpose.

2.4 Software Agents

Wrapper-based architecture is not the only alternative for Information Integration. A different approach is to use software agents. Software agents are applications that use software tools and services on a person's behalf. A representative family of software agents is the Softbot (software robots) family. Softbots [Etz96] allow a person to communicate what they want accomplished and then dynamically determine how and where to satisfy the person's request. ShopBot, "domain independent comparison-shopping agent", belongs to this family, and addresses the problem of Information Integration on the Web [DEW97], [PDEW97]. The main idea behind ShopBot is that of querying information sources on the Web with known objects and of analyzing the answers. This technique applies only to sources for which the user, the agent in this

case, retrieves the information according to some forms to fill. Consequently, once the agent is deployed, it identifies the appropriate search form, it determines how to fill out the form, and it discerns the format of the product description in the pages that are returned by the form. ShopBot relies on a combination of heuristic search, pattern matching, and inductive learning techniques.

The drawback of this approach is that it can integrate only sources in which the information is queried through forms. In our case, where the information sources rely heavily on navigational patterns, this approach is inadequate.

2.5 Languages to query the Web

We presented above several systems that partially solve our problem. A desirable system should have capacity to extract information from large and complex Web pages, navigational capabilities, and should allow easy integration of new sources. A category of languages qualifies, at least partially, for the task. In this category are the languages to query the Web. These languages are WebSQL, W3QL, WebLog. All these languages are first generation of languages to query the Web.

The first two, WebSQL [Mih96], [MMM96], [AMM97], [WebSQL], W3QL [KS97-1], [KS97-2], [W3QL] are very similar. Both are capable of retrieving data by querying search engines and combine queries with controlled automated navigation. The use of navigational patterns makes them well suited for our problem. Unfortunately, their navigational pattern matching and their query capability are not powerful enough to describe the structure of complex Web pages.

The third language WebLog [LSS96], [WebLog] accommodates partial knowledge that users may have about the information that is queried. This knowledge is relative to how the information of interest is grouped in the page. This method works well with relatively simple pages with a high level of regularity. This case is an ideal setting compared with our complex HTML pages that we want to integrate.

2.6 Web Site Management Systems

The systems presented so far share one characteristic: they are not well suited to describe Web based information sources that have a complicated structure and that are characterized by multiple hyperlinks. These later requirements lead to a relatively new domain of research: Web-site management systems. These systems are well suited to manipulate large collections of complex HTML pages. In the following, we present two such systems: Araneus and STRUDEL, also we present a related system, FLORID, that is not a Web site management system per se, but has similar characteristics.

All of these systems incorporate query languages meant to query the Web. In order to accommodate the requirements of Web site restructuring these languages have a considerable computational power that goes beyond the first generation of languages to query the Web. This is the reason why these languages are considered to belong to the second generation of languages to query the Web.

Araneus [AMM97-1], [AMM97-2], [MAM+98], [Araneus] is a complex system design for managing and restructuring data coming from the Web. Araneus extracts, analyzes, manipulates, restructures, and integrates Web sites. Araneus works as follows: first a data schema is established using a specific data model. A specialized language, Editor is used to extract the data from the site according to the data schema specification. Then views are built; these views preserve the navigational property of the underlying data. By projection, these views can be flattened into relational views. Later, from these relational views, new Web sites can be built according to new schemata. Thus this whole process is to define derived hypertextual views over the original sites.

At the first glance, Araneus appears well suited for the purpose of integrating Web catalogs. Its drawback comes from Editor, the data extraction language. Editor uses simple patterns for searching documents. These patterns are lists of HTML tags, similar to the *extractor specification file* used in the TSIMMIS projects that we mentioned in the beginning of this chapter.

This technique works well in the case of simple HTML documents, with a regular structure. Web catalogs have a much more complicated structure, and simple pattern searching is not adequate.

A different approach is provided by STRUDEL [FFK+1], [FFK+2], [FFLS], [STRUDEL]. At the core of STRUDEL is STRUQL, a datalog-like query language, which uses a labeled directed graph as data model. In addition, the Strudel data model includes named collections, and supports several atomic types that commonly appear in web pages, such as URLs, and Postscript, text, image and HTML files. STRUQL extracts data, reorganizes it, and creates new graphs from the existing graphs. Also, it is capable of interpreting regular path expressions. In terms of expressive power, STRUQL has the expressive power of stratified linear datalog.

STRUQL is appropriate for the task of extracting and integrating information from complex Web pages. At the time when we started our research, STRUQL was not available. With all its capabilities, STRUQL can be considered an alternative for our problem of extracting information from complex Web-sources.

Also an alternative is FLORID [HKL+98], [HL98], [HLL+97], which is another recently developed system. FLORID is a prototype implementation of the deductive and object-oriented formalism F-logic. FLORID provides a powerful formalism for manipulating semistructured data in the Web context. The FLORID data model consists of two classes, url of all URLs and webdoc, the class of actual Web documents. Using deductive rules FLORID is capable of extracting information from complex Web sources, and thus is suitable for our problem.

STRUDEL and FLORID can be used as an alternative for solving our problem. They are recent developments, and were not available at the time when we started the WebCat research.

2.7 Commercial products

There are several commercial products that are in the same area as WebCat. Due to the lack of public information on these commercial products, we just present a list of them with a short description of each.

The first one, Jango [www.jango.com], a partner of the Excite search engine, offers a large selection of products from computers to flowers. BidFind [www.vsn.net/af] offers an interface to the on-line stores that specialize in computer parts. The last one, Netbuyer by Junglee [www.junglee.com] is a Yahoo site and offers a wide range of

products that are typical for a departmental store. Netbuyer [www.shopguide.yahoo.com] is the most recent and complex of the presented systems. It is very similar in capabilities to WebCat.

Chapter 3: Building Wrappers using WebOQL

3.1 Motivation

As mentioned earlier, in order to achieve our goal of integrating Web catalogs, we have to follow two steps: first, extract the information of interest from catalogs and second, integrate this information. The information extraction step can be divided into extracting the schema of a new catalog, and extracting the attributes of interest from this catalog.

There are several approaches to information extraction from the Web. As we argued in Chapter 1 and exemplified in Chapter 2, most of these approaches are inadequate for Web catalogs. Related to this inadequacy, we can group these approaches as being:

- too complex. This is the case of IIS wrappers. As presented in Chapter 1, and exemplified by the TSIMMIS project in Chapter 2, the IIS wrappers are complex and hard to build. In addition they rely on some features, such as query capability, which are not present in Web catalogs.
- not powerful enough. This is the case of agents, learning methods, and pattern matching. All these approaches are capable of extracting information from HTML pages with a simple structure, far from the complexity encountered in Web Catalogs.
- not appropriate. This is the case of web site management systems such as Araneus and the first generation of languages to query the Web. The inadequacy of these systems resides either in the extraction technique (simple patterns in the case of Araneus) or in the lack of the expressive power (the case of the first generation of languages to query the Web).

Considering the enumerated requirements and taking into consideration all these inadequacies presented above, we searched for a language capable of extracting the catalog schema and also with enough expressive power to extract attributes matching predicates following navigational patterns. The perfect match for these requirements is WebOQL [Aro97], [GM98], [WebOQL].

WebOQL is capable of extracting information from complex HTML files using predicate matching and navigational patterns. Furthermore it is capable of extracting the Abstract Syntax Tree of HTML documents, a characteristic that is crucial for extracting the schema of Web catalogs.

Next, we introduce briefly some of the characteristics of WebOQL, especially those characteristics that we use to build our wrappers.

3.2 WebOQL

3.2.1 Overview

WebOQL is a complex web-based query system developed at University of Toronto. WebOQL is based on a "middleware" approach to data integration: it uses a simple and flexible schema-less data model into which many data structures can be logically mapped, and a powerful query language that provides facilities for handling data with unknown or irregular structure.

The two main elements of the data model are *Hypertrees* and *Webs*. Hypertrees are ordered edge-labeled trees with two types of arcs, internal and external. Internal arcs are used to represent structured objects, while external arcs are used to represent references (typically hyperlinks) among objects. Hypertrees are very useful data structures because they subsume the basic abstractions: collections, nestings, orderings and references. Webs are data structures found at a higher level of abstraction than trees. A Web is a set of trees, each having an associated URL. A Web can be used to model a small set of related pages (for example a manual), a larger set of pages (for example, all pages in a corporate intranet), or the whole WWW.

The WebOQL query language is a functional, OQL-like query language [Cat96] in which both trees and Webs can be manipulated and created. It exploits ordering and provides facilities for searching graphs and trees. The goal of the query language is to enable the navigation, querying and restructuring of Webs. The query language is purely functional; queries can be nested arbitrarily, like in OQL. WebOQL has a formal semantics, and the expressive power of the language is bounded to express feasible queries, i.e., queries of polynomial complexity. More precisely, WebOQL can simulate all

operations in nested relational algebra and can compute transitive closures on arbitrary binary relations. In the following, we present the WebOQL query language by means of examples. The presentation is not exhaustive; we present only the main operators in order to provide basic understanding of what we are going to use in this thesis.

3.2.2 Operators

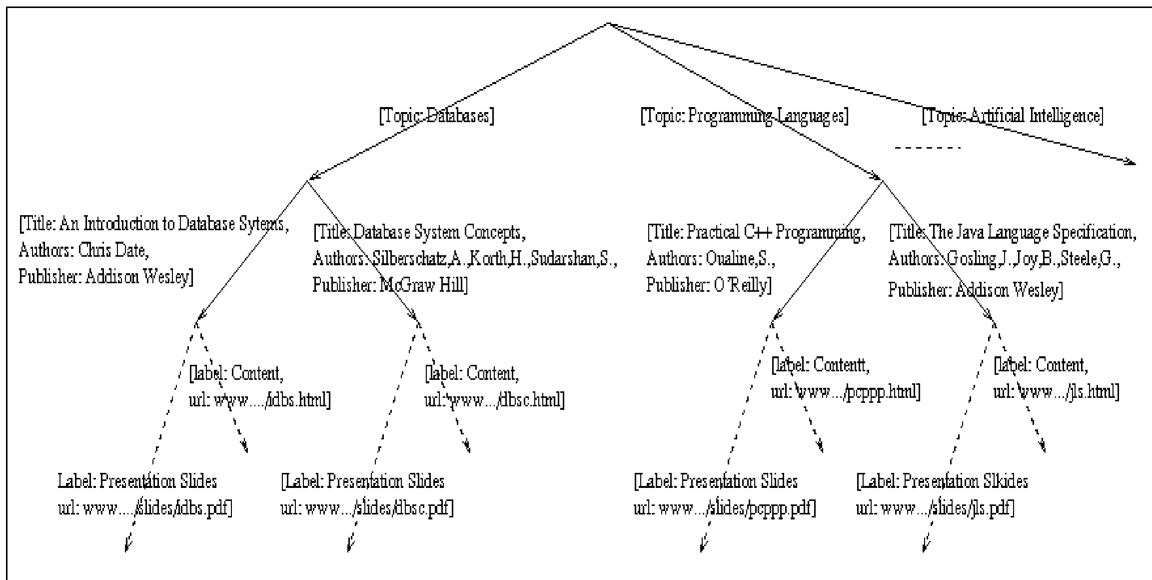

Figure 1: A Computer Science textbook database

For our presentation, we use a sample hypertree that reflects a collection of HTML pages that refer to computer science textbooks (see Figure 1). In the directed tree from Figure 1, each arc has a record associated with it. The associated record has different attributes. These attributes can be “topic” for the deep first arcs, and can be “title”, ”authors”, “publisher” for arcs of depth two, etc. The solid lines in Figure 1 represent internal arcs, dotted lines represent external arcs. External arcs cannot have descendants and the records that label them must have a field named Url.

The query language is built around the familiar *select-from-where (sfw)* paradigm. WebOQL has two groups of operators; one for building arbitrary trees and one for breaking trees into pieces. In the first group, we have the hang and concatenate operators. In the second, we have prime, peek, head and tail operators. The notation for

prime and peek is “`” *(quote)* and “.” *(dot)* , for head and tail the notation is “&” *(ampersand)* and “!” *(exclamation mark),* respectively.

We present next an example. Suppose that csTextBooks denotes the Computer Science (CS) textbooks database that was sketched in Figure 1. The first query q1, extracts the title and the URL of the database textbooks.

Query q1: select [y.title, y’.Url]
from x in csTextBooks, y in x
where y.Title ~ “Database”

Query q1 works as follows: x iterates over the simple sub-trees of csTextBooks (i.e., over the Computer Science topics) and, given a value for x, y iterates over the simple sub-trees of the only sub-tree of x (i.e., over the textbooks of the CS topic represented by x). The operators used in the query are:

*The “’” *(quote)* represents the symbol for the *prime postfix operator*, which returns the first sub-tree of its argument.
*The “.” *(dot)* represents the *peek operator,* which extracts a field from the first outgoing arc of its argument.
*The “[]” *(square brackets)* represent the *hang operator*; which builds an external arc; in general, it builds a simple tree.
*The “~” *(tilde)* represents *the string pattern-matching predicate*: it takes as left argument a string and as right argument a grep string pattern.

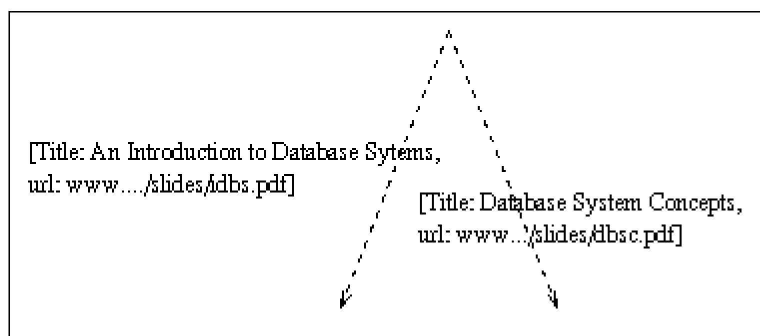

Figure 2: Output of Query q1

Another operator is concatenate. Concatenate juxtaposes two trees. An example of concatenate and hang is Query q2 where we create a tree purely from constants. The output is shown in Figure 3.

Query q2: [Tag "UL", Text "Root"/
[Tag "LI", Text "First"]+
[Tag "LI", Text "Second"]+
[Tag "LI", Text "Third"]
]+
[Url "http: // address" Label "External Link"]

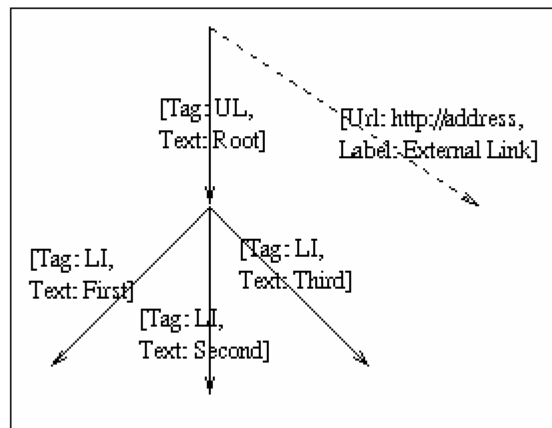

Figure 3: Output of Query q2

In queries q3 to q5 of Figure 4, we show how prime, head and tail operators work. In Query q6, we use an extended version of the head operator, which allows us to get the first n simple trees of a tree, for a nonnegative integer n. The tail operator has a similar extended version.

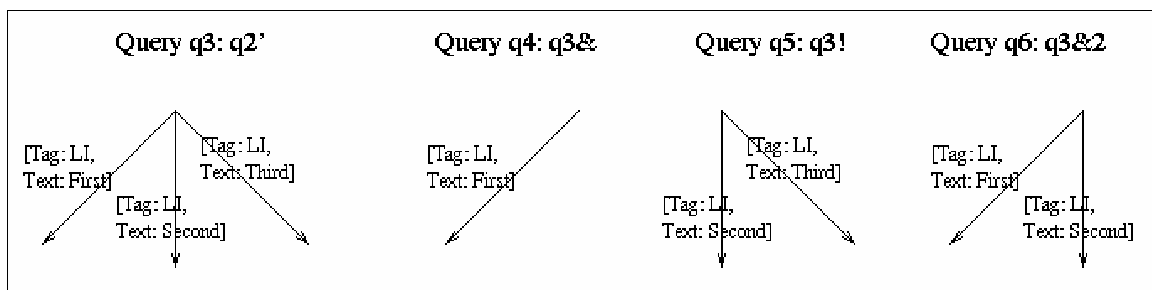

Figure 4: Query q3, q4, q5, q6 and their output

3.2.3 Navigational Patterns

In the queries presented above we used only simple operators to extract sub-trees. WebOQL is much more powerful than this; it allows one to use navigation patterns (NPs), which are regular expressions over an alphabet of record predicates. These patterns allow one to specify the structure of the paths that must be followed in order to find instances for variables. These patterns are basically regular expresions.

For example, the NP `[not(Tag = "TD")]*' specifies paths of any length composed of arcs not having an attribute Tag with value "TD". Similarly, the NP `[Tag = "TR"][Tag = "TD"]' specifies paths of length two with the obvious meaning.

Two predicates are used quite frequently in queries: `[?Url]' and `[not(?Url)]'. These predicates test whether an arc is external or not; they can be abbreviated by the symbols “>” and “^”, respectively. Thus, for example, the combination `^*>' specifies all paths in a tree that lead from the root to an external arc.

NPs are used mainly for two purposes:

- First, for extracting sub-trees from trees whose structure we do not know in detail, or whose structure contains irregularities (for example, extracting from an HTML document all the anchors or all the headers containing some keyword).
- Second, for iterating over the members of collections of trees connected by external arcs. The next examples illustrate both uses. Query q7 retrieves the URLs of all the external arcs in the document pointed to by "http://document.html" that do not occur within a table.

```
Query q7: select [ x.Url ] from
              x in "http://document.html" via [not(Tag = "Table")]*>
```

In order to answer q7, WebOQL matches paths starting at the root of the source tree. For each matching path p, the variable x is instantiated to the simple tree starting at p's last arc. Variables are instantiated following the order in which paths are matched during a left-to-right depth-first search.

An important property of NPs is that they allow one to traverse external arcs. In fact, the distinction between internal and external arcs in hypertrees becomes crucial when we use navigation patterns that traverse external arcs. Suppose that we have a product catalog in which each product points to the next one by a hyperlink with the "Next Product" label. If we want to extract all products we could use Query q8.

```
Query q8: select [ x.Url, x.Text ]from
        x in "http://catalog.html" via (^*[Label ~ "Next Product"]>)*
```

With the presented basic operators and using NP we are able to write queries that can be used to extract the structure of complex Web catalogs. In Chapter 7, where we present in detail how wrappers are built, we present also the "real" WebOQL queries used to build the WebCat wrappers.

Chapter 4: Web Catalogs

Having established the method to extract information from Web sources, we have to delimit our information sources of interest. As mentioned, we are interested in large collections of HTML pages interrelated via hyperlinks, and arranged according to some kind of hierarchy. We call these sources Web catalogs. In the following, we present an example of Web catalog, followed by the data model, then a formal definition for the Web catalogs followed by a classification.

4.1 Web Catalogs, an example

A large number of data collections that are available on the Internet are found in what we call *Web catalogs*. By Web catalog, we mean a collection of interrelated HTML pages connected by hyperlinks according to a logical hierarchy. Therefore, a Web catalog has an entry point from which the whole hierarchy can be explored. We call this entry point the *catalog home page*. A Web catalog can be a subpart of a larger Web catalog. With this basic knowledge about Web catalogs, we present now some examples.

An example of a Web catalog is the collection of HTML pages related to the activity of a university, such as University of Toronto (UofT). In this example, we have a unique entry point: the UofT homepage [www.toronto.edu]. From this home page, we can reach all the UofT resources using some navigational paths. We call this collection of pages, from the UofT domain, a Web catalog. Given the UofT catalog, if we are interested only in the collection of the individual home pages of the persons who are affiliated with UofT, from the UofT homepage we choose only the "departments" option. Then, we follow the links to each departmental home page, and to different lists which contain links to the individual homepages. For the Department of Computer Science the departmental home page is [www.cs.toronto.edu]; to get the homepages of the faculty, we have to chose "DCS Faculty Members" [www.cs.toronto.edu/DCS/People/Faculty], and for the students: "Other DCS Members" [www.cs.toronto.edu/DCS/People/PersonalPages]. If we repeat this procedure on each departmental homepage, we get all

the homepages from UofT. We also call this latter collection of pages (from the CS department) a Web catalog. This new Web catalog is evidently part of the much larger UofT Web catalog.

The former simple example suggests that we can view as Web catalogs the sites of any organization. Furthermore, several other categories of sites qualify for the Web catalog denomination. Some simple examples includes news agencies sites, newspaper sites, library catalogs on the Web, product catalogs on the Web, etc.

The Web catalog concept helps us to achieve our initial goal of integrating information sources from the Web. Once we delimit the Web catalogs, it is easy to group them into domains. Using the former examples, possible domains are universities, news agencies, newspapers, library catalogs, and product catalogs for a certain group of products. The advantage of grouping the information into domains is the common structure of the Web catalogs. For the university domain, e.g., it is highly probable that most university sites are organized similarly with the UofT site. Thus a structure like: university homepage->departments->faculty, people->individual homepages, should be common. Since the structure is nothing more than the catalog's internal hierarchy, we can assume that most Web catalogs in the university domain have an approximately similar structure.

4.2 Data Model

At the core of the Web catalog concept is its data model: WebCat Data Model (WDM). In this data model, the information is organized as a directed tree. That is, a directed graph, with no cycles, with exactly one vertex of zero in-degree, called the *root* of the tree and all other vertices have in-degree equal to 1. Arcs are labeled with atomic values. The labels for arcs that stem from the same vertex have distinct labels. Vertices are labeled, not with atomic values, but with records. Each record has a unique ID that is built using a specific rule. We present the WDM in Figure 5.

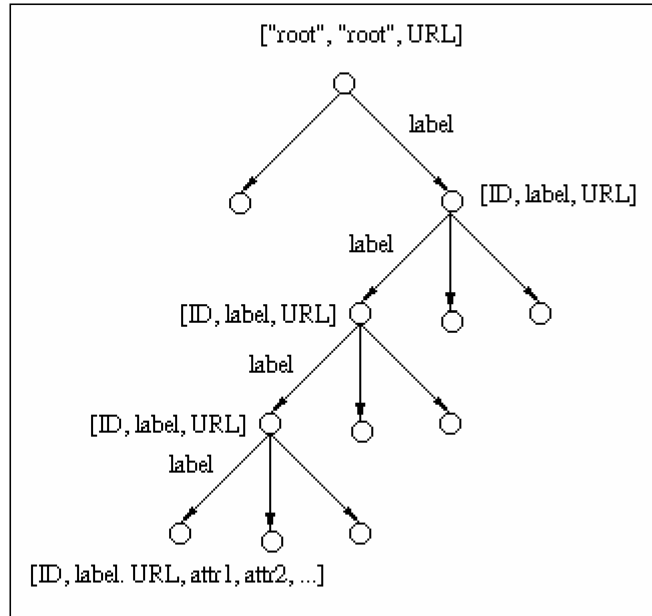

Figure 5: WebCat data model

4.2.1 Vertices and Arcs

Vertices in WDM are classified as root, branches and leaves. A *leaf* is a vertex with zero-out degree. A *branch* is a vertex that is not root or leaf.

The records associated with labels have different schema for different vertex categories. The record schema associated with branches is:

[ID, label, URL].

The root has associated a similar record:

[“root”,”root”,URL]

For the leaves the associated record schema is:

[ID, label, URL, attr1, attr2, …].

Arcs are labeled with atomic values that are instances of string. Moreover the value of the arc's label is the same as the value of the label attribute from the record associated with the vertex to which the arc points.

Vertices are situated on different levels. We say that a set of vertices belong the same *level* if they are situated at the same distance from the root. By distance we mean the number of vertices that have to be traversed in order to reach the vertex from the root.

4.2.2 Building Record IDs

Because our model is a directed tree, the path from root to any vertex is unique. We encapsulate this property in our data model. The record ID is a string that reflects the path from the root to the vertex. We build the ID as follows:

- the root has the ID "root",
- a vertex, other than the root, has the ID: ID of the parent + "->" + the label of the edge from parent to vertex

We exemplify this rule by the following example. Suppose that we have a vertex with the root as parent, and the edge that connects the root to the vertex has the label "label1". Using the rule above, the vertex has the ID: "root->label1". Furthermore suppose that this vertex is parent for another vertex, and the edge between these vertices has the label "label2". Using the rule, the latter vertex has the ID: "root->label1->label2". The ID has the meaning: in order to get to this vertex, we have to start from the root, follow an edge labeled "label1", and next follow another edge labeled "label2".

4.2.3 An Instance of the Data Model

We present next an instance of the WebCat data model, the data schema of Eddie-Bauer (EB) clothing catalog (Figure 6). In this schema each vertex is associated with an

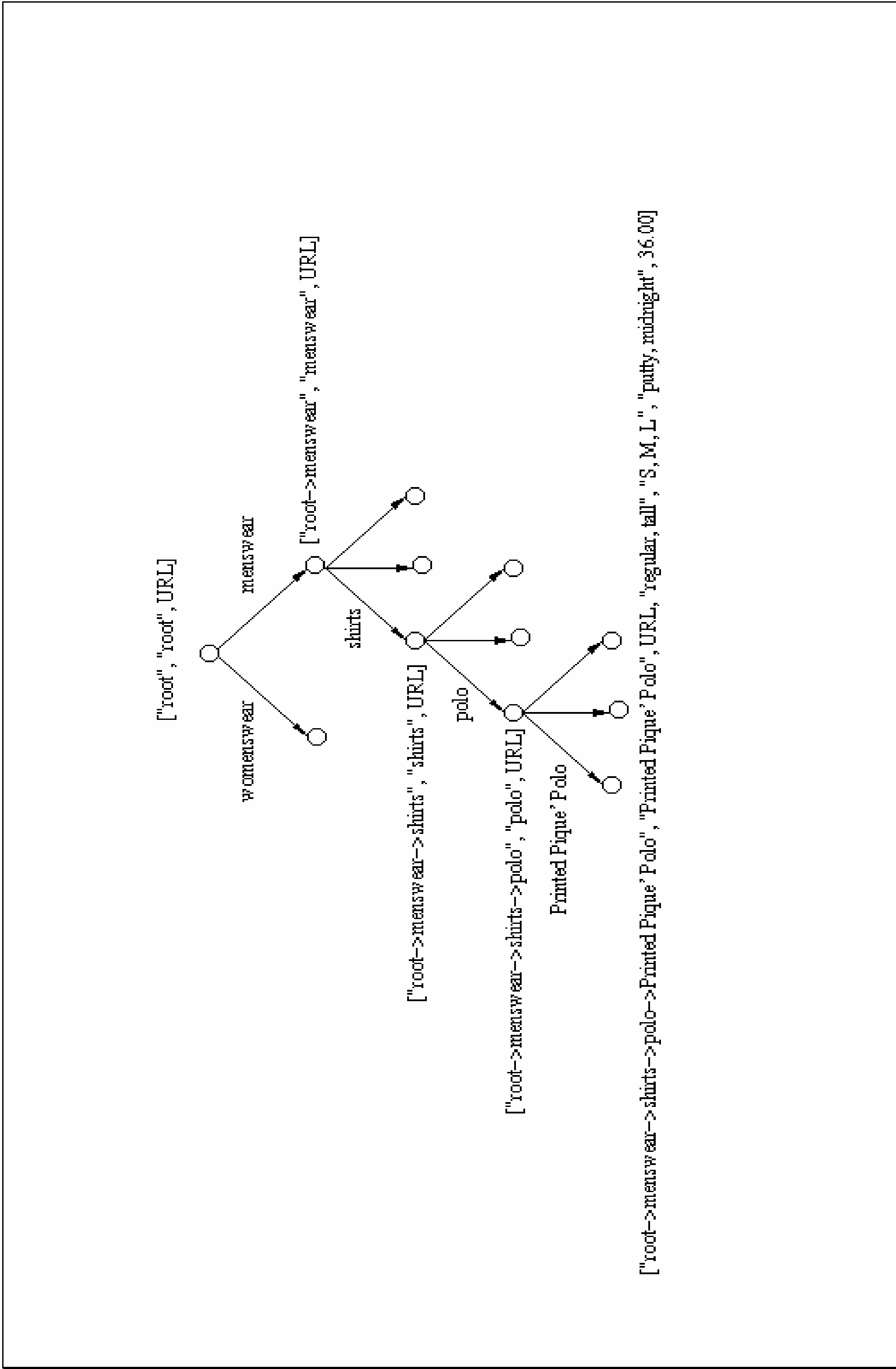

Figure 6: Instance of WebCat data model, the data schema of Eddie-Bauer catalog

HTML file. The root vertex is the home page of the EB catalog, each branch represents a classification criterion, and a leaf is a product.

In the case of the EB catalog, a level is nothing more than a classification level. Thus, “root->menswear” and “root->womenswear” are on the same level (level 1), and represents the first classification criterion for the EB catalog, which is divided in two category menswear and womenswear.

Each leaf represents an HTML page, the page that is generated for a particular product. This is not the case with branches. In the EB catalog some levels are nested on the same HTML page. Thus, “root->menswear->shirts”, which is a level 2 branch, is located in the same HTML page with “root->menswear->shirts->polo”, which is a level 3 branch.

An arc between two vertices represents an HTML link between the two vertices and the label associated with the arc is the text from the HTML document associated with this link. As an example, an arc between “root” and “root->menswear” implies that from the EB homepage there exists a link to the menswear page, a link that has the “menswear” text associated.

The record ID “root->menswear->shirts->polo” has the meaning that the polos can be found by navigating from the root to menswear, from menswear to shirts and from shirts to polos.

The records are more complicated if we get to the leaves. In addition to the “standard” attributes (ID, label, URL), we have several attributes that describe the product. These attributes, in this case, are size type, size, color and price. For the product from Figure 6 (“Printed Pique’ Polo”), these attributes have the values: “regular, tall”, “S,M,L”, “putty, midnight” and 36.00. All these attributes are needed for querying.

4.3 Mapping Real Catalogs to Web Catalog

Earlier in the thesis, we defined Web catalogs as a collection of interrelated HTML pages that are ordered by some hierarchy. Once we have defined the WebCat data model, we are able to describe better what a Web catalog is. Thus, a web catalog is a collection of HTML pages that can accommodate a WebCat data model. By accommodate we mean that starting with the initial collection of HTML pages, using a small number of transformations, we can obtain a new collection which respects the

WebCat data model. Usually these transformations are a "stripping" process. By "stripping" process, we mean to the following: usually the information sources on the Web are for commercial use; consequently, the HTML pages contain not only the information of interest, but also additional information such as advertising. Furthermore, to ease the user's navigation, in each HTML page, there are a multitude of backward links to the levels above. All this additional information can be easily stripped out. Very often this additional information is concentrated in a frame or in Java scripts that are both easy to locate and eliminate.

Once we define the Web catalog, we have to define some informal terms. We call *schema* or *structure*, the hierarchy according to which the catalog is organized. That is, the catalog schema is nothing more than the directed tree from the data models. We use the terms *root*, *leaves*, *branches and levels* in the way defined in the data model. Therefore, a Web catalog has a schema and is composed of a root, branches, leaves and has levels. The root is a unique entry point from which any of the branches or leaves can be reached. The branches are nothing more a classification criterion (similar to classes in the object-oriented paradigm) and the leaves represents the information of interest.

In the earlier example with the UofT people, the root is the UofT homepage, a branch is the Computer Science Department homepage, and a leaf is my homepage [www.cs.toronto.edu/~atibarta]. A label is my name associated with the link to my homepage.

4.4 Classification of Web catalogs

Our Web Catalog concept was influenced by the product catalogs that are available on the Web. These product catalogs represent an electronic form of the common printed catalogs. Consequently, Web product catalogs are collections of HTML pages interconnected via hyperlinks, and ordered according to a classification which is common to their domain.

These catalogs are truly navigational sources. That is, the only way to locate a product is to follow some navigational path, starting with the root. In order to follow the navigational path via the hyperlinks, one condition has to be satisfied: the hyperlink labels should be meaningful to us. That is, labels should have distinctive names, and these names should belong to a common domain.

In the product catalogs, this condition is built in. If we consider the example from Chapter 1, where one searches for gloves, in a product catalog we have some leaves with labels including the word “gloves”. Some possible labels could be “leather gloves”, “polar gloves”, etc.

Furthermore, since the labels are nothing more than classification criteria, these labels are uniform across catalogs from the same domain. This observation allows us to integrate in an automated fashion the product catalogs, once we establish a common domain.

According to the method of integration, we classify Web catalogs into automatically integrable and manually integrable. A Web catalog is *automatically integrable* if the label attributes from both leaves and branches are meaningful. That is, the labels have distinctive “names” which belong to a common domain, allowing an automatic integration. If the labels are not meaningful we call the Web catalog *manually integrable.* The manually integrable catalogs are not uncommon. One possible reason is that the hyperlinks are not implemented using the <A> HTML tag, rather are implemented using pictures. Consequently, there is no automated way to “guess” the classification hierarchy.

Web catalogs also can be classified into structured and semi-structured. This classification criterion is inspired from Kolmogorov complexity [Lee90]. We call a Web catalog *structured,* if we have a finite set of root-leaf path patterns such that any root-leaf path is an instance of one of these patterns, and the size of the pattern set is considerably smaller than the size of the set of all root-leaf patterns from the catalog. Consequently, a Web catalog is *semi-structured* if the opposite holds.

In another formulation, the definition above states that a Web catalog is structured if it possesses a regularity of its structure from root to leaf. That is, if we add a new leaf, the new root-leaf path will respect an existing pattern. If this is the case we are able to write a query to instantiate over a certain sub-tree group. If this regularity does not hold, we have to write distinct queries to extract any leaf.

An example of semi-structured catalog is a typical university catalog. In order to extract the home pages we have to write distinct queries for each department. On the other hand, product catalogs are structured catalogs. They have a very regular structure; consequently it is enough to write one or two queries for each level.

This classification criterion allows us to divide all possible web sources into two types: semi-structured, that could be any web-source, and structured, that are our sources of interest.

In the next table we present some real Web catalogs in order to exemplify each classification criterion.

	structured	**semi-structured**
automatically integrable	Eddie-Bauer clothing catalog (www.eddiebauer.com/eb/ShopEB/frame_line.asp)	The collection of departmental pages from University of Toronto (www.toronto.edu)
manually integrable	Bloomingdale clothing catalog (www.bloomingdales.com/bloom/shopping)	Maxsol home page (www.maxsol.com/Main.htm)

The Eddie-Bauer clothing catalog, is an example of structured, automatically integrable Web catalog. We use this catalog as an example through this thesis.

The Bloomingdale clothing catalog is a structured, manually integrable Web catalog. It is structured because is a product catalog, and manually integrable because at the first level they use pictures for the first level of classification, thus it is impossible to recognize with an automated method the meaning of the pictures, unless we use sophisticated pattern recognition techniques.

The collection of departmental pages from UofT is an example of semi-structured, automatically integrable Web catalog. It is automatically integrable because the links bear meaningful labels, and it is unstructured because a distinct WebOQL query has to be written for each department. Furthermore, if a new department is added we have to write a totally new WebOQL query because is very likely that the new departmental home page will be different from any other departmental home page.

Finally, the home page of Maxsol, a company specialized in extranets, is a good example of an unstructured, manually integrable Web catalog. It is unstructured because it is a company's home page, and it is only manually integrable with an existing ontology because the selection menu is implemented using pictures.

Chapter 5: The WebCat Method

The goal of the WebCat method is to integrate Web catalogs. As we mentioned earlier, WebCat belongs to the broader area of IIS, although it has some significant differences from the existing IIS. In order to present the WebCat method, we continue to use the existing IIS as a comparison. We mentioned earlier (in Chapter 3) that one of the differences resides in the WebCat wrappers. Two other differences are the location of the information and the information integration technique. We present next these differences followed by the description of the WebCat method, and ending the chapter we present an example of the WebCat method.

5.1 Location of the Information

The IIS keep data in the initial repositories and extract them only on request. The IIS keep only the description of the information source and the information about the query capability of the sources.

Because the Web catalogs have no query capabilities, the IIS approach is not appropriate. To overcome this problem we store not only the description of the source (the catalog schema) but also we store attributes of interest that we later use for answering queries. That is, we store locally a description of each piece of information from the source (HTML pages in this case). By description we mean a set of attributes of interest and the URL of the page. Thus instead of querying directly the information sources, we query a local repository and then retrieve only those HTML pages that satisfy the query criteria. We call this local repository the *virtual catalog*. We define the virtual catalog in the next sub-chapter.

The presented approach situates our method between the IIS and data warehouse because on one hand, we retrieve on request the HTML pages, and on the other hand, we consolidate local repositories which can be later queried.

5.2 Information Integration with Semantics

The IIS integrates several sources that have different schema. In order to achieve this, the IIS use the concept of general schema. That is, the general schema integrates and maps individual schemata to a schema super-set called a general schema. In the case of IIS, this is a complex process and is done manually.

In the case of integrating Web catalogs since we are interested in integrating only catalogs from a specific domain, the integration issue is not as complicated as it is in the case of IIS, thus it can be done in a semi-automatic mode. Because we are using the knowledge of the domain to which the catalogs belong, we call our semi-automatic method: Information Integration with Semantics (IISem).

The first step in IISem is to determine an ontology of the domain to which the Web catalogs belong. By ontology we mean a controlled vocabulary in which the terms are grouped into classes and the classes are arranged into class hierarchies. That is, in our approach an ontology is a taxonomic hierarchy of classes.

Our definition of ontology is less formal than others given in different works. For example a more formal definition is given by Thomas Gruber [Gru93] where an ontology is a statement of logical theory. Another definition is presented by Fritz Lehmann [Leh96] where an ontology is an elaborate conceptual schema of generic domain concepts and relations in the form of a semantic network with constraints or axioms, and with a formal lexicon or concept dictionary.

According to the classification of ontologies made by Arthur Keller [Kel95], [Kel97], [KG96] where the ontologies are grouped in base ontologies, domain ontologies, product ontologies and translation ontologies, our ontologies correspond to the domain ontologies.

In the case in which we do not have an ontology available we build one using the schemata of the catalogs that we know.

Because our ontology is a hierarchy of classes, and can be represented as a directed tree, we use the same notation for the elements of the ontology as we use for the WebCat data model. That is, the representation of the ontology has vertices, edges, and levels. The only difference is in the records attached to each vertex. Thus the associated record for the ontology has only two attributes: ID and label, and the label attribute is a multi-valued one, thus we can accommodate synonyms that are different names for the same class.

We mentioned earlier the term “virtual catalog”. By “virtual catalog” we mean a catalog that has as schema the given ontology and has as leaves descriptions of the HTML pages from the catalogs that have being integrated so far.

The information integration process consists in mapping each catalog to the ontology that is equivalent to populating the virtual catalog. The process of populating works as follows: first we extract the leaves from the catalog to be integrated and second we “attach” the leaves into the virtual catalog according to the mapping.

Before presenting the mapping algorithm, we have to present in detail how we build our ontologies.

5.2.1 Building the ontology

As mentioned earlier, we defined our methodology with product catalogs in mind. The product catalogs have a particularity: some of the classes do not have a unique name. For example, in the case of clothing catalogs two classes with the name “outerwear” exist. One class is a child of the menswear class and the other is a child of the womenswear. That is, we have some sub-tree in which each class has a unique name, but the class names are not unique in the whole tree. For example, in the case of the clothing catalog the sub-trees that stem from menswear or womenswear have unique class names. These vertices, from which stem sub-trees with unique class names, have important roles in the integration process. We call these vertices *pseudo-roots.* Details of the ontology for clothing catalogs are presented in Appendix 1.

The level of nesting of pseudo-roots gives us the *level of undecidability*. The meaning of level of undecidability is that we need the same number of pseudo-roots as the level of undecidability in the leaf’s ID in order to be able to attach the leaf to a proper place. In the example above, if we have a leaf ID with the “outerwear” key word but without “menswear” or “womenswear” also present, we cannot decide where to attach the leaf, consequently an error should be generated. Thus if the level of undecidability is, for example, 3 we should have 3 pseudo-roots present in the leaf’s ID. If this is not the case, we cannot integrate the leaf.

We have to mention here a special class of vertices that are labeled “all others”. Each vertex, except those on the last level, has a child vertex labeled “all others”. We

use this special vertex to accommodate missing classification terms from the ontology. We explain later how this process works.

5.2.2 Mapping a Web catalog to an ontology

Once we have the background on our ontologies, we are able to present how our algorithm works. The basic idea is that because we have encoded in each leaf's ID the leaf's position in the tree (see the data model) it is easy to parse and interpret this codification and to compute the leaf's position in the virtual catalog. The steps that we have to perform in order to integrate a leaf are:

- First, we parse the leaf's ID in tokens.
- Second, we count the number of pseudo-roots and we check that it is equal to the level of undecidability. If not, we generate a report and we quit.
- Third, we use the algorithm from Figure 7 to compute the leaf's new position.

The algorithm works as follows: we take the first token from the leaf's ID. Our search space is the whole tree. We start the search from the root. We search through the children of the root. If we find a child vertex labeled with the term that we are looking for, we are done for this step. We restrict our search space to the sub-tree that stems from the vertex that is labeled with the term that we were looking for. We move to the next term from the leaf's ID and we start the process again.

If we are unable to find such a child vertex, we descend one level and make the current vertex the first child vertex. Next we search through the children of the current vertex and we proceed until we rich the last level or we find the term that we are looking for. We continue this process recursively.

In order to identify the vertices that we already explored, we color them. Thus once we reach the last level and the term is not between these vertices, we color them and we color their common parent too. If in the end all vertices are colored that means that the term is not found.

If the term is not found, we attach the leaf to the current root's “all others” vertex. In this way, once the integration process is terminated, we are able to check if it went

well by checking all the "all others" vertices. If a leaf is found in such a vertex, it means that one of its terms was not found in our ontology.

Although we search the whole tree, because the ontology trees that we use are small, this process is not time consuming. This is the reason why we do not use any indexing technique for searching the tree.

```
INPUT:     ontology: Tree
           leaf ID: List of String
OUTPUT: node of the ontology where to attach the leaf: Node

Search(leaf ID, root of ontology)

Procedure Search(list of terms, root)
        if (list = empty) then done
        else if first term ϵ tree at node v
                then search (rest of list, v)
                else attach rest of list to v.All_others
end
```

Figure 7: Algorithm to compute the leaf's new position

5.3 The WebCat Method

The WebCat method works as follows:

- First, we establish the ontology of the domain.
- Second, we choose the attributes of interest according with the query needs. We choose the attributes from the set of all possible attributes that are associated with a leaf entity. Here, by query needs, we mean the set of queries that we anticipate could be of interest to a user.
- Third, we build wrappers for each catalog.
- Fourth, we extract, for each catalog, the catalog schema and the attributes of interest.
- Fifth, we integrate the extracted information into the virtual catalog using the IISem technique.
- Finally, we build the query interface for the catalog.

In the earlier example of building a virtual catalog for all the UofT personal homepages, the steps of the integration are:

- First, we establish the generalized schema, which is: homepage of university -> departments -> faculty/people -> individual homepages;
- We are interested in providing querying capabilities on person names, departments and universities. Therefore the attributes that we extract are name, department, and university.
- We build wrappers for each department.
- We extract for each home page: URL, name of the owner, university and department.
- We add this information to our repository in order to build the virtual catalog.

The queries that our system can answer are:

- "Retrieve the home page of X"
- "Retrieve the home pages of all students from the department Y"
- "Retrieve the home pages of all students from the university Z"
- "Retrieve the home pages of all departments from the university Z"
- Or any logical combination of the above.

As mentioned earlier, the university catalog is a semi-structured Web catalog. Consequently, we expect it to be difficult to build a large number of individual wrappers. It turns out that it is not the case in this particular integration. Later in this thesis, we present how the wrappers are built. For this particular example, a wrapper for a department is no more than a line of WebOQL code. We stress that this simplicity is due to the simplicity of the catalog, and it is not a general property for the semi-structured catalogs.

Chapter 6: The WebCat Architecture

In the previous chapter we presented the WebCat method for integrating Web catalogs. We implemented this method in a system that we also call WebCat. WetCat is more a methodology than a system because it has to be implemented for each particular domain. We present later in this chapter an implementation of WebCat for the domain of clothing catalogs. We call this implementation *Apparel WebCat*.

Before presenting the WebCat architecture, we discuss first the nature of the catalogs that we integrate, which we refer to as "real world" catalogs as opposed to simple catalogs, such as the university catalogs that we used previously in our example.

6.1 "Real World" Web Catalogs

"Real world" catalogs are far more complex than a university catalog. As mentioned, our work was inspired by the problem of integrating product catalogs. Once we succeeded with the product catalogs, we learned that our method has applicability beyond product catalogs, because many of the information sources on the Web can be modeled as Web catalogs. For a better understanding of our system, we introduce next a category of "real" Web catalogs, the product catalogs.

Product catalogs have a long tradition. They started first on paper, then they moved on-line, and now they can be found on the Web. Product catalogs are perfect Web catalogs. They have a strict hierarchy, which is based on the category of products, and contain one entry point, which is the home page of the catalog. Furthermore, they exhibit a symmetric structure, because they are generated in an automated fashion. By symmetric we mean that the catalog has only a small number of distinctive paths from root to leaves, consequently once we identify these paths we can extract the catalog schema. The product catalogs cover a large area of product categories. Among them are electronic components, books, CDs, electronic appliances, computers, computer parts, software, and clothing collections.

We focused our interest on a representative category of product catalogs, the clothing catalogs. We present the particularities of these catalogs later in the chapter when we present our implementation, Apparel WebCat.

6.2 The WebCat Architecture

WebCat is capable of integrating in a semi-automated fashion structured integrable Web-catalogs from a certain domain. The system architecture is presented in Figure 8.

WebCat has two groups of components, one group is related to the integration process itself; the other group is dedicated to handle user queries. We call these groups Integrator Components (IC) and User Components (UC), respectively. IC is intended for the use of the person who performs the integration and UC is intended for the general user. These two groups share a common component that is the virtual catalog.

Next we present briefly the functionality of each component.

We start the presentation of the architecture with the IC group. As depicted in the figure, on the bottom are the Web catalogs, and for each of them a wrapper is built. These wrappers extract and export the catalog schema that they wrap, augmented with the attributes of interest. All this data is sent to the System in description files, which are files respecting a special format, that we call File Description Format. Thus, the information from a particular catalog is stored in an individual description file.

StruViz is a visualization component. It takes as input a description file, and as output displays, in a graphical format, the catalog schema of the catalog which is stored in the description file. StruViz is used for control purpose only, to assure that the wrappers are working properly.

The ontology depicted in the figure is the ontology that we use in the integration process, as mentioned in the previous chapter. Also the ontology represents the virtual catalog schema. If we do not have a priori an ontology, the Integrator will build one during the integration process. The ontology is stored in a knowledge base (KB).

The main goal of the Integrator is to build the virtual catalog. Thus, the Integrator integrates each individual description file into the virtual catalog using the knowledge base. The integrator takes as an input a description file and as output updates the virtual catalog. The Integrator also updates the knowledge base to accommodate missing terms.

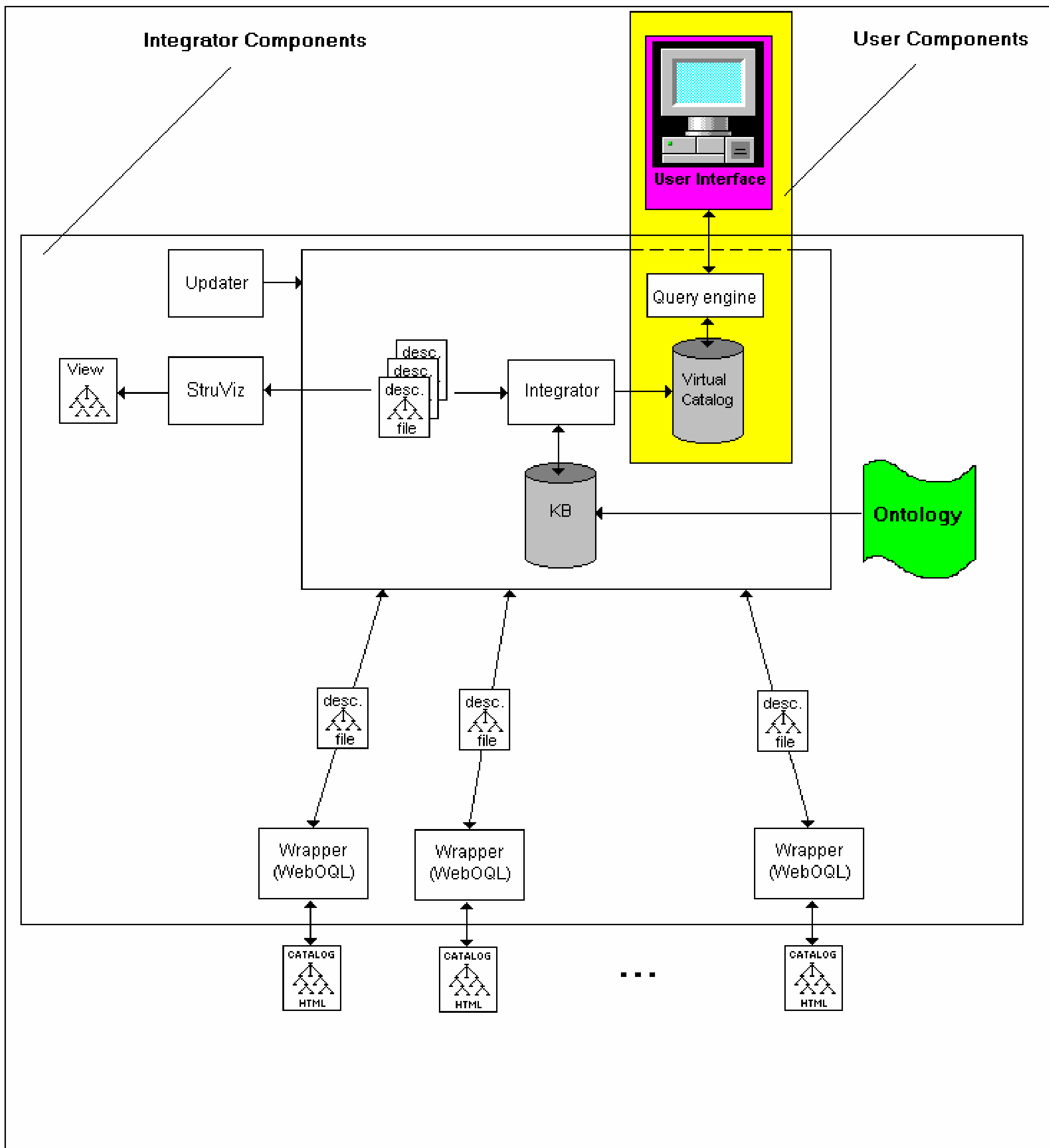

Figure 8: The WebCat architecture

Another component is the Updater. The Updater checks on a timely basis, if the catalog schemata are up to date. If not, the whole process is restarted.

The UC group contains the virtual catalog, the query engine and the user interface. The user interface is specific to each domain, and has to be rebuilt for each new implementation of WebCat.

Next, we present in detail each component of the architecture, except the wrappers, which we present in detail in Chapter 7.

6.3 Integrator Components

6.3.1 Wrappers

The major strength of our implementation resides in its wrappers; because of their importance, we present the wrappers in detail in Chapter 7. In what follows, we present only an overview of the wrappers in order to offer an understanding of the WebCat architecture.

The WebCat wrappers are much “lighter” than the first generation of wrappers, i.e. those employed by IIS. The WebCat wrappers extract the schema of each source and other information of interest. At the core of the WebCat wrappers are the WebOQL queries which are responsible of extracting both schemata and attributes. The process works as follows: first, we choose a representative page for each level. By representative we mean a page that has a structure that is repeated through the pages of the same level. Next, we establish the Abstract Syntax Tree (AST) for these representative pages. Then we write the WebOQL queries for each level according to the AST. Finally, we incorporate the WebOQL queries into the wrapper.

An element of interest for this stage of our presentation is the *description file format (DFF)*. DFF is the format of the file (DF) that encapsulates the description of a wrapped source; we present the DFF in detail in Chapter 7.

A description file is generated by a wrapper and is used by several other WebCat modules. The DFF augments the standard output format of the WebOQL query engine with information related to the levels. The standard WebOQL output is a tree in which the edges are represented by square brackets, and the records associated with the edges are explicitly specified as: <attribute; value> tuples.

In our approach, we built wrappers for each individual source. The wrappers extract the source schema and the necessary attributes. All this information is packed in description files, which are sent later to other modules.

6.3.2 StruViz

Deriving the AST, an essential element in the wrapper building process, is an error-prone task. In order to verify the operation, we provide a module called Structure Visualizer (StruViz). This component takes a description file as input, and displays the associated tree. In Figure 9 we display part of the structure of Eddie-Bauer's catalog. Once the structure of the catalog is visualized, it is easy to verify whether it is accurate. An accurate hierarchy is one that preserves the structure of the catalog (including the links).

StruViz is implemented on the top of Graphite, a universal tool for graph visualization developed at University of Toronto. Graphite can manipulate, visualize, and query multi-dimensional graphs (node-and-link diagrams) and hygraphs. Multi-dimensional graphs or multigraphs are graphs in which each vertex has associated multiple attributes (or dimensions). Hygraphs are multigraphs in which the nodes can be blobs. A blob associates a (container) vertex with a set of vertices that it contains. Graphite combines structural visualization, which helps in understanding the topological structure of a graph, and quantitative visualization of highly-dimensional data, which aids in exploring statistical relationships, trends and exceptions in data. A more detailed description is given by Noik [Noik96].

The role of StruViz is primarily a control one although we envision an additional role for this component. The essential role of StruViz is to visualize a description file. As we discuss later in this chapter, once the catalogs are integrated, we have a unique representation of a given collection. This representation can be easily translated into a description file format and inputted to StruViz. Consequently, we are able to “visit” the virtual catalog as a whole. Using the powerful query capabilities of Graphite, it is easy to pose complicated queries to virtual catalogs and to visualize only the components that satisfy the input query.

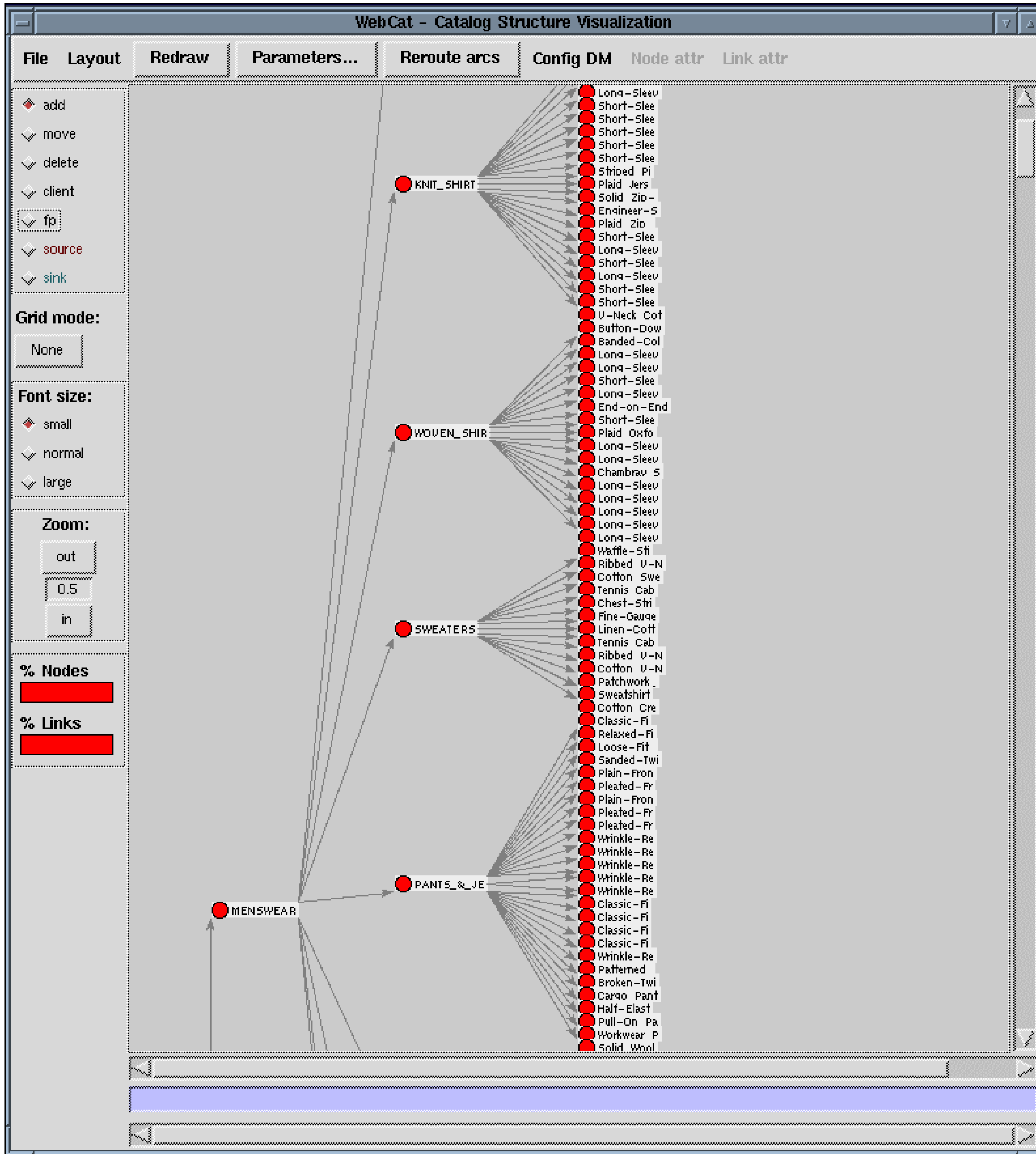

Figure 9: The Eddie-Bauer structure visualized by StruViz

6.3.3 Domain Knowledge-base

One of the key properties of WebCat is that we know ahead of time the domain of the sources. Once the domain is chosen, either we rely on an existing ontology, or we build an ontology of the domain. We build the ontology by using all possible terms at the finest granularity possible. For example in the case of clothing catalogs, if one catalog has the first branch above the leaves called shirts, and another has casual shirts, and shirts is one level above, we choose the lowest level possible. That is, our hierarchy will be shirts-> casual shirts. With this approach, we ensure a high accuracy for the retrieval.

The ontology is not fixed; it is updated each time it is necessary. A report is generated after each integration. Using this report, we can check whether the ontology still accommodates all possible terms, or whether some additional terms need to be added.

The ontology is stored in a knowledge base, which is implemented using the description file format.

6.3.4 The Integrator

As the name suggests, the Integrator assimilates a new catalog into the virtual catalog. The Integrator reads a description file, combines the new data with the existing data, and builds a new version of the virtual catalog.

The integrator maps the new catalog structure to the virtual catalog structure using the algorithm presented in Chapter 5. If the algorithm fails to work, which is the case for catalogs with a structure totally different from the structure of the virtual catalog, the mapping has to be done manually. However, in most cases only branches have to be mapped, which is no more than a few tens of records.

The algorithm that we presented in Chapter 5 is general purpose and accommodates the most complex ontology. In the case of “real” catalogs these ontologies are not very complex. For example for the clothing catalog the ontology of the domain has only four levels and the level of undecidability is one.

6.4 User Components

The User components are the “visible part of the iceberg”. These are the only components that the user interacts with. These components are the virtual catalog, the query engine and the user interface. The virtual catalog is a “cross-listed” component because it also belongs to the Integrator component. The other two components, the query engine and the user interface, are implementation dependent. Because these components are domain dependent, we present them as they are implemented in Apparel WebCat, the application for clothing catalogs.

6.4.1 Apparel WebCat

Apparel WebCat integrates clothing catalogs. They are the catalogs whose printed versions first promoted the products of fashion houses and departmental store chains. Lately, due to the expansion of the electronic commerce, most of these catalogs became available on the Web.

These catalogs are by no means simple Web catalogs. While in the university catalog, the HTML pages at any level have a relatively simple structure, in the clothing catalogs every detail of an HTML page pays tribute to the aesthetic look. Consequently, the HTML pages are very complex, using to the extreme the whole expressive power of the latest HTML versions.

The size of these catalogs is also significant. A clothing catalog has in the range of one thousand HTML pages. Because of this size, most of the pages are dynamically generated via CGI scripts. Dynamically generated pages are not an exception for our model since they are still organized in a hierarchy.

The levels in the hierarchy are also complicated. While in the university example, we had a clear hierarchy, in the clothing catalogs it is often the case that elements of distinct levels are nested in the same HTML page. The leaves, which are products in this case, have a regular structure through the catalog; therefore a large number of attributes can be extracted. These attributes include features such as the size, color, price, sale tag, new tag, etc.

As an example we integrated three catalogs: Eddie Bauer, Gap and JC Penney. They can be found at: [http://www.eddiebauer.com/eb/ShopEB/frame_line.asp], [http://www.gap.com], and [http://shopping.jcpenney.com/jcp/indev.asp], respectively.

Next we present in detail the User components as they are implemented in the Apparel WebCat.

6.4.2 The Query Engine

The Query Engine is a combination between an SQL query engine (SQL-QE) and an additional component, which performs full text search on specified attributes. The additional component is called Query Filter (QF).

We need the Query Filter because most of the data from the catalog is semi-structured. It is the case that some attributes may be, or may not be present even in the same catalog. Therefore, we use the SQL-QE to query on attributes that we know always exist, such as labels, catalog names, and prices. We use the Query Filter to "filter" on additional attributes such as size, color, and any other description from the same clothing catalog.

In order to explain how the Query Filter works, let's take the size attribute. No two catalogs use the same range of sizes. Furthermore, it is often the case that size comes in combination with color. Consequently, instead of considering a separate attribute for each size, we store all values for sizes in one attribute in a multi-valued field fashion. If we have a query request for a specific size, we first extract the size attribute using the SQL engine, and then we filter the result by checking each value from the multi-valued field size.

The idea of Query Filter is not a new one; a similar capability is built into the TSIMMIS wrapper (see [HG-MN+]).

Once the query answer is computed, the answer is formatted in a tabular form that shows all the specified details, and the URL for each leaf (see Figure 11). Thus, the user can navigate through the answer and can pick a leaf for a closer examination.

6.4.3 The User Interface

For each of the virtual catalogs that we built, we constructed a specific user interface. The user interface is implemented as an HTML form that communicates with the Query Engine via CGI scripts. We present the user interface for Apparel WebCat in Figure 10.

The interface reflects the ontology of the domain; also it allows users to make multiple selection at the first level above the leaves. In the clothing catalog, for example, the user can select an item such as T-shirt from menswear and a dress from womenswear; as a consequence, WebCat retrieves both categories.

In the example (see Figure 10), we query for menswear shirts, dress shirts, and T-shirts. Moreover, additional conditions can be specified. In our case, we query for regular size type with X, XL, XXL size, in any of the colors: white, red or navy with prices between $10 and $75 and being on sale. Here by “regular size” we mean one of the following size types: ”small”, “regular”, “tall”.

Part of the answer returned by the system is presented in Figure 11. We display the answer with respect to the additional attributes that describe the leaves (products). The answer can be ordered according to any of the attributes. An additional attribute is present, the product description, which is displayed if the user clicks on the product name.

The “Buy!” icon leads to the product itself. Because we store the URL of the products in our virtual catalog, the icon is a hyperlink to the product.

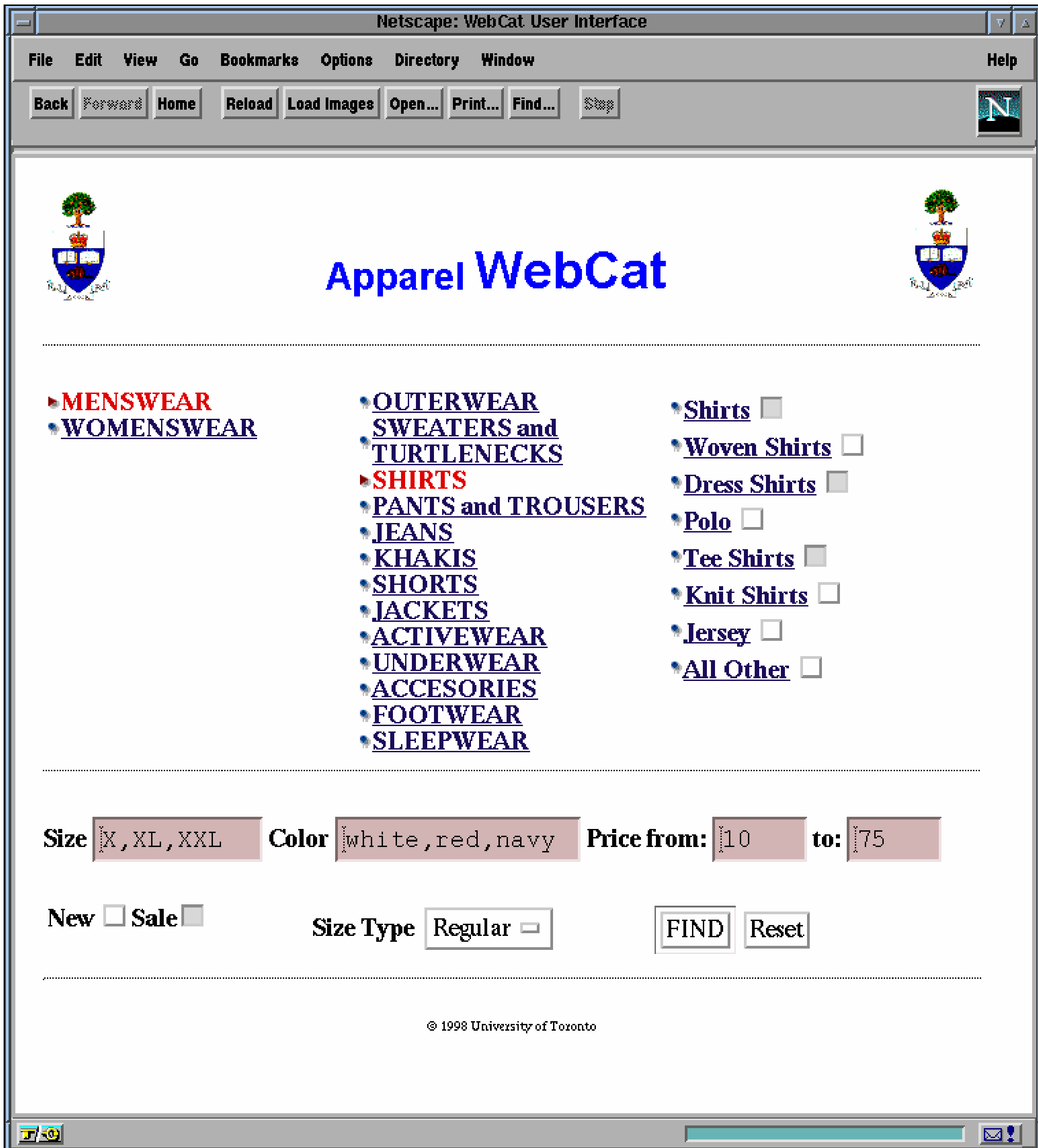

Figure 10: The Apparel WebCat User Interface

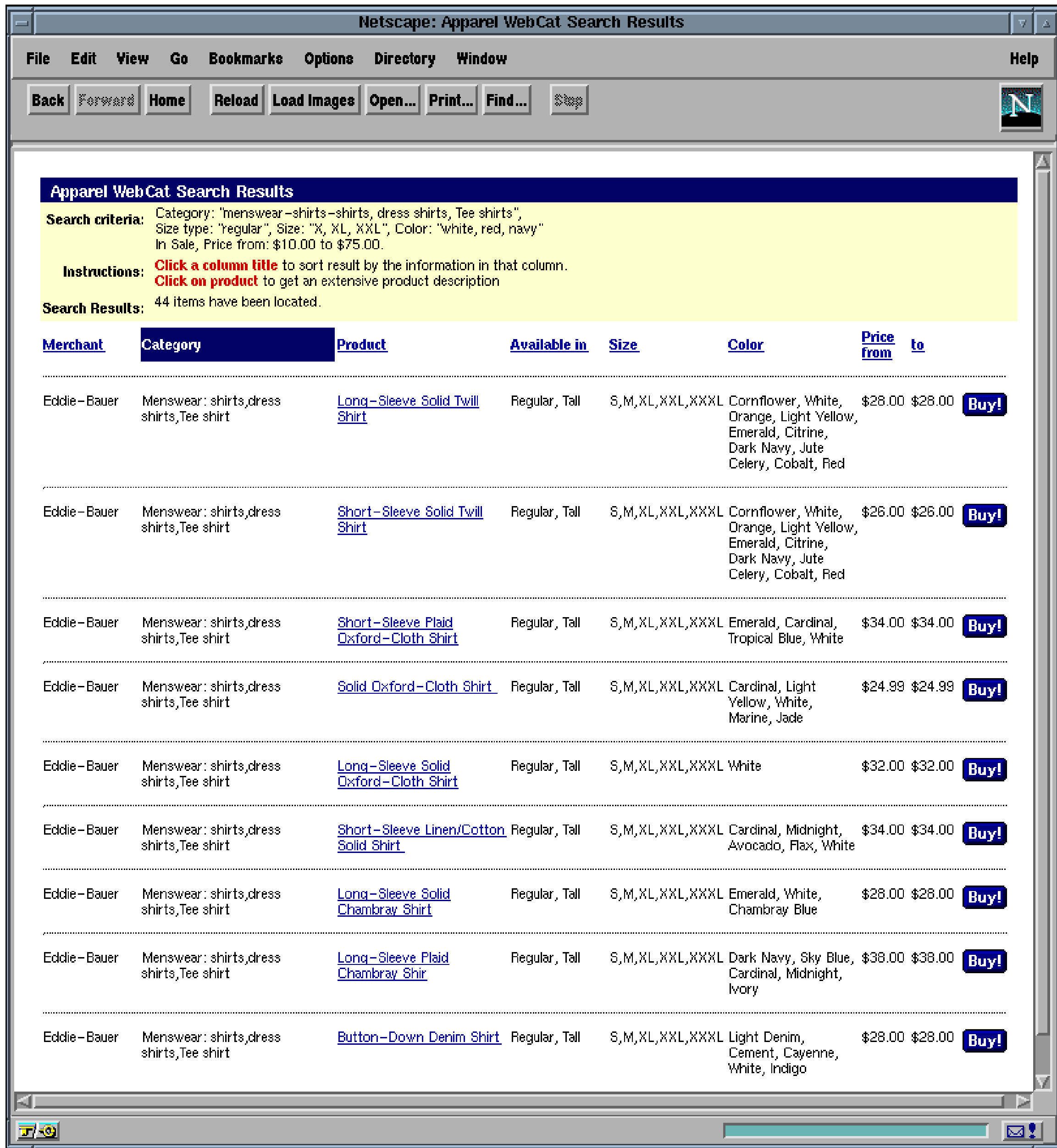

Figure 11: The Apparel WebCat result file

Chapter 7: WebCat Wrappers

7.1 Background

We mentioned that one of the main contributions of our method is found in its wrappers. WebCat wrappers are significantly different from any other wrappers used by similar systems. The main difference resides in the capability of our wrapper to extract the schema of Web catalogs. This capability in based on the use of the WebOQL language at the core of our wrappers. We present the wrappers by means of examples.

Wrappers are a common presence in IIS. They are first mentioned by Gio Wiederhold [Wie92], and early IIS systems such as TSIMMIS implement wrappers. In TSIMMIS [HG-MN+], the specification for wrappers is to encapsulate the data sources and mediate between them and the mediators. Furthermore, wrappers are responsible for part of the query plan and query execution. Since this early implementation of wrappers, several other systems use this approach. In most of these systems, wrappers have a much “lighter” purpose. Therefore, recently any middle-ware component between the information source and the system is considered a wrapper. The WebCat wrappers are no exception; they are much “lighter” than the initial ones.

Although numerous systems use wrappers, building wrappers for complex Web sources is a relatively new issue. As mentioned in Chapter 2, several projects extract and integrate data from the Web, but none of these deals with such complex sources as Web catalogs. The novelty of our approach resides in using WebOQL queries at the core of our wrappers. As mentioned earlier in Chapter 4, WebOQL supports a general class of data restructuring operations in the context of the Web. WebOQL synthesizes ideas from query languages for the Web, for semi-structured data and for website restructuring. The flexibility of WebOQL allows us to write complex queries in order to extract catalog structures.

7.2 WebCat Wrappers

In our system, wrappers have to be built for each Web catalog. In the existing system, this process is semi-automated. Our goal is to automate this process as much as possible; and in Chapter 8, we present a set of guidelines in order to achieve this goal.

The process of building WebCat wrappers is a layered one. We have to complete several steps in a strict order. These steps are:

- First, we identify the levels of the catalog and we chose a typical HTML page for each level.
- Second, we define the Abstract Syntax Tree (AST) for each level based on the typical page.
- Third, we define WebOQL queries for each level based on the typical page.
- Finally, we encapsulate the WebOQL queries in the wrapper.

We present by example how we built a wrapper. For this task, we have chosen the Eddie-Bauer (EB) clothing catalog. We present simultaneously the structure of EB catalog level by level and the wrapper layers for each corresponding level.

For the EB catalog, there are three levels of representation. Examples of HTML pages that belong to each level of representation are shown in Figures Figure 12, 16 and 22. The URLs for these pages are:

[http://www.eddiebauer.com/eb/ShopEB/frame_line.asp]
[http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312]
[http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10437&CatID=127
5&LineID=312]

Because each level poses different challenges we dedicate a separate sub-chapter for each level.

7.2.1 First Level

We start the building process by accessing the EB catalog home page (Figure 12). From the "look" of the page, we infer that the HTML page contains frames, and that the information of interest is found in one particular frame. In Figure 13 we present schematically how these frames are arranged. The frame of interest is "shopmain", because this frame contains the data. The other two frames, "shopnav" and "sysnav", have navigational purposes only.

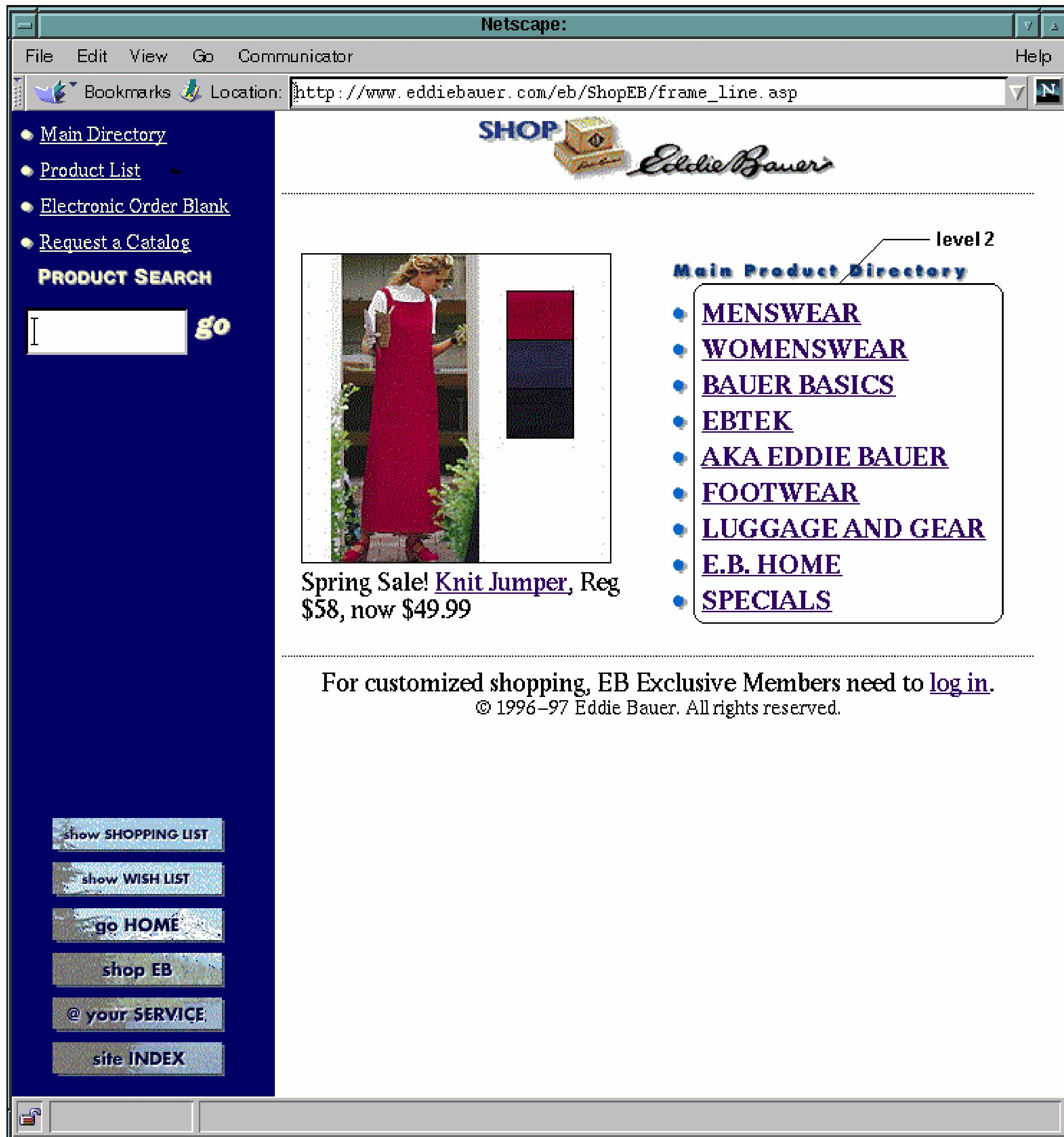

Figure 12: Eddie-Bauer catalog homepage (level 1)

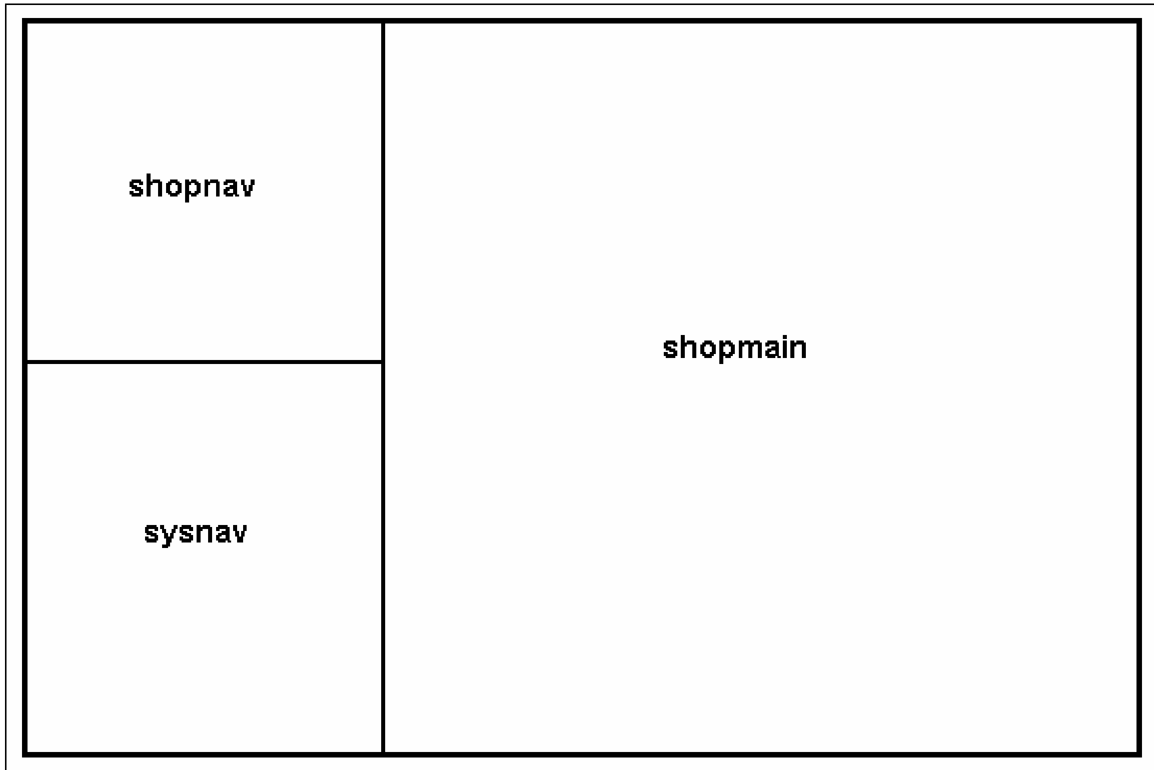

Figure 13: Frames in the catalog homepage

Once we established the presence of the frames, we check to see if the design is consistent. We check by going through a number of pages situated on different levels. The design is consistent, therefore each time we reach a page we consider only the “shopmain” frame, and disregard the other two. At this step, we have to note that the checking process is done manually, and thus is error prone, since we cannot check the thousands of files that are in the catalog.

The presence of the frames has another impact too. The URLs above are not the actual URLs of the frames. That is, the URLs are referring to the pages in which the frames reside. Since the frames can be isolated and loaded as “stand-alone” pages, we will further refer to them as pages.

In each of the figures 12, 16, and 22 we circled the links to the next level. Despite the simplistic look, the presented HTML pages are complicated. In Appendix 2 we present a listing of such an HTML page.

The next step is to establish the Abstract Syntax Trees (AST). In figures 14, 18 and 25 we present a simplified AST for each level. By simplified we mean that we

depicted only the HTML links that we have to follow in order to reach the information of interest.

In our representation of AST, each arrow identifies an HTML tag; if an arrow starts from another one, that means that the tags are nested. In the representation, we kept the original HTML tags, except for the last level that refers to another HTML page. For example, in Figure 14, the “tag” “level 2” is just a suggestion that this hyperlink represented by tag <A> refers to the next level, level 2 in this case.

In order to achieve a better understanding of how our system works, we present in detail not only the levels of the catalog, and the associated AST, but also the WebOQL queries that extract the information from each level.

In Figure 12, which represents the first level, we circled the information of interest, which are the links to the next level (level 2). It is easy to guess that all the links are grouped in a table. As depicted in Figure 14, the information of interest is really in a table. The problem is that this table is not quite easy to reach. Once we “navigate” to it, we have to iterate through the table’s rows in order to extract all the links pointing to the second level. The WebOQL query for this task is presented in Figure 15.

Next, we present a detailed description of how the query in Figure 15 works.

The query of interest is a compound one. The first query extracts the address of the frame of interest and stores it in the *new_target* variable; the second query extracts the data from the frame. The second query is the specific query for extracting the links towards level 2. The *target* variable in the first query is the address of the HTML page that we chose to be the typical page for level 1. The second query works as follows:

- First, we locate the first table: *via ^[tag = "table"],*
- Second, we navigate “down” to the table of interest: *((((x")!!)')!!!)',*
- Third, we iterate variable y through the rows of the table.
- Finally, variable z extracts from variable y (each row) the link to the second level: *(((y')!)')&.*

In the following, we present only the particular WebOQL queries since the query to extract the third frame is the same through all levels.

Figure 14: AST for level 1

```
# get the third frame
target := browse("http://www.eddiebauer.com/eb/
                                        ShopEB/frame_category.asp?LineID=312");

new_target := (select y.url from x in target, y in x'! );

# this is the query for level 2
select [z.base, z.url, z.text]
from x in new_target  via ^[tag = "table"],
y in ((((x'')!!)')!!!)', z in (((y')!)')&;
```

Figure 15: WebOQL query to extract the links to level 2

7.2.2 Sub-tree Group

In order to better manage the derivation of AST, we introduce a new informal term, that of *sub-tree group.* We need this term in order to describe exactly what we want to extract. For example, in Figure 14 we observe that the information that we are interested in, the links to the second level, is all situated at the end of sub-trees that stem from a common root, the second “Table” mark-up in this case. Moreover, the sub-trees have an identical structure. We call all these sub-trees instances of a sub-tree group.

Consequently, a sub-tree group is a collection of sub-trees from the AST that ranges over a non-trivial sub-tree pattern and all instances of the sub-tree group stem from a common root. By non-trivial pattern we mean a sub-tree of depth greater than one. That is, we do not want to capture sub-trees of depth one, which are very unlikely to contain information in which we are interested. In other words, the purpose of a sub-tree group is to capture repeating sub-trees.

In Figure 14 we circled an instance of a sub-tree group. Moreover, the sub-tree group from Figure 14 is bounded to the *y* variable in WebOQL query for extracting the links to level 2.

We divide the sub-tree groups into *basic sub-tree groups* and *augmented sub-tree groups*. A basic sub-tree group is a sub-tree group for which the sub-tree instances have the same structure, as we presented in Figure 14.

An augmented sub-tree group is a sub-tree group for which the sub-tree instances have a similar, but not identical, structure. By similar we mean a structure that differs in only a few arcs. We need to introduce the concept of augmented sub-tree group because it is possible to have some attributes (represented by arcs in the AST) that may or may not exist. This is the case of second level (see Figure 18) where the "sale" and "new" attributes may or may not be present for each sub-tree. This is the reason why in Figure 18 we present them with dotted lines.

7.2.3 Second Level

The second level pages are more complicated than the first ones. We expected each level to have a distinctive page. This is not the case in the EB catalog. As can be inferred from Figure 16, links to both level 3 and 4 are present on the same page. This is not an impediment, since we write separate WebOQL queries for each level.

An important aspect is the presence of some product attributes in the page. Two attributes "new" and "sale" are associated with some of the products. Consequently, along with the queries that extract links, we have to write queries that extract these attributes.

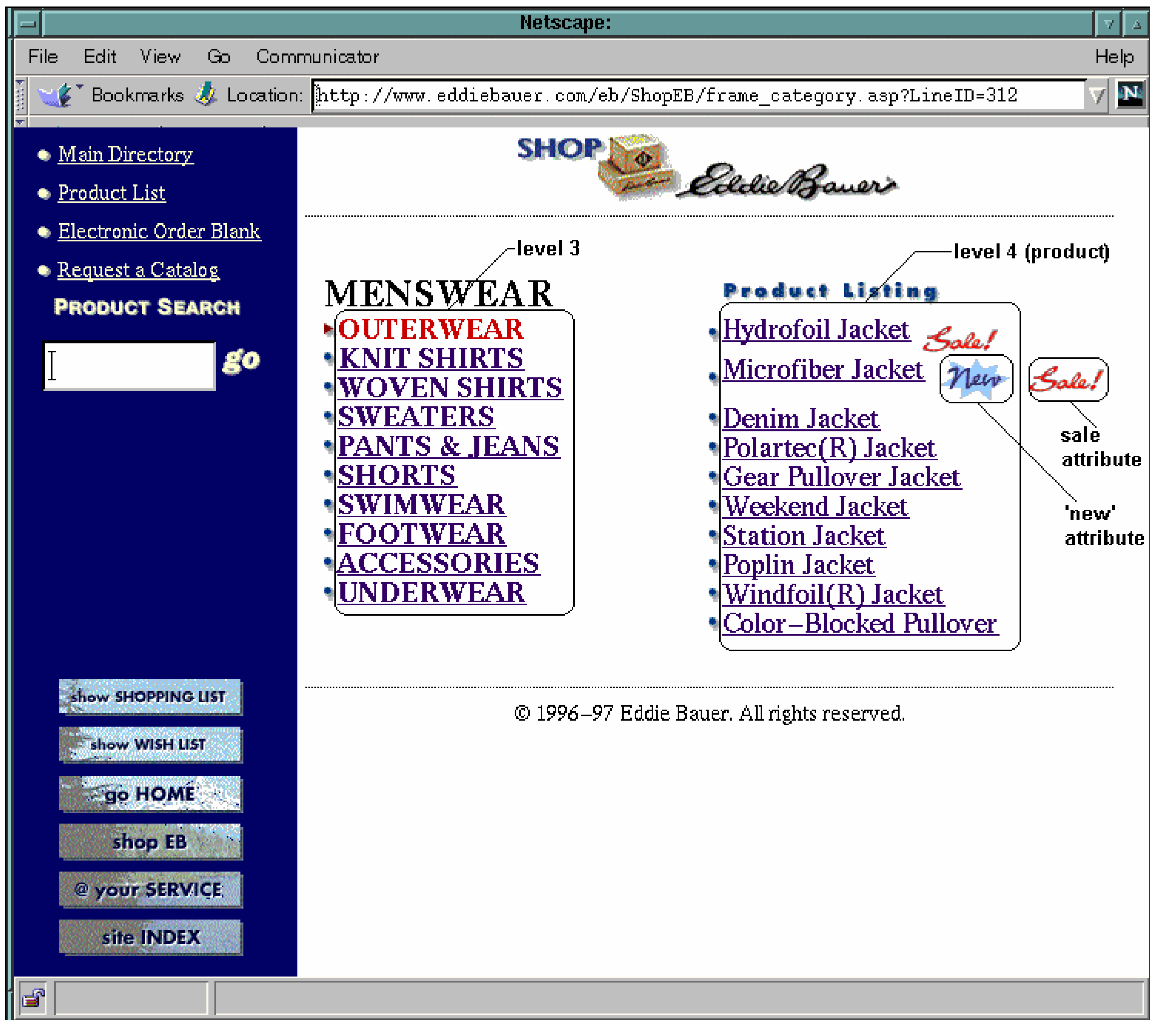

Figure 16: Eddie-Bauer Catalog, Level 2

We present the AST for these levels in Figure 18. We identify the table of interest as being the left one containing the links to the third level, and the right one containing the links to the fourth level (product level). The dotted lines in the right table represent the “new” and “sale” attributes with the associated tags. These attributes may, or may not be present for each product. These attributes represent a perfect example for an augmented sub-tree group. The optional attributes (“sale” and “new”) represent the “augmentation” of the “basic” sub-tree group.

Related to the third level links, the left table reveals a problem: the first third level “link” is actually not a link, rather it is a simple text. This can be deduced from the font used for “Outerware”, font which is not underlined – the typical convention for HTML links. After a check, we conclude that this problem is not consistent in all third level

pages. In each page, one “link” is not a link, but the position of this “non-link” varies from page to page. In our wrapper we overcome this problem by extracting all the links from two different third level pages, and then extracting a unique list of links for the third level.

This latter problem reveals the fact that even the most structured catalogs may present some kind of irregularities at some level. Furthermore, this problem is another argument for the necessity to build a tool to check the regularity of a structure.

We present the WebOQL queries to extract the level 3 links, level 4 links, and the “new” and “sale” attributes in figures 17, 19, 20, 21 respectively.

```
target := browse("http://www.eddiebauer.com/eb/ShopEB/
        category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR");

select  [z.base, z.url, z.text]
from x in target via ^[tag = "table"], y in ((x"!)'!!!!!)', z is ((((y')!)')')&;
```

Figure 17: WebOQL query to extract the links to level 3

Figure 18: AST for level 3 and level 4 (product)

```
target := browse("http://www.eddiebauer.com/eb/ShopEB/
     frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1275");

select [z.base, z.url, z.text]
from x in target via ^[tag = "table"],
y in ((((x'')!!)')!!!)', z is (((y')!)')&;
```

Figure 19: WebOQL query to extract the links to products

```
target := browse("http://www.eddiebauer.com/eb/ShopEB/
category.asp?ProdID=&CatID=1309&LineID=312&LineName=MENSWEAR");

select   [z.text,  t.src]
from x in target via ^[tag = "table"],
y in (((x''!!)')!!!)', t is ((((y')!)')!4);
```

Figure 20: WebOQL query to extract the “sale” attribute

```
target := browse("http://www.eddiebauer.com/eb/ShopEB/
category.asp?ProdID=&CatID=1309&LineID=312&LineName=MENSWEAR");

select   [z.text, w.src]
from x in target via ^[tag = "table"],
y in (((x''!!)')!!!)', z is (((y')!)')&, w is ((((y')!)')!!)&;
```

Figure 21: WebOQL query to extract the “new” attribute

7.2.4 Fourth Level

Figure 22: Eddie-Bauer Catalog, Level 4 (Product Level)

Using the terminology introduced in Chapter 5, the products are the leaves that we were looking for. We present an example of a product in Figure 22. Because the products are leaves, we have to extract additional information from them. This information is grouped according to four attributes: price, description, size type, size and color. By size type, we

mean the different groups in which the clothes are made such as: regular, tall, short, etc. We circled each attribute in Figure 22. We present the AST for this level in Figure 25.

In can be inferred from the AST how complex the structure of the product pages is. Another important aspect is the power of WebOQL, which is reflected, in its navigational capabilities that allow us to extract the information of interest. A possible alternative to WebOQL could be a full text search for certain patterns. For example, we can search for the dollar sign in order to get the price. In the case of the product in Figure 22, there are two dollar signs in the pages, and consequently two prices, from which only one is the real price, and this is the price from the title. In this respect, a full text search would yield an inconsistent answer.

We write separate WebOQL queries to extract each attribute. We present these queries in figures 23, 24, 26, 27 respectively.

```
target := browse("http://www.eddiebauer.com/eb/ShopEB/product.asp?
opcode=&item=&qty=&VariantID=&hemming=&inseam=&sku=&ProdID
=10769&CatID=1275&LineID=312&color=&WLItem=");

select [y.text]
from x in target via ^[tag = "form"],
y in (((((((x')!!!)')!)')!!)') where y.tag = "strong";
```

Figure 23: WebOQL query to extract the price for the products

```
target := browse("http://www.eddiebauer.com/eb/ShopEB/product.asp?
opcode=&item=&qty=&VariantID=&hemming=&inseam=&sku=&ProdID
=10437&CatID=1275&LineID=312&color=&WLItem=");

select [y.text]
from x in target via ^[tag = "form"],
y in ((((((((x')!!!)')!)')!!)')!!!)&;
```

Figure 24: WebOQL query to extract the description for the products

Figure 25: AST for level 4 (product)

```
target := browse("http://www.eddiebauer.com/eb/ShopEB/product.asp?
opcode=&item=&qty=&VariantID=&hemming=&inseam=&sku=&ProdID
=10769&CatID=1275&LineID=312&color=&WLItem=");

select y as schema
from x in target via ^[tag = "form"],
y in ((((((x')!!!)')!)')!!)' via [tag="table"]&
|
select z
from w in schema,
z in ((w')&2)!;
```

Figure 26: WebOQL query to extract the size type for the products

```
target := browse("http://www.eddiebauer.com/eb/ShopEB/product.asp?
opcode=&item=&qty=&VariantID=&hemming=&inseam=&sku=&ProdID
=10403&CatID=12&LineID=&color=&WLItem=");

select [z.text]
from x in target via ^[tag = "form"],
y in ((((((x')!!!)')!)')!!)', z in (y')!!!!
where y.tag = "select" and z.tag = "notag";
```

Figure 27: WebOQL query to extract the size and color for the products

We have to make here a special mention for the query associated with Figure 27 that extracts the size and color for the products. We infer from Figure 22 that the size and color attributes are displayed in a pop-up box. This approach is not an impediment for WebOQL because, for WebOQL the pop-up is only a list of “OPTION VALUE” HTML tags, thus is treated like any other list of tags.

7.3 The Wrapper

Once the WebOQL queries for each level are created, we glue them together in the wrapper. The wrapper is a Java application that accesses the WebOQL query engine using the WebOQL API.

As described in the previous section, the most complicated part of the wrapper building process is the definition of the AST for each level. The HTML pages are complicated, and not always regular or clean in terms of syntax. By regular we refer to the indentation in text, and the presence of comments. By clean, we mean the right nesting of tags. If just one tag is overlooked, the AST is no longer accurate and if this tag is in the "extraction path" the outcome becomes unpredictable. Once we have the AST, the WebOQL queries are easy to write.

In the process of building wrappers, we use WebOQL queries in an incremental manner. Consequently, we start with a known part of the AST that is easy to infer, and we write WebOQL queries to get "deeper" in the structure. Therefore, we propose a tool for deducing the AST in a semi-automated manner. We present the specification for such a tool in Chapter 8.

Building a wrapper is moderately time consuming. In our experience, a new wrapper needs 2-3 man/day effort. The major burden in this process is to establish the AST.

The wrapper outputs a *description file*, which is an abstract representation of the AST. This description file respects a format which we call *description file format (DFF).* We need a special file format for several reasons:

- First, we store the catalog description locally. The most time consuming process is the wrapping of the catalogs. We need a format to store the description since the integration process may iterate several times over a catalog description. Querying the information source for each iteration would be too costly.
- Second, we need a format for communicating among several modules. DFF files are not used only by the wrapper, but also by StruViz, which reads DFF files and displays the associated trees.

The DFF augments the standard output format of the WebOQL query engine with information of the levels. WebOQL, in the standard mode, outputs a tree in which the

edges are represented by square brackets, and the records associated with the edges are explicitly specified in the format: <attribute; value>. Furthermore, because of this format a WebOQL output can be used as input for another WebOQL query. We kept this functionality in our description files, so that any description files can be used as a input for a new WebOQL query. Arocena [Aro96] gives more details on how WebOQL accepts its inputs.

Because we use distinct WebOQL queries for distinct levels, we have to specify which output represents which level. For this reason, we augmented the WebOQL output format with information about the level. We present a part of the description file for EB in Appendix 3.

Chapter 8: Conclusion

8.1 Contribution

Our initial goal was to provide uniform access to large hierarchical collections of HTML pages, in which the HTML pages are interrelated by hyperlinks, and the collections offer information that pertains to a common domain. Our goal was inspired by the idea of integrating product catalogs that can be found on the Web, and which belong to a common domain.

In order to achieve our goal, first we define the concept of Web catalogs, which represents a particular information source on the Web. We defined a data model, the WebCat data model, to accommodate Web catalogs. We offered a foundation for this concept, and we presented a classification in order to delimit the sources of interest.

According to this classification, we are interested in integrating only structured, automatically integrable Web catalogs. For these catalogs we proposed WebCat, a method to integrate such Web catalogs.

Using this method, we defined the architecture of a system capable of implementing the WebCat method. We call this system also WebCat. In order to prove the soundness of our method, we built an implementation of the WebCat system for the domain of clothing catalogs, an implementation that we call Apparel WebCat. With Apparel WebCat, we integrated three popular catalogs: Eddie Bauer, Gap and JC Penney. The method proved to be feasible which encourages future enhancement of the method.

The novelty of our work resides in the integration method that we used and in our wrappers. We call the integration method used Information Integration with Semantics. The method used represents a semi-automated process for integrating information sources from a common domain. For building wrappers, we used WebOQL, a complex web-based query system. Our work represents the first major implemented application of this query language.

8.2 Shortcomings of the design

One of the novelties of WebCat resides in its wrappers, but wrapper building is the most time consuming process of the integration. Although wrappers are built in a semi-automated fashion, it is still a complicated process to build them.

The major burden of the wrapper building is to write the WebOQL queries. As mentioned earlier, in order to write a WebOQL query, first the AST should be inferred. In the case of large HTML files, which is our case, inferring the AST is not a trivial task and furthermore, is very error prone. Our experience shows that the best way of inferring the AST is to use WebOQL queries for this task too. The process that we use follows:

- First we infer the higher levels from the AST and we derive an approximate “look” of the AST.
- Next, we write WebOQL queries iterating first on the known higher levels, and we “increment” the queries to go “deeper” until we capture the whole AST.

This process, as mentioned, is very time consuming. Our intention is to build a tool to visualize the AST and furthermore to automatically generate the WebOQL query. This tool should act like a Graphical Query by Example for AST. The specifications for this tool are presented in a following section.

Another aspect that is time consuming is the process of building the interface. As mentioned, the interface is built according to the ontology of the domain. Since the elements of the interface are the same, only the content changes, according to the associated ontology. Our intention is to build an interface generator. This generator should read the ontology and automatically build the interface. This process would simplify the integration of a new domain.

The ontology is present not only in the interface, but also in the integration process. As mentioned, a specific integrator has to be built for each new domain. Our goal is to use a general-purpose ontology and to build a unique versatile Integrator.

As mentioned in the StruViz section, the visualizer that we actually use has an “engineering” purpose only. We use StruViz only to display the schema of the catalog that we want to integrate, in order to perform a visual check between the schema that we deduced and the real one. Our intention is to extend this function to the functionality of a browser. This will enable us to use the entire query power of Graphite, which is the

underlying system for StruViz. The browse capability will allow the user to “walk” through the virtual catalog, to perform visual queries, and to be able to get the leaves in an interactive way.

8.3 Future work

Integrating Web sources with WebCat proved to be a feasible task. The experience gained allows us to propose some improvements to the method. The main goal of improvements is to increase the degree of automation. At this state, our opinion is that the integration process cannot be made fully automatic, although some improvements are possible to increase the degree of automation.

We propose in the following two such tools, first a tool for automatically extracting the AST of an HTML document, and second a tool to compare the AST from different documents. We call these tools AST Extractor and AST Compare, respectively.

8.3.1 AST Extractor

As mentioned earlier, the most important part of the WebCat system are the wrappers. Moreover, the wrappers are the most time consuming part of using the system, not only because we have to build a wrapper for each catalog, but also because building wrappers is a complicated task by itself.

As presented in Chapter 7, the main burden of building a wrapper is in defining the AST of the sources. This process is equivalent to defining the AST for each level, which is equivalent to determining a representative HTML page for each level, and extracting its AST. Because of the complexity of the Web catalog HTML pages, this task is by no means simple or error free. Extracting the AST manually is very difficult in most cases. Although in the existing systems we used “progressive” WebOQL queries to advance “deeper” in the structure, we still had to infer and to draw the AST manually from the WebOQL output files. We want to overcome this problem by building an AST Extractor.

We present the specifications of the AST Extractor in a “to do list” fashion. The tasks that the Extractor should fulfill are:

1. Extract the AST of a specified HTML document.
2. Display in a graphical fashion the AST, providing implode/explode capabilities; in this way we will be able to implode parts of the trees that are irrelevant for our data.
3. Discover the sub-tree groups. That is, the Extractor should be capable to discover which are the possible sub-tree groups. This functionality will not only aid the user but also will ease the display. To display 20-30 identical sub-trees, which is the case of the products (see Figure 25), will take a lot of display space, when in fact these identical sub-trees are just instances of a sub-tree group. It is possible that in a page there are more than one sub-tree groups (see Figure 18). The sub-tree groups should be colored differently to distinguish them from the rest of the AST.
4. Allow us to indicate the augmented sub-tree groups. As an example, consider Figure 18. In the right sub-tree group, the “new” and “sale” attributes may or may not be present. Although we could have four distinct sub-tree groups according to all possible combinations, we want to be able to indicate that the “new” and “sale” are optional variables, and consequently, just one sub-tree group should be defined.
5. Allow us to choose the variables of interest. That is, once we have a sub-tree group, we should be able to specify which are the variables of interest. These variables can be part of a sub-tree group (see variable z from the y sub-tree group in Figure 14), or not (see the variable for price in Figure 25).
6. The paths that do not lead to a sub-tree group should be imploded by default, because more likely the variables of interest are in sub-tree groups. If they are not, we can explode the desired path. This procedure will eliminate unnecessary paths on the screen.
7. Once the variables of interest are chosen, the tool should generate the corresponding WebOQL query.
8. The tool should also be able to save the AST in a file description format.

8.3.2 AST Compare

We establish the AST for each level by choosing a typical HTML page. The question is, how significant is this page? Furthermore, do all pages have the same structure? In the actual system, we examine a few randomly chosen pages, to determine whether all pages have the established structure. Consequently, we have no way to measure whether all pages are consistent with the AST pattern. If a page does not respect the pattern, in consequence the data is not extracted properly from that page. The tool that we propose should be able to overcome this problem.

AST Compare should:

1. Accept as input a reduced AST in file description format and a list with the URLs of the pages to check. By reduced AST, we mean an AST in which all but the paths leading to variables of interest are imploded. The tool will check only if the variables of interest are on the right path. The outcome should not depend on whether the other paths respect the pattern.
2. Extract the reduced AST for each HTML file in the list using the AST Extractor.
3. Check all the HTML pages in the list for the matching AST pattern.
4. Display the ASTs that are not consistent with the pattern.

In conclusion, with these two new proposed tools, we will able to automate the process of wrapper generation almost completely.

Bibliography:

[AK97-1] Ashish, N., Knoblock, C., “Wrapper Generation for Semi-structured Internet Sources”, *ACM SIGMOD Workshop on Management of Semi-structured Data, Tucson, Arizona, 1997.*

[AK97-2] Ashish, N., Knoblock, C., “Semi-automatic Wrapper Generation for Internet Information Sources”, *Second IFCIS Conference on Cooperative Information Systems (CoopIS), Charleston, South Carolina, 1997.*

[AMM97] Arocena, G., Mendelzon, A.O., Mihaila, G., “Applications of a Web Query Language”*, Proceedings of the 6th International WWW Conference, Santa Clara, California, 1997.*

[AMM97-1] Atzeni, P., Mecca, G., Merialdo, P., “To Weave the Web”, *Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97), Athens, Greece, 1997.*

[AMM97-2] Atzeni, P., Mecca, G., Merialdo, P., “Semistructured and Structured Data in the Web: Going Back and Forth”, *In Proceedings of the Workshop on the Management of Semistructured Data (in conjunction with ACM SIGMOD), 1997.*

[Araneus] [http://poincare.inf.uniroma3.it:8080/Araneus/araneus.html]

[Aro97] Arocena, G., “WebOQL: Exploiting Document Structure in Web Queries”*, Master's Thesis, University of Toronto, 1997*.

[Cat96] Catell, R., (Ed.), “The Object Database Standard, ODMG-93”, *Morgan Kaufmann Publishers, San Francisco, Calif., 1996.*

[Deo74] Deo, N., “Graph Theory with Applications to Engineering and Computer Science”, *Prentice-Hall, Englewood Cliffs, N.J., 1974*.

[DEW97] Doorenbos, R.B., Etzioni, O., and Weld, D.S., “A Scalable Comparison-Shopping Agent for the World-Wide Web”, *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, California, 1997.

[Disco] [http://www-rodin.inria.fr/disco]

[Etz96] Etzioni, O., “Moving up the Information Food Chain: Deploying Softbots on the Web”, Proceedings of AAAI '96.

[FFK97] Fernandez, M., Florescu, D., Kang, J., Levy, A., Suciu, D., "STRUDEL: A Web-site Management System", *Proceedings of SIGMOD 1997, Tucson, Arizona.*

[FFK98] Fernandez, M., Florescu, D., Kang, J., Levy, A., Suciu, D., "Catching the Boat with Strudel: Experience with a Web-site Management System", *In Proceedings of ACM-SIGMOD International Conference on Management of Data , Seatle, WA, 1998.*

[FFLS97] Fernandez, M., Florescu, D., Levy, A., Suciu, D., "A Query Language for a Web-Site Management System", SIGMOD Record, vol. 26 , no. 3 , pp. 4-11 , September , 1997.

[Garlic] [http://www.almaden.ibm.com/cs/garlic/homepage.html]

[GM98] Arocena, G., Mendelzon, A.O., "WebOQL: Restructuring Documents, Databases and Webs", *Proc. of 14th. Intl. Conf. on Data Engineering (ICDE 98), Florida, 1998.*

[G-MHI+95] Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J., "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS", *Proceedings of the AAAI Symposium on Information Gathering, pp. 61-64, Stanford, California, March 1995.*

[Gru93] Gruber, T., "Toward Principles for the Design of Ontologies used for Knowledge Sharing", *The Padua Workshop on Formal Ontology, March 1993.*

[HG-MC+97] Hammer, J., Garcia-Molina, H., Cho, J., Aranha, R., Crespo, A., "Extracting Semistructured Information from the Web", *Proceedings of the Workshop on Management of Semistructured Data, Tucson, Arizona, 1997.*

[HG-MN+97] Hammer, J., Garcia-Molina, H., Nestorov, S., Yerneni, R., Breunig, M., Vassalos, V., "Template-Based Wrappers in the TSIMMIS System", *Proceedings of SIGMOD 1997, Tucson, Arizona.*

[HKL+98] Himmeröder, R., Kandzia, P.-Th., Ludäscher, B., May, W., Lausen G., "Search, Analysis, and Integration of Web Documents: A Case Study with FLORID", *Proceedings of International. Workshop on Deductive Databases and Logic Programming (DDLP'98), Manchester, UK, 1998.*

[HKWY97] Haas, L., Kossmann, D., Wimmers, E., Yang, J., "Optimizing Queries across Diverse Data Sources", *Proceedings of the 23rd VLDB Conference Athens, Greece, 1997.*

[HL98] Himmeröder, R., Ludäscher, B., “Querying the Web with FLORID”, *10. GI-Workshop: Grundlagen von Datenbanken (GvD'98), Konstanz, Germany, 1998.*

[HLL+97] Himmeröder, R., Ludäscher, B., Lausen G., Schlepphorst, C., “On a Declarative Semantics for Web Queries*”, Proceedings of the 5th Intl. Conference on Deductive and Object-Oriented Databases (DOOD'97), 1997, Montreux, Switzerland.*

[Kel97] Keller, A.M., "Smart Catalogs and Virtual Catalogs," *Readings in Electronic Commerce, Ravi Kalakota and Andrew Whinston, eds., Addison-Wesley, 1997.*

[Kel95] Keller, A.M., "Smart Catalogs and Virtual Catalogs," *International Conference on Frontiers of Electronic Commerce, October 1995.*

[KG96] Keller, A.M., Genesereth, M.R., "Multivendor Catalogs: Smart Catalogs and Virtual Catalogs*," The Journal of Electronic Commerce, Vol. 9, No. 3, September 1996.*

[KG98] Kushmerick, N., Grace, B. “The Wrapper Induction Environment”, *Workshop on Software Tools for Developing Agents, AAAI-1998*.

[KMA+98] Knoblock, C., Minton, S., Ambite, J.,L., Ashish, N., Modi, P., J., Muslea, I., Philpot, A., Tejada, S., “Modeling Web Sources for Information Integration”, *Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI, 1998.*

[KS97-1] Konopnicki, D., Shmueli, O., “W3QS: A Query System for the World-Wide Web”, *Proceedings of the 21rd International Conference on Very Large Databases (VLDB'95), Bombay, India, 1997.*

[KS97-2] Konopnicki, D., Shmueli, O., “W3QS - A System for WWW Querying”, *ICDE 1997: 586*

[KTV97] Kapitskaia, O., Tomasic, A., Valduriez, P., “Dealing with Discrepancies in Wrapper Functionality”, *INRIA Technical Report no 3138, 1997.*

[Kus97] Kushmerick, N., “Wrapper Induction for Information Extraction” Ph.D. Dissertation, Department of Computer Science & Engineering, University of Washington. Technical Report UW-CSE-97-11-04.

[KWD97] Kushmerick, N., Weld, D.S., Doorenbos, R.B., “Wrapper Induction for Information Extraction”, *Proceedings of IJCAI-1997.*

[Lee90] Leeuwen, J. van, “Handbook of Theoretical Computer Science” – Volume A, Algorithms and Complexity”, *The MIT Press/Elsevier, Cambridge, Mass., 1990.*

[Leh96] Lehmann, F., “Machine-Negotiated, Ontology-Based Electronic Data Interchange”, *Electronic Commerce, Curent Research Issues and Applications, Nabil Adam and Yelena Yesha, eds., Sringer, 1996.*

[LG98] Lawrence, S., Giles, C.L., “Searching the World Wide Web”, *Science, 3 April 1998, Volume 280, Number 5360.*

[LSS96] Laks Lakshmanan, L.V.S., Sadri, F.N., Subramanian, I.N., “A Declarative Language for Querying and Restructuring the WEB”, *RIDE-NDS 1996: 12-21.*

[MAM+98] Mecca, G., Atzeni, P., Masci, A., Merialdo, P., Sindoni, G., “The Araneus Web-Based Management System”, *Exhibits Program of ACM, SIGMOD '98, 1998.*

[Mih96] Mihaila, G., “WebSQL - an SQL-like query language for the WWW”, *MSc. Thesis, University of Toronto, 1996.*

[MMM96] Mendelzon, A., Mihaila, G., Milo, T., “Querying the World Wide Web”, *Proceedings of PDIS'96, Miami, Florida, 1996*.

[Noik96] Noik, E.G., “Dynamic Fisheye Views: Combining Dynamic Queries and Mapping with Database Views”, *PhD Thesis, Computer Science Department, University Of Toronto, 1996.*

[PDEW97] Perkowitz, M., Doorenbos, R.B., Etzioni, O., Weld, D.S., “Learning to Understand Information on the Internet: An Example-Based Approach”, *Journal of Intelligent Information Systems 8(2): 133-153 (1997).*

[Strudel] [http://www.research.att.com/~suciu/strudel/external/External_NodeSiteGraph_proj_RootProjects.html#strudel]

[TrS97-1] Tork Roth, M., Schwarz, P., “Don’t Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources”, *Proceedings of the 23rd VLDB Conference Athens, Greece, 1997.*

[TrS97-2] Tork Roth, M., Schwarz, P., “An Architecture for Legacy Data Sources”, IBM Technical Report RJ10077.

[TRV95] Tomasic, A., Raschid, L., Valduriez, P., “Scaling Heterogeneous Databases and the Design of Disco, ”, *INRIA Technical Report no 2704, 1995.*

[TSIMMIS] [http://www-db.stanford.edu/tsimmis/tsimmis.html]

[W3QL] [http://www.cs.technion.ac.il/~konop/w3qs.html]

[WebLog] [http://www.cs.concordia.ca/~special/bibdb/weblog.html]

[WebOQL] [http://www.cs.toronto.edu/~weboql]

[WebSQL] [http://www.cs.toronto.edu/~websql]

[Wie92] Wiederhold, G., "Mediators in the Architecture of Future Information Systems", *IEEE Computer, March 1992, pages 38-49.*

Appendix 1

Example of an HTML page from the Eddie-Bauer catalog

The following HTML code represents the code for the HTML pages from figure 21.

```
<HTML>

<HEAD><TITLE>Eddie Bauer Product Page</TITLE>

<META NAME="GENERATOR" Content="Microsoft Visual InterDev 1.0">

<META HTTP-EQUIV="Content-TYPE" content="text/html; charset=iso-8859-1">
</HEAD>

<BODY BGCOLOR="#FFFFFF" LINK="#1C0B5A" VLINK="#1C0B5A">

<!--------- BEGIN FORM -----------><!--<FORM TYPE="get" ACTION="redirect.asp"
TARGET="shopmain">-->
<FORM TYPE="post" ACTION="redirect.asp" TARGET="_parent">

<!-- HEADER -->

<CENTER>
<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
<TR>
        <TD ALIGN="right">
                <IMG SRC="/EB_assets/images/ShopEB/shopheadshort.gif"
ALT="Abbrev. Shop Logo" BORDER="0" ALIGN="middle" WIDTH="114" HEIGHT="47">

        </TD>
        <TD ALIGN="left">
                <STRONG>
                <FONT SIZE="0">

                <IMG SRC="/EB_assets/icons/sportswear.gif" ALT="Store Logo"
BORDER="0" ALIGN="middle" WIDTH="170" HEIGHT="38">

                </FONT></STRONG>
        </TD>
</TR>
<TR>
        <TD>
        </TD>
```

```
</TR>
</TABLE>
</CENTER>

<!------  Begin Main Table ------>

<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0">
<TR>
        <TD COLSPAN=2><HR SIZE="1"></TD>
</TR>

<TR>
        <TD VALIGN="top">
  <!-- Product Photo BEGIN -->

  <TABLE BORDER="0" CELLPADDING=0 CELLSPACING=0>
  <TR>
   <TD VALIGN="top">

      <IMG SRC="/EB_assets/prodimg/B010245.jpg"
                          ALT="Product Image: B010245.jpg"
          WIDTH="216" HEIGHT="216">

   </TD>
        <TD></TD>
  </TR>

  <!-- Product Photo END -->

  <TR>
        <TD NOWRAP="NOWRAP">

          <!-- Browse Options BEGIN-->

      <A
HREF="http://www.eddiebauer.com/eb/ShopEB/product.asp?ProdID=11559&CatID=127
5&LineID=312"><IMG SRC="/EB_assets/images/ShopEB/lprev.gif" ALT="Browse
Previous" WIDTH="114" HEIGHT="15" BORDER="0"></A>
                <A
HREF="http://www.eddiebauer.com/eb/ShopEB/product.asp?ProdID=11175&CatID=127
5&LineID=312"><IMG SRC="/EB_assets/images/ShopEB/lnext.gif" ALT="Browse Next"
WIDTH="110" HEIGHT="15" BORDER="0"></A>

          <!-- Browse Options END-->

        </TD></TR>

   <TR><TD NOWRAP="NOWRAP">
     <!-- Ordering Options BEGIN -->
```

```
                    <INPUT TYPE="image" NAME="wish"
SRC="/EB_assets/images/ShopEB/addwish.gif" BORDER="0" WIDTH="114"
HEIGHT="21"></INPUT>

       <INPUT TYPE="image" NAME="order"
SRC="/EB_assets/images/ShopEB/order.gif" BORDER="0" WIDTH="110"
HEIGHT="21"></INPUT>
                    </TD>

            <!-- Ordering Options END -->

  <TD></TR>


  <TR><TD>

          <!-- horizontal divider -->

          <IMG SRC="/EB_assets/images/sm_rule.gif" ALT="line" WIDTH="222"
HEIGHT="6" BORDER="0">

  </TD></TR>

  <TR><TD ALIGN="center">

          <!-- Table for Similar, Creed, Matching BEGIN -->

           <TABLE BORDER="0">
           <TR>

            </TR>
            </TABLE>
          <!-- Table for Similar, Creed, Matching END -->

  </TD></TR>
  </TABLE>

</TD>

<!--========== Product Information BEGIN ==========-->


<TD VALIGN="top">

          <STRONG>Microfiber Jacket ~ $88.00
          </STRONG>
          <P>
          <FONT SIZE="-1">The tight weave of our Microfiber Jacket makes it very
durable and naturally water-resistant, yet soft and luxurious to the touch. The upper body
is lined with 100% cotton, yarn-dyed in a rich plaid pattern; the lower body is lined with
```

```
cotton poplin. Nylon-taffeta-lined sleeves. A Teflon(R) coating resists rain and stains.
Machine wash. Imported. Colors as shown top to bottom:<P>Colors: Fawn, Juniper
   </FONT>
        <P>

        <!-- Style information BEGIN -->
                <TABLE BORDER="0" CELLPADDING="3" CELLSPACING="0">

                <TR>
                        <TD COLSPAN=3>Available in:</TD>
                </TR>

                        <TR>
                                <TD>
                                </TD>

                                <!-- CURRENT SELECTED STYLE -->
                                        <TD>
                                                <IMG
SRC="/EB_assets/images/ShopEB/checkbox.gif" ALT="selected style button"
BORDER="0" WIDTH="29" HEIGHT="29" ALIGN="bottom">
                                        </TD>
                                        <TD>
                                                Tall
                                        </TD>
                                        <TD>
                                                i01 690 0246
                                        </TD>
                        </TR>
                </TABLE>

        <!-- Style information END -->

        <INPUT TYPE="hidden" NAME="sku" VALUE="i01 690 0246"></INPUT>
        <INPUT TYPE="hidden" NAME="ProdID"  VALUE="10437"></INPUT>
        <INPUT TYPE="hidden" NAME="CatID"  VALUE="1275"></INPUT>
        <INPUT TYPE="hidden" NAME="LineID"  VALUE="312"></INPUT>

        <!-- editing item number, if present -->
        <INPUT TYPE="hidden" NAME="item" VALUE=""></INPUT>

        <BR>
        Quantity:
        <INPUT TYPE="text" NAME="qty" SIZE="3" MAXLENGTH="2"
VALUE="1"></INPUT>

        <BR>
        <BR>
        <!--==========  BEGIN color and size information =========-->

                <SELECT NAME="VariantID" SIZE="1">
```

```
                <OPTION VALUE="**">Select a Size and Color:</OPTION>
                <OPTION VALUE="**">--------------------------------------</OPTION>
               <OPTION VALUE="40702">M - FAWN<OPTION VALUE="40705">M -
JUNIPER<OPTION VALUE="40706">L - FAWN<OPTION VALUE="40709">L -
JUNIPER<OPTION VALUE="40710">XL - FAWN<OPTION VALUE="40712">XL -
JUNIPER<OPTION VALUE="40715">XXL - FAWN<OPTION VALUE="40716">XXL -
JUNIPER<OPTION VALUE="40718">XXXL - FAWN<OPTION VALUE="40720">XXXL -
JUNIPER
               </SELECT>

        <!--==========  END color and size information =========-->

        <BR>
        <BR>
        <P>

        <!--------- Gift Wrap BEGIN nested table ----------->
        <TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">
        <TR>
               <TD BGCOLOR="#f6f6e3">
               <INPUT TYPE="checkbox" NAME="giftwrap">Gift Wrapping (add $5 per
item)</INPUT>
               <BR>
               </TD>
        </TR>
        <TR>
               <TD ALIGN="right">
               To: <INPUT TYPE="text" NAME="giftto" VALUE=""
MAXLENGTH="25"></INPUT>
               <BR>
               From: <INPUT TYPE="text" NAME="giftfrom" VALUE=""
MAXLENGTH="25"></INPUT>
               <BR>
               Message: <INPUT TYPE="text" NAME="giftmess" VALUE=""
MAXLENGTH="25"></INPUT>
               <BR>
               </TD>
        </TR>
        </TABLE>
        <!----------- Gift Wrap END nested table ---------><!-- Product Information END -->
</TD></TR>
</TABLE>
<!-- end table -->
</FORM>
<!-- end form --><HR SIZE="1"><CENTER><FONT COLOR="#000000" SIZE=-
1>&copy; 1996-97 Eddie Bauer. All rights reserved.</FONT></CENTER>
</BODY>
</HTML>
```

Appendix 2

An example of File Description Format

The example is a small part from the Eddie-Bauer description file. The information related to the level is *in bold for better visibility.*

parent: root
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312",
text:"MENSWEAR"]
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=313",
text:"WOMENSWEAR"]
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=337",
text:"BAUER_BASICS"]
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=324",
text:"EBTEK"]
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=336",
text:"AKA_EDDIE_BAUER"]
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=325",
text:"FOOTWEAR"]
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=326",
text:"LUGGAGE_AND_GEAR"]
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=327",
text:"E_B__HOME"]
[base:"http://www.eddiebauer.com/eb/ShopEB/line.asp?ProdID=&CatID=&LineID=",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=332",
text:"SPECIALS"]
parent: root->MENSWEAR
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1275", text:"OUTERWEAR"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1389", text:"KNIT_SHIRTS"]

[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1390", text:"WOVEN_SHIRTS"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1278", text:"SWEATERS"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1280", text:"PANTS_&_JEANS"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1354", text:"SHORTS"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1442", text:"FOOTWEAR"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1309", text:"ACCESSORIES"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_category.asp?LineID=312&LineName=MENSWEAR&CatID=1430", text:"UNDERWEAR"]
parent: root->MENSWEAR->OUTERWEAR
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10768&CatID=1275&LineID=312", text:"Windfoil(R)_Jacket"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10769&CatID=1275&LineID=312", text:"Hydrofoil_Jacket"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10417&CatID=1275&LineID=312", text:"Polartec(R)_Jacket"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10437&CatID=1275&LineID=312", text:"Microfiber_Jacket"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10467&CatID=1275&LineID=312", text:"Gear_Pullover_Jacket"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",

url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10468&CatID=1275&LineID=312", text:"Weekend_Jacket"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=9510&CatID=1275&LineID=312", text:"Reversible_Quilted_Vest"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=9096&CatID=1275&LineID=312", text:"Nordic_Polarfleece_Pullover"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=8004&CatID=1275&LineID=312", text:"Microfiber_Parka"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=9511&CatID=1275&LineID=312", text:"Nylon_Bomber_Jacket"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1275&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=9515&CatID=1275&LineID=312", text:"Vintage_Wool_Jacket"]

parent: root->MENSWEAR->KNIT_SHIRTS

[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10397&CatID=1389&LineID=312", text:"Short-Sleeve_Pique__Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10396&CatID=1389&LineID=312", text:"Long-Sleeve_Pique__Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10758&CatID=1389&LineID=312", text:"Short-Sleeve_Interlock_Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10759&CatID=1389&LineID=312", text:"Long-Sleeve_Interlock_Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10780&CatID=1389&LineID=312", text:"Short-Sleeve_Tattersall_Pique__Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10782&CatID=1389&LineID=312", text:"Short-Sleeve_Striped_Pique__Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&LineID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10784&CatID=1389&LineID=312", text:"Short-Sleeve_Pigment-Dyed_Pique__Polo"]

[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10788&CatID=1
389&LineID=312", text:"Short-Sleeve_Striped_Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10791&CatID=1
389&LineID=312", text:"Short-Sleeve_Pique__Polo_with_Tipping"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10411&CatID=1
389&LineID=312", text:"Striped_Pique__Short-Sleeve_Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10410&CatID=1
389&LineID=312", text:"Plaid_Jersey_Short-Sleeve_Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10469&CatID=1
389&LineID=312", text:"Solid_Zip-Neck_Polo_Jersey_Knit"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10471&CatID=1
389&LineID=312", text:"Engineer-Stripe_Zip-Neck_Polo_Jersey_Knit"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=7976&CatID=13
89&LineID=312", text:"Plaid_Zip_Polo"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10766&CatID=1
389&LineID=312", text:"Short-Sleeve_Basic_Tee"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10460&CatID=1
389&LineID=312", text:"Long-Sleeve_Basic_Tee"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10767&CatID=1
389&LineID=312", text:"Short-Sleeve_Pocket_Tee"]
[base:"http://www.eddiebauer.com/eb/ShopEB/category.asp?ProdID=&CatID=1389&Lin
eID=312&LineName=MENSWEAR",
url:"http://www.eddiebauer.com/eb/ShopEB/frame_product.asp?ProdID=10459&CatID=1
389&LineID=312", text:"Long-Sleeve_Pocket_Tee"]