**Universidade Federal de Minias Gerais**
**Instituto de Ciências Exatas**
**Departamento de Ciência da Computação**

# Estratégias Baseadas em Exemplos para Extração de Dados Semi-Estruturados da Web

**Altigran Soares da Silva**

**Belo Horizonte**
**Junho de 2002**

**Universidade Federal de Minias Gerais**
**Instituto de Ciências Exatas**
**Departamento de Ciência da Computação**

# Example-Based Strategies for Extracting Semistructured Web Data

**Altigran Soares da Silva**

**Belo Horizonte**
**Junho de 2002**

Altigran Soares da Silva

# Estratégias Baseadas em Exemplos para Extração de Dados Semi-Estruturados da Web

Tese apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Belo Horizonte
26 de junho de 2002

Folha de Aprovação AQUI

# Resumo

Neste trabalho, são propostas, implementadas e avaliadas estratégias e técnicas para o problema de extração de dados semi-estruturados de fontes de dados da Web, dentro do contexto de uma abordagem chamada *DEByE (Data Extraction By Example)*. Os resultados obtidos com o trabalho foram usados na implementação de um ferramenta de extração de dados, também chamada DEByE, e tiveram sua eficácia verificada através de experimentação.

A abordagem DEByE é dita semi-automática, no sentido em que o papel dos usuários (ou seja, dos desenvolvedores de extratores) é limitado ao fornecimento de exemplos dos dados a serem extraídos, o que os isola de ter que conhecer as características especificas de formação das páginas alvo. Os exemplos fornecidos descrevem a estrutura dos objetos a serem extraídos por meio de tabelas aninhadas, as quais são simples, intuitivas e expressivas o suficiente para representar a estrutura dos dados normalmente presentes em páginas Web. Para lidar com variações típicas de objetos complexos semi-estruturados, estendemos o conceito original de tabelas aninhadas, relaxando a restrição original de que as tabelas aninhadas em uma mesma coluna devem todas ter a mesma estrutura interna.

Com base nesta forma estendida de tabelas aninhadas, formalizamos o conceito de *wrapper* através de gramáticas tabulares. Tais gramáticas livres de contexto são compostas por produções que levam a árvores de *parsing* que podem ser diretamente mapeadas para tabelas aninhadas. Foram desenvolvidas estratégias para gerar gramáticas tabulares a partir de um conjunto de objetos de exemplo fornecidos por um usuário que os seleciona de uma página de amostra. O processo de geração inclui: (1) geração de produções terminais para extrair valores atômicos pertencentes a um domínio específico (por exemplo, uma descrição de um item, um preço, etc.) e (2) a geração de produções não-terminais que representam a estrutura de objetos complexos a serem extraídos.

A extração dos dados das páginas alvo é feita através de um processo de *parsing* sobre estas páginas usando a gramática tabular. Para isso, desenvolvemos uma eficiente estratégia *bottom-up*, a qual envolve duas fases distintas: uma fase de extração, na qual os valores atômicos dos atributos são extraídos como base na informação de contexto local disponível nas produções de extração, e uma fase de montagem, na qual os valores atômicos extraídos são montados de acordo com a estrutura alvo fornecida pelo usuário através de exemplos e que está representada nas produções não terminais. A eficácia da estratégia *bottom-up* foi comprovada experimentalmente, em especial para tratar objetos complexos multi-nível que apresentam variações estruturais.

O principio geral utilizado pela estratégia *bottom-up*, ou seja, primeiro extrair valores atômicos e depois agrupar estes valores para montar objetos complexos, foi também explorado por um outro algoritmo que desenvolvemos chamado *Hot Cycles*. Este algoritmo tem como objetivo descobrir estruturas tabulares plausíveis para montar objetos complexos com um conjunto de valores atômicos extraídos de uma página alvo. Ele é útil para o emprego da abordagem DEByE em aplicações onde não se pode depender do usuário para montar tabelas de exemplo.

# Abstract

In this work we propose, implement and evaluate strategies and techniques for the problem of extracting semistructured data from Web data sources within the context of an approach we call *DEByE (Data Extraction By Example)*. The results we have reached have been used in the implementation of a data extraction tool, also called DEByE, and have their effectiveness verified through experiments.

The DEByE approach is semi-automatic, in the sense that the role of users (i.e., wrapper developers) is limited to providing examples of the data to be extracted, what shields them from being aware of specific formatting features of the target pages. The examples provided describe the structure of the objects being extracted by means of nested tables, which are simple and intuitive, and expressive enough to represent the structure of the data normally present in Web pages. To deal with typical variations of complex semistructured objects, we have extended the original concept of nested tables by relaxing the original assumption that all inner tables nested in a column should have a same internal structure.

Based on this extended form of nested tables, we formalize the concept of wrappers by means of tabular grammars. Such context-free grammars are formed by productions that lead to parse trees that can be directly mapped to nested tables. We have developed strategies for generating tabular grammars from a set of example objects provided by a user from a sample page. This includes: (1) the generation of terminal productions for extracting single values belonging to a specific domain (e.g., an item description, a price, etc.) and (2) the generation of non-terminal productions that represent the structure of the complex objects to be extracted.

The extraction of data from target pages is accomplished by parsing these pages using a tabular grammar. For this, we have developed an efficient bottom-up strategy. This strategy includes two distinct phases: an extraction phase, in which atomic attribute values are extracted based on local context information available in the extraction productions, and an assembling phase, in which such values are assembled to form complex objects according to the target structure supplied by the user through examples, which is encoded in the non-terminal productions. We experimentally demonstrate the effectiveness of the bottom-up strategy for dealing with multi-level objects presenting structural variations.

The general principle used by the bottom-up algorithm, that is, first extracting atomic values and then grouping these values to assemble complex objects, has been further exploited by the Hot Cycles algorithm we have developed. This algorithm aims at uncovering a plausible tabular structure for assembling complex objects with a given set of atomic values extracted from a target page. This algorithm is useful for deploying the DEByE approach in applications where the user is not available for assembling example tables.

# Agradecimentos

Agradeço primeiramente a Deus, pelas generosas bençãos e por ter me permitido chegar até aqui.

O desenvolvimento e a conclusão deste trabalho não teriam sido possíveis sem a participação de várias pessoas a quem tenho a alegria de registrar meu agradecimento.

Aos meus orientadores Prof. Alberto Henrique Frade Laender (o Chefe) e Prof. Berthier Ribeiro-Neto, não só por mostrarem o caminho, mas também pela paciência, apoio, amizade e confiança.

A todos os meus colegas, alunos de pós-graduação do DCC/UFMG, em particular à Karine Chaves, Paulo Golgher, Karine Versieux, Irna Evangelista, Pável Calado, Tatiana Coelho, Karine Louly, Rodrigo Cardoso, Rodrigo Barra, Eveline Veloso, Luciano Lima, Juliana Teixeira, Allisson Arantes, Denilson Barbosa, Betânia Barbosa, Maria de Lourdes, Edleno Moura, Karla Albuquerque, Joyce Paiva, Manoel Palhares, Davi Reis e Robson Braga. Tenho orgulho de dizer que o desenvolvimento deste trabalho contou com a colaboração, direta ou indireta, destas pessoas.

Aos demais professores e funcionários do DCC/UFMG por seu apoio e pela dedicação. Gostaria de agradeçar particularmente ao Professor Nívio Ziviani pelas oportunidades e pelo apoio. Também agradeço aos amigos da Akwan Information Technologies pela chance de testemunhar como tecnologia de ponta se aplica no dia-a-dia.

Aos membros externos da banca examinadora, Professores Carlos Heuser, David Embley e Marco Casanova que me deram a honra de ter meu trabalho avaliado por eles com muita dedicação. Fico feliz de ter tido a oportunidade de aprender com eles.

Meus agradecimentos também à CAPES, CNPq e Universidade Federal do Amazonas por proporcionarem o suporte financeiro para este trabalho, e também aos colegas do Departamento de Ciência da Computação da UFAM, um belo projeto coletivo de desenvolvimento científico e tecnológico na região Norte, do qual tenho a honra de participar.

Agradeço finalmente à minha família: Ana, minha mãe, Aliny, minha irmã, e Altino, meu pai (*in memoriam*) pelo amor, dedicação e paciência.

À Tânia, minha esposa, mais do que agradecer pelo amor, carinho, paciência e atenção, dedico carinhosamente este trabalho.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Over the last decade, the astonishing growth of the Word-Wide Web (number of users, pages, servers, on-line services, etc.) made it clear that it would soon become a huge repository of data of interest for a variety of application domains. However, the same features that have made the Web so useful and popular also impose important restrictions on the way the data it contains can be manipulated. Particularly, in the traditional Web scenario, there is a inherent difficulty in gaining access to data items which are implicitly present in Web pages but are not readily available.

This inherent unstructured characteristic of the data on the Web is largely due to the nature of its objects. Text documents, such as on-line books, newspaper articles, scientific articles, technical brochures, etc., along with binary objects, such as images, video, and sound streams, are inherently unstructured and their content can hardly be subject to some form of data processing. For accessing and retrieving such unstructured objects, a variety of information retrieval techniques [Baeza-Yates and Ribeiro-Neto, 1999] has been successfully applied.

However, a large portion of the Web is composed of pages that can be regarded as "data containers" in the sense that they implicitly contain data that can be identified, extracted, and manipulated independently. Examples of such pages are found in Web sites such as bookstores, electronic catalogs, travel agencies, and classified advertisements. A common feature of such pages is that their data usually presents an inherent structure that, although not explicitly described, can be recognized by a user looking at them through a browser, because of visual "clues" such as colors, fonts, bullets, and indentations provided by the page's author to help the user examine their contents. These pages are said to be *data-rich Web pages* [Embley *et al.*, 1999a] and Web sites and on-line services containing them or producing them dynamically are said to be *data-rich Web sources*.

For such pages, besides the problems of locating and retrieving them, a new problem arises, namely, manipulating adequately the implicit data they contain. In fact, it is an interesting paradox, that in spite of being publicly and readily available, implicit Web data can hardly be properly queried or manipulated as done, for instance, in traditional databases.

To appreciate the relevance of this problem, consider the data implicitly present in the

Web pages generated as a result of a query posed to the main page of the Amazon Web site. Figure 1.1 shows an example of such a page, for the Brazilian composer Antonio Carlos Jobim.


Figure 1.1: Example of page from the Amazon Web site.

Gaining access to the implicit data available in these pages would allow us to answer complex queries such as "What items from the *Popular Music* store also appear in the *Auctions* store", monitoring the Web site during a period of time looking for a particular item, or publishing a new Web page with the ten cheapest used items (available at the *zShops* store) by a given artist.

The importance of accessing implicit data in Web pages is further stressed by the necessity of enabling their processing by automated tools on the Web. In current days, the Web has established itself as a platform for data-intensive applications. Indeed, major software applications have become (or are in the process of becoming) Web-based. Further, Web-based interfaces have been used as a viable alternative for connecting modern software tools to legacy systems, with the purpose of complementing the functionality provided by the latter ones. As a consequence, there is an increasing demand for Web contents amenable to automatic processing by application programs (or automated tools), developed for the corporate world. According to the *Semantic Web Activity Statement* [Miller, 2002], "The Web can reach its full potential if it becomes a place where data can be shared and processed by automated tools as well as by people." The most evident symptom of this trend is the

attention that both the industry and academia are devoting to XML technology [Bray *et al.*, 2002].

However, despite attempts to define standards such as XML [Bray *et al.*, 2002] and RDF [Miller *et al.*, 2002] to provide some form of structure to the Web, most of the Web is still composed of HTML pages, either static or automatically generated. It is worth noticing that the spreading of such standards does not provide a trivial solution to the problem of manipulating existing Web data, since the volume of HTML pages currently available is enormous and is still increasing. Moreover, most of these standards are meant to be deployed in business-to-business scenarios, which suggests that most of the Web will continue to be composed of HTML pages.

A common approach to gaining access to implicit Web data is to build specialized programs called *wrappers* that extract the data of interest available in data-rich Web pages and represent this data in a suitable format such as relational tables or XML. Once available in a such a format, application programs can adequately process the data according to their specific needs.

The traditional approach to developing wrappers until recently was programming them using general purpose languages such as *Perl* and *Java*. Developing wrappers manually has many well known shortcomings, mainly due to the difficulty in writing and maintaining them. Nowadays, many tools have been proposed to better address the issue of generating wrappers for Web data extraction [Adelberg, 1998; Arocena and Mendelzon, 1998; Califf and Mooney, 1999; Crescenzi and Mecca, 1998; Crescenzi *et al.*, 2001; Embley *et al.*, 1999a; Freitag, 2000; Hammer *et al.*, 1997; Hsu and Dung, 1998; Kushmerick, 2000; Liu *et al.*, 2000; Muslea *et al.*, 2001; Laender *et al.*, 2002b; Sahuguet and Azavant, 2001]. Such tools are based on several distinct techniques [Laender *et al.*, 2002c] such as declarative languages [Arocena and Mendelzon, 1998; Crescenzi and Mecca, 1998; Hammer *et al.*, 1997], HTML structure analysis [Crescenzi *et al.*, 2001; Liu *et al.*, 2000; Sahuguet and Azavant, 2001], natural language processing [Freitag, 2000; Muslea *et al.*, 2001; Soderland, 1999], machine-learning [Califf and Mooney, 1999; Hsu and Dung, 1998; Kushmerick, 2000], data modeling [Adelberg, 1998; Laender *et al.*, 2002b], and ontologies [Embley *et al.*, 1999a].

The development of wrappers for Web data sources presents a number of interesting challenges, mainly due to the fact that data is organized to be accessed by the final user and not interpreted by programs. In most cases, data items (or objects) of interest (e.g., artist names, prices, time references, etc.) appear mixed inside the text of the page along with markup tags, hyperlinks, in-line code, and other uninteresting strings. These data are distinguishable only by their positions or appearances on the page. Further, the structure of the data is implicit and only suggested by presentation features. This structure is often loose, with the possibility that two similar items (e.g., data on two distinct books) present structural variations between them. This means that structural variations on the data can occur and should be tolerated and treated accordingly. In the recent literature, data presenting implicit and irregular structure, such as typical Web data, has been termed *semistructured* [Abiteboul, 1997; Buneman, 1997].

For instance, consider the sample page of Figure 1.1. Observe that if we are only interested in obtaining data on products available in each store, the wrapper must be able

to recognize this data while ignoring the remaining strings on the page. Further, notice that the information about items from each Amazon store is distinct. For instance, for the Store *Popular Music*, the information consists of Item, By, and Format, whereas for the Store *Auctions*, the information consists of Item, Bid, and Time. Thus, the page presented in Figure 1.1 can be considered as a good example of a data-rich Web page containing implicit semistructured data.

Besides the problems described above, the design and implementation of methods or tools for wrapper generation have to take into account two additional requirements. First, the wrappers they generate must be general enough to correctly extract data from a set of pages considered as *similar*. Here, we use the term *similar* in a very empirical sense, meaning pages provided by a same site or Web service, such as pages of the same Web bookstore. For instance, a useful wrapper is expected to work not only for the page of Figure 1.1, but also for all answer pages returned from the same query interface, as long as the overall structure and presentation features remain the same. Second, they must generate wrappers that are highly accurate and robust, while demanding as little effort as possible from the wrapper developers. In practice, as discussed in [Laender *et al.*, 2002c], this imposes an important trade-off between the degree of automation of a tool or method and the flexibility of the wrappers generated by it.

The problem of wrapper generation can be represented as follows. Given a Web data source $S$ containing a set of pages $T$, determine a mapping $w$ that is capable of populating a repository $R$ with a set $O$ of objects (data items) extracted from the pages in $T$. The mapping $w$ is, in general, a set of rules or text patterns used to recognize (among other uninteresting pieces of text) attribute values for objects of interest, associating an appropriate semantics to them. Based on this definition, we can say that a wrapper is an implementation of the mapping $w$.

Recent work in the literature proposes semiautomatic approaches for the generation of wrappers that derive the mapping $w$ from a given set of examples of the objects to be extracted. These approaches are called *example-based*. According to these approaches, given a set $E \subset O$ of example objects, taken from a subset $T_0 \subset T$ of the pages of the source $S$, a wrapper generation procedure $g$ generates the mapping $w$. That is, $g(E, T_0) = w$. A desirable feature of example-based wrapper generation approaches is that useful wrappers can be generated with as few examples as possible, i.e., $|E| << |O|$.

The main subject of the work presented here is the development of example-based strategies for the generation of wrappers for extracting semistructured data from data-rich Web sources. These strategies were conceived, designed and implemented having as a framework a particular example-based approach for Web data extraction, which is described next.

## 1.1 The DEByE Approach

The work we developed is based on an example-based approach for generating wrappers, which we call *DEByE (Data Extraction By Example)*[1]. Within the DEByE approach, a user or database designer specifies examples of the objects to be extracted by identifying pieces of useful data existing in a sample page $S$ and structuring these pieces of data in a suitable structure. From these example objects, a mapping can be generated for populating a repository with objects extracted from $S$, or from other pages similar to $S$. The objects extracted will have the same structure as the examples provided by the user.

Based on this data extraction approach, a tool, also called *DEByE* [Laender *et al.*, 2002b], was implemented. To allow a convenient specification of examples, the DEByE tool represents the structure of the data through nested tables [Makinouchi, 1977]. Nested tables are interesting because they are simple, intuitive, and expressive enough to represent the semistructured data normally present in Web pages [Laender *et al.*, 2000].

From the examples provided by the user through nested tables, a mapping, or wrapper, is generated in the form of a *tabular grammar*. Tabular grammars are context-free grammars whose productions have specific formats that lead to parse trees that can be directly mapped to nested tables. In a tabular grammar, we distinguish two sets of productions with distinct roles. The first is the set of *terminal* productions, that drive the lexical analysis of the target Web pages. These productions are used to recognize atomic values in a page (e.g., the title of a book, the name of an artist, etc.). The second is the set of *non-terminal* or *structural* productions, that parse the result of the lexical analysis and "assemble" complex objects from the atomic values derived by the terminal productions. Generated tabular grammars are then used to drive data extraction strategies based on information retrieval techniques which are very effective with various Web sources, as we demonstrated through experimentation.

In what follows, we will further detail the DEByE tool. Figure 1.2 illustrates the operation of this tool. The two modules, called *Graphical User Interface* (*GUI*) and *Extractor*, comprise the DEByE tool.

The GUI module provides the user with an interface to assemble example objects. Figure 1.3 presents a snapshot of the GUI during the specification of examples for the page of Figure 1.1. In this figure, the structure of the nested table in the lower half of the screen is constructed dynamically by the user during the specification of examples. For accomplishing this, the GUI provides built-in operations for manipulating rows, columns and nesting levels. The values appearing inside table cells are taken from a sample page chosen by the user, as show in the upper half of the screen. This is accomplished by using copy and paste operations, also provided by the GUI. In the nested table built by the user, each row is considered as a distinct example of an object to be extracted. In this particular case, three examples are given, each one corresponding to a store in which data about products have a distinct structure.

[1]This name is an homage to Moshé Zloof, creator of QBE [Zloof, 1977], who suggested the paradigm we use to specify the data to be extracted from Web pages.

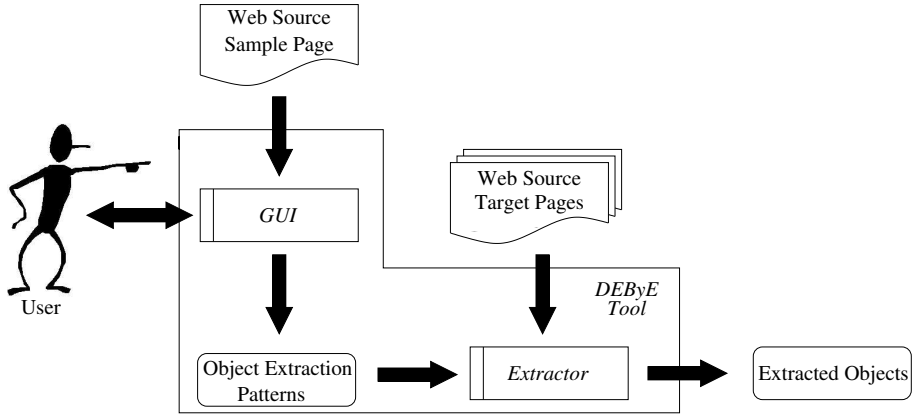



Figure 1.2: Modules of the DEByE tool and their role in data extraction.

Once the user has finished the specification of the examples, the wrapper generation process begins. For this, in the DEByE tool, the user simply selects the "Build Wrapper" operation. The assembled objects are then used to generate what we call an *object extraction pattern (oe-pattern)*, which corresponds to a tabular grammar. The generated oe-pattern can then be fed to the Extractor module, which applies it to extract new data from new pages, similar to the sample page, according to examples provided. The set of *extracted objects* is encoded in an XML-based format which can be easily converted to other formats and manipulated according to specific application needs.

## 1.2 Main Contributions

In the present work, we have achieved a number of results concerning the problem of generating wrappers for extracting semistructured data implicitly present in data-rich Web pages, having the DEByE approach [Laender *et al.*, 2002b] as a framework.

We have developed strategies for generating tabular grammars from a set of example objects provided by a user from a sample page. This includes: (1) the generation of terminal productions from single values identified by the user as belonging to a specific domain (e.g., an item description, a price, etc.) and (2) the generation of structuring productions by capturing the structure of the objects to be extracted from the nested tables assembled by the users. Both strategies were implemented in the GUI module of the DEByE tool and had their effectiveness experimentally verified.

We have also developed two strategies, called *top-down* [Ribeiro-Neto *et al.*, 1999a] and *bottom-up* [Ribeiro-Neto *et al.*, 1999b], that use the generated oe-patterns as guides to extract data from pages given as input. Both strategies can be regarded as specialized parsing algorithms that take advantage of the particularities of tabular grammars. Both strategies constitute alternatives for implementing the Extractor module of the DEByE tool. Having such specialized parsing algorithms is important because, as we shall discuss later, semistructured Web data often present an implicitly nested structure subject

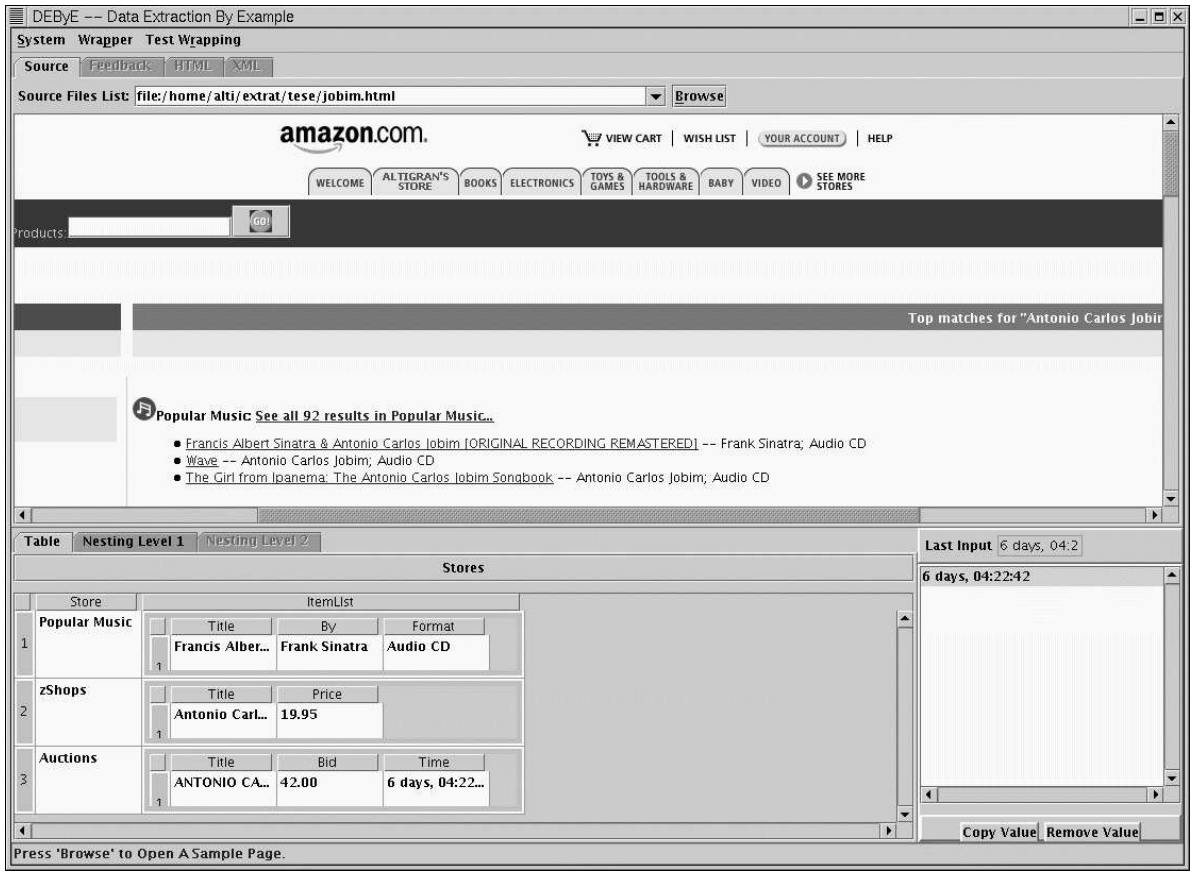



Figure 1.3: Specification of example objects with distinct structures with the DEByE tool.

to several kinds of variations. This leads to complex tabular grammars. Through experimentation over several Web sources, we show that the *bottom-up* strategy is very effective in such cases, making the data extraction process feasible for practical purposes. Further, an algorithm was developed to suggest a plausible nested tabular structure for the objects without relying on nested tables assembled by the user. This algorithm, called *Hot Cycles*, is targeted to applications where users are not available to design a data structure of their preferences. Although not integrated into the DEByE tool, the Hot Cycles algorithm is entirely based on the framework of the DEByE approach. Its effectiveness was also verified through experiments.

## 1.3 Further Contributions

As we mentioned earlier, the ultimate motivation for the work we have carried out on the DEByE approach was helping in important data management tasks such as storing, querying and integrating Web data. Thus, to verify the application of our work for such tasks, we developed a number of Web data management methods and tools based on the

DEByE approach [Arantes *et al.*, 2001a; 2001b; Evangelista-Filha *et al.*, 2001; Magalhães *et al.*, 2001a; 2001b; da Silva *et al.*, 2002]. An integrated perspective of these methods and tools in the context of Web data management environment is presented in [Laender *et al.*, 2002a]. In what follows, we briefly summarize such a work.

In [Laender *et al.*, 2000] and [da Silva *et al.*, 2002], we have shown that one of the advantages of using nested tables for representing the extracted data is that they allow us to extend well known operations for nested tables to deal with internal variations, as defined by the DEByE approach. In particular, we have implemented query operations over our extended form of nested tables in a graphical query interface suitable for semistructured Web data, which we call *QSByE (Querying Semistructured data By Example)* [Evangelista-Filha *et al.*, 2001; da Silva *et al.*, 2002]. QSByE combines features of QBE (*Query By Example*) with typical features of query languages for semistructured data. In particular, QSByE provides the structure of the data as a nested table "skeleton" so that users do not have to uncover the structure of the data by themselves.

Similarly, the underlying tabular structure of the data we manipulate simplifies the task of storing it into relational databases. Relational databases have been explored as an alternative to store semistructured data because they can efficiently manage huge volumes of data. Therefore, taking advantage of our underlying data model, we implemented a mechanism for storing and retrieving semistructured data in relational databases, which we call the *DEByE Data Storage Manager* [Magalhães *et al.*, 2001a; 2001b].

Another interesting benefit brought by the DEByE approach is that it assigns the semantics implicitly suggested by the user when providing examples to the data extract from a Web source. In [Arantes *et al.*, 2001a; 2001b], we have shown how to take advantage of this feature for the task of integrating data from distinct Web sources. These papers describe the *WebView* tool, which is used to build Web views composed of data extracted from several related Web data sources. In the DEByE context, a Web view can be simply regarded as a nested table that results from the integration of nested tables storing data extracted from target data sources.

Within the DEByE approach, the usual way for providing examples is through the DEByE GUI. However, for dealing with situations in which the user is not available for using the GUI, we have also investigated how example objects can be automatically provided. We have developed a method for bootstrapping the example-based data extraction process [Golgher *et al.*, 2001]. Given a sample page, this method automatically identifies in it values that can be used as examples. For this, it needs a data repository containing objects from the same domain of the sample Web page. This method allowed us to implement an automatic *example generator*, which can be applied to make the extractor resilient (i.e., immune to changes in the formatting features of the source pages) and adaptive (i.e., capable of working with pages from distinct sources belonging to a same application domain).

Finally, an issue not directly addressed within the DEByE approach is how to automatically obtain the Web pages from where data will be extracted. Although this may be seen as a problem that is orthogonal to what we focus in our work, we have also designed and implemented a tool for assisting the user in the tasks of generating agents for collecting Web pages containing data of interest, possibly produced as results of form

submission (i.e., dynamic pages). This tool, called *ASByE* [Golgher *et al.*, 2000a; 2000b; Arantes *et al.*, 2001a] is very useful in practical situations in which the target pages are produced dynamically or exist in large volumes in the Web sources.

## 1.4 Organization of the Dissertation

This dissertation is organized as follows. The following section of this chapter discusses work related to ours that propose alternative approaches for wrapper generation. Chapter 2 presents basic concepts and the notation used throughout the text for representing semistructured data. Chapter 3, formalizes the concept of tabular grammar and shows how such grammars are generated within the DEByE approach. Chapter 4 discusses strategies for recognizing and extracting semistructured data by parsing Web pages using tabular grammars. Chapter 5 shows the results of the experiments we have performed using this approach. Finally, Chapter 6 contains our conclusions and discusses future work.

## 1.5 Related Work

In the recent literature, many approaches have been proposed for dealing with problems related to Web data management. Most of them deploy graph-based formalisms for representing the structure and the contents of Web sites and pages. This has yielded effective and elegant solutions to the problems of extracting, modeling, querying and integrating Web data. A good survey on such approaches is presented in [Florescu *et al.*, 1998]. This section presents a brief review of Web data management work related to ours.

To represent semistructured data, several data models have been proposed [Buneman *et al.*, 1996; 1999; Papakonstantinou *et al.*, 1995]. These models are, in general, based on labelled directed graphs that capture the irregular structure inherent in such data. OEM (Object Exchange Model) is an object-based model adopted by the TSIMMIS project [Papakonstantinou *et al.*, 1995]. An OEM object can be either atomic or complex. The value of a complex OEM object is a set of object references to its components; these references can be cyclic. The data model proposed in [Buneman *et al.*, 1996] for the UnQL query language is quite similar to OEM. The difference is that the UnQL data model lacks the notion of an object, describing data by a means of a set of trees whose leaf nodes have the actual instances associated with them. The model presented in [Buneman *et al.*, 1999] also represents data as a directed labelled graph in which each node corresponds to an object, but labels the edges emanating from any node (that describes data) distinctly. In our work, we adopt a very simple data model in which complex objects can be represented as nested tables. Our model is based on ideas described in [Abiteboul *et al.*, 1995; Libkin, 1991] and can be seen as an extension of the relational model.

For the specific task of Web data extraction, which is the focus of our work, several tools based on a variety of techniques have been discussed in the literature. A survey on this subject is presented in [Laender *et al.*, 2002c].

One of the first initiatives for addressing the problem of wrapper generation was the definition of languages specially designed to assist users in developing wrappers. Such languages were proposed as alternatives to general purpose languages such as Perl and Java, which were prevalent at that time for this task. Some of the best known tools that adopt this approach are Minerva [Crescenzi and Mecca, 1998], TSIMMIS [Hammer *et al.*, 1997], Web-OQL [Arocena and Mendelzon, 1998], FLORID [Ludäscher *et al.*, 1998], and Jedi [Huck *et al.*, 1998]. Although such languages provide an effective approach for wrapper generation, their main drawback is that they require manual wrapper development. Due to such a limitation, efforts have been made to automate the wrapper generation process. In particular, approaches based on examples (such as DEByE) have proved to be very effective for the task of wrapper generation. This is because they do not require any previous knowledge of target Web pages. Additionally, any structural changes in those pages can be easily accommodated by providing new examples.

Many recent research efforts propose the use of machine-learning techniques to semi-automatically induce wrappers [Hsu and Dung, 1998; Kushmerick, 2000; Muslea *et al.*, 1999]. In general, these approaches consist of using training examples to generate automata that recognize instances in contexts similar to the ones of the given examples. The approach proposed in [Kushmerick, 2000] and adopted in the WEIN system relies, like ours, on examples from the source to be wrapped. The main drawbacks of this pioneering work are: (1) it does not deal with missing or out-of-order components and (2) although it identifies the need for extraction of complex objects present in nested structures, the solution provided is computationally intractable and has not been implemented. These two very important features of semistructured data extraction are addressed in SoftMealy [Hsu and Dung, 1998] and Stalker [Muslea *et al.*, 2001]. Both systems also generate wrappers, generalizing given examples through machine-learning techniques, and are very effective in wrapping several types of Web pages. The main problem with SoftMealy is that every possible absence of a component and every different ordering of the components must be represented beforehand by an example. Stalker [Muslea *et al.*, 1999] can deal with such variations in a much more flexible way since each object component is extracted independently through a top-down decomposition procedure. A common feature to all the approaches above is that the extraction process relies on knowledge of the structure of the source document (e.g., an HTML page). In WEIN and SoftMealy, for example, pages are assumed to have a defined structure (e.g., a head, then a body with a set of tuples, and then a tail) that must be flat. This prevents the exclusive extraction of only the objects (or sub-objects) of interest and might generate extraction difficulties if unwanted text portions (such as advertisements) occur between tuples or tuple components in the body. In Stalker, the extraction of nested objects is possible but the approach also relies on a previous description of the entire source page.

Besides wrapper induction, there are several other approaches for learning extraction patterns that are more suitable for extracting data from semistructured text such as newspaper classified advertisements, seminar announcements, and job posting, which present grammatical elements, commonly used in a telegraphic style. In general, these approaches use techniques typical of Natural Language Processing (i.e., semantic class, part-of-speech

tagging, etc.) sometimes combined with the recognition of syntactic elements (e.g., delimiters). This is the case of Rapier [Califf and Mooney, 1999] and SRV [Freitag, 2000]. WHISK [Soderland, 1999] goes beyond and addresses a large spectrum of types of documents ranging from rigidly formatted to free text. For formatted text, this system has a behavior that is closer to wrapper induction systems like WEIN [Kushmerick, 2000].

An ontology-based approach to extracting data from Web sources is presented in [Embley *et al.*, 1999a]. The approach uses a semantic data model to provide an ontology that describes the data of interest, including relationships, lexical appearances, and context keywords. By parsing this ontology, a relational database schema and a constant/keyword recognizer are automatically generated, which are then used to extract the data that will populate the database. Prior to the application of the ontology, the approach requires the application of an automatic procedure to extract chunks of text containing data "items" (or records) of interest [Embley *et al.*, 1999b]. Then, the extraction process proceeds from the set of records extracted. Although this approach also requires the user to provide a conceptual description of the data to be extracted, it is radically distinct from ours with respect to the extraction strategy adopted. While we rely on the textual context surrounding the data of interest, the ontology-based approach relies mainly on the expected contents of the pages, according to what was anticipated by the pre-specified ontology. Further, this approach requires a specialist to build the ontology using a notation specially designed, whereas the DEByE approach provides a visual metaphor that helps users specify their view of the data. On the other hand, if the ontology is representative enough, the extraction is fully automated. Furthermore, wrappers generated according to such an approach are inherently resilient (i.e., they continue to work properly even if the formatting features of the source pages change) and adaptable (i.e., they work for pages from many distinct sources belonging to a same application domain). Indeed, these features are unique to this approach.

NoDoSE [Adelberg, 1998] is a tool that, like ours, adopts an user-driven approach for data extraction. This tool provides a graphical interface which the user uses to decompose a given document (e.g., a Web page) into a hierarchy that describes its structure. Additional documents of the same type are then provided to the tool and automatically parsed. If tuning is required (which frequently is the case), the user must inspect the results (using the interface), modify the hierarchy that describes the document, and use it to parse the pages again. The process is complete when all of the documents have been successfully parsed. NoDoSE requires the user to specify the structure of the whole document (i.e., the set of pages provided as input) in a top-down fashion which, in some cases, might be hard to do. Despite these drawbacks, the approach is effective for a large class of textual documents once the parsing is successful. The most noticeable distinction between NoDoSE and DEByE is the way examples are provided by the user to generate the extraction patterns. While in NoDoSE users must decompose the whole document marking regions in the entire document body, in DEByE users mark only atomic values and organize them according to their perception of the implicit structure of the objects being extracted. Besides, this assembling of objects is supported by a quite intuitive and simple metaphor, namely nested tables. As a result, the users are completely shielded from the specific formatting features

of the page they are dealing with.

Another interesting approach for user-driven Web data extraction is the one adopted by XWRAP [Liu *et al.*, 2000]. In this tool, the user is presented with a syntax tree that describes the HTML structure of a page. The nodes of this tree correspond to HTML tags (e.g., `<TABLE>`, `<TR>`, etc.). By browsing this tree, the user selects the portions of the page that are of interest, and for each portion the tool applies a special set of pre-defined extraction rules. For instance, there are rules for tables, list, etc. The extracted data is output in XML, but the tags used in the final document are also derived from the source page, under the assumption that the page contains text that can be used as metadata. The major drawback we see in this tool is the explicit use of HTML syntax and structure by the end user. This is a remarkable difference when compared with our tool in which the page formatting features are completely transparent to the user.

A recent tool that further explores the inherent features of HTML documents to automatically generate wrappers is RoadRunner [Crescenzi *et al.*, 2001]. It works by comparing the HTML structure of two (or more) given sample pages belonging to a same "page class", generating as a result a schema for the data contained in the pages. From this schema, a grammar is inferred which is capable of recognizing instances of the attributes identified for this schema in the sample pages (or in pages of the same "class"). To accurately capture all possible structural variations occurring on pages of a same page class, it is possible to provide more than two sample pages. The extraction process is based on an algorithm that compares the tag structure of the sample pages and generates regular expressions that handle structural mismatches found between the two structures. In this way, the algorithm discovers structural features such as tuples, lists, and variations. It should be noted that the process is fully automatic and no user intervention is required, a feature that is unique to RoadRunner. The DEByE approach certainly does not provide the same level of automation as RoadRunner. However, our approach has a broader range of applications since, contrary to RoadRunner, it does not rely on specific features of HTML to uncover the structure of the objects to be extracted. Furthermore, the fact that we require the user to select examples of the objects to be extracted makes our approach applicable to Web pages that feature several uninteresting pieces of data, among the interesting ones. In cases like that, fully automated tools tend to make many mistakes, in the sense that they can extracted several unwanted data.

In fact, XWRAP and RoadRunner are examples of tools that rely on inherent structural features of HTML documents for accomplishing data extraction. Before performing the extraction process, these tools turn the document into a parsing tree, a representation that reflects its HTML tag hierarchy. Afterwards, extraction rules are generated either semi-automatically or automatically and applied to the tree. Other representative tools based on such an approach are W4F [Sahuguet and Azavant, 2001] and Lixto [Baumgartner *et al.*, 2001].

# Chapter 2

# Data Modeling Concepts for Representing Semistructured Web Data

In this chapter, we discuss the data modeling concepts we adopt to represent the data of interest present in data-rich Web pages. Such concepts play an important role in the work developed, since they drive all the techniques for Web data extraction we present latter. These modeling concepts rely on the assumption that such pages can be seen as collections of *complex objects* which have an inherent implicit structure. In many cases, these objects are composed of sub-objects, that themselves also have an implicit structure yielding a hierarchy of objects.

Consider, for instance, the page from the Amazon Web site shown in Figure 1.1. There is an inherent structure to the text on this page. We are able to identify distinct portions of data that correspond to five "stores" and their products. Each one of these portions can be regarded as a distinct implicit *object*. For each of these objects, we can distinguish the name of the store and a corresponding list of items available in it. For the items in these lists, we can identify information on item descriptions, artists, format, prices, etc. Thus, there is an inherent structure associated with the objects implicitly present in the Web page of Figure 1.1. Such structure has not been declared anywhere but is clearly identifiable.

To illustrate, Figure 2.1 presents the hierarchical structure for the objects corresponding to two stores (*Popular Music* and *Auctions*), according to a graphical representation similar to OEM [Papakonstantinou *et al.*, 1995]. Observe that the way the objects are represented in Figure 2.1 reflects a particular interpretation of the data implicitly available in the page of Figure 1.1. In particular: (1) among all other strings in the page of Figure 1.1, only those considered as components of Store objects are represented, and (2) these strings were organized according to a specific structure.

From this observation, we can see that some data modeling paradigm is necessary for representing the implicit semistructured data present in data-rich Web pages. For this, we could have adopted a general semistructured data model such as OEM or even

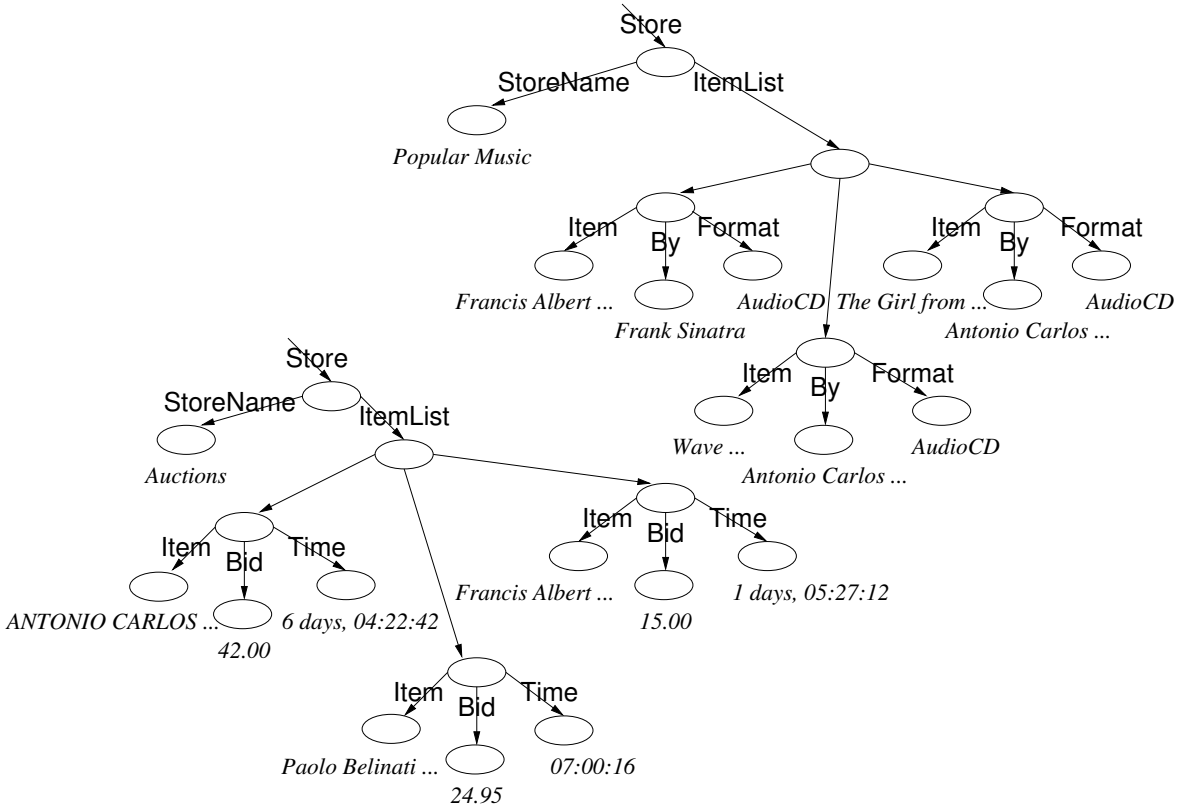



Figure 2.1: Hierarchical structure for two of the implicit objects in Figure 1.1.

XML, which has been largely used as a data model for representing semistructured data in general [Abiteboul *et al.*, 1999].

However, in the DEByE approach, we use an extension of *nested tables* that allow internal structural variations for representing semistructured Web data. As we show latter in this chapter, such tables provide a simple and intuitive representation, close to record-based database representation, but that also naturally accommodates hierarchical objects presenting variations and irregularities typical of semistructured data. The main distinction between the nested tables as we use them and regular nested tables is that we allow a column to have two or more distinct substructures. An example of this solution is presented in the nested table in Figure 2.2. This table illustrates how the semistructured objects implicitly present in the page of Figure 1.1 can be represented using our extended form of nested tables. Note that the internal structures of the objects in the column ItemList are distinct for each of the rows.

The main motivation for the use of nested tables within the DEByE approach comes from the DEByE tool. As illustrated in Figure 1.3, the use of nested tables allowed us to build an interface that is both simple and intuitive, while expressive enough to allow users to provide examples of typical semistructured objects found in data-rich Web pages. In

**Store**

| # | StoreName | ItemList |
|---|---|---|

**1 — Popular Music**

| # | Item | By | Format |
|---|---|---|---|
| 1 | Francis Albert Sinatra & Antonio Carlos Jobim [ORIGINAL RECORDING REMASTERED] | Frank Sinatra | Audio CD |
| 2 | Wave | Antonio Carlos Jobim | Audio CD |
| 3 | The Girl from Ipanema: The Antonio Carlos Jobim Songbook | Antonio Carlos Jobim | Audio CD |

**2 — Video**

| # | Item | By | Format | Year |
|---|---|---|---|---|
| 1 | Antonio Carlos Jobim: An All Star Tribute | Herbie Hancock | VHS | 1995 |

**3 — Books**

| # | Item | Format |
|---|---|---|
| 1 | Antonio Carlos Jobim for Guita | - |
| 2 | Antonio Carlos Jobim Anthology/00312477 | - |
| 3 | Antonio Carlos Jobim Piano Solos | Paperback |

**4 — zShops**

| # | Item | Price |
|---|---|---|
| 1 | Antonio Carlos Jobim: An All-Star Tribute | 19.95 |
| 2 | Antonio Carlos Jobim - Guitar Sheet Music | 16.95 |
| 3 | lp - SINATRA, FRANK & JOBIM - Francis Albert Sinatra & Antonio Carlos Jobim | 15.00 |

**5 — Auctions**

| # | Item | Bid | Time |
|---|---|---|---|
| 1 | ANTONIO CARLOS JOBIM. The Wonderful World Of Antonio Carlos Jobim RECORD LP M- | 42.00 | Ends in 6 days, 04:22:42 |
| 2 | "Frances Albert Sinatra & Antonio Carlos Jobim" Record Album | 15.00 | Ends in 1 days, 05:27:12 |
| 3 | Paulo Bellinati Plays the Music of Antonio Carlos Jobim Video | 24.95 | Ends in 07:00:16 |

Figure 2.2: Example of a nested table allowing internal structural variations.

fact, even early experiments with the interface have demonstrated its effectiveness for the process of example specification [Silva, 1999; Laender *et al.*, 2000].

It is worth mentioning that, despite the relative simplicity in dealing with semistructured data in the form of nested tables, it is easy to see that such a representation is not as expressive as general semistructured data models or XML. We cannot, for example, have different structures at the top level. Thus, we sacrifice some flexibility to greatly increased simplicity. However, in our work, we are mainly concerned with representing data from data-rich Web pages, like the one in Figure 1.1. Examples of such pages are found in Web sites such as bookstores, electronic catalogs, travel agencies, and classified ads and include pages composed of data whose overall structure is naturally hierarchical, but exhibits a modest degree of variation. In particular, we are interested in manipulating data extracted from these kinds of Web pages by DEByE. For such a task, nested tables with structural variations have proved to be a suitable alternative.

In the remainder of this chapter, we first formalize the extended form of nested table we use in our work. Then, we discuss the expressiveness of nested tables as a data model for representing semistructured Web data by briefly comparing them with typical semistructured data models. Next, we describe how a table scheme can be obtained from a given table instance, an important issue regarding our data extraction approach. Finally, we show how we implement our modeling paradigm by means of XML, which is important for allowing the manipulation of the extracted data by applications in general.

## 2.1 Basic Concepts and Terminology

In this section, we formalize the data modeling concepts we adopt for representing semistructured Web data. These concepts are based on the notion of *nested table* [Makinouchi, 1977], augmented with the concept of *variant* [Libkin, 1991].

We begin by defining a *table scheme*.

**Definition 1** *A **table scheme** $\tau$ is defined using the notation*

$$\tau = (C_1 : [\tau_1^1;\ \ldots;\ \tau_1^{n_1}], C_2 : [\tau_2^1;\ \ldots;\ \tau_2^{n_2}], \ldots, C_m : [\tau_m^1;\ \ldots;\ \tau_m^{n_m}])$$

*where, $m \geq 2$, $n_k \geq 1$, $1 \leq k \leq m$. Further, $\tau_j^i$ denotes exactly one of the following: (i) an atomic value, represented by* atom*, (ii) a set of atomic values, represented by* {atom} *or (iii) a table scheme. Each $C_j$ is called a **column**, and, for the sake of simplifying the notation, if $n_j = 1$, we can use $C_j : \tau_j^1$ instead of $C_j : [\tau_j^j]$.*

Intuitively, a table scheme describes the structure of a kind of nested table in which a column $C_j$ may store "values" or objects with distinct structure in distinct tuples. The structures of the possible objects are given by the alternatives $\tau_j^1, \ldots, \tau_j^{n_j}$ which can be either atomic values, lists of atomic values, or other nested tables. Throughout the text, if $C_j$:atom, $C_j$ is called an ***attribute*** and, if $C_j$:{atom}, $C_j$ is called a ***list***.

Consider the page excerpt illustrated in Figure 1.1. The structure of the objects implicitly present can be described by the following table scheme $\tau$:

$$\tau = (\mathsf{StoreName} : \mathsf{atom}, \mathsf{ItemList} : [\tau_2^1;\ \tau_2^2;\ \tau_2^3]),$$

where

$$\tau_2^1 = (\mathsf{Item} : \mathsf{atom}, \mathsf{By} : \mathsf{atom};\ \mathsf{Format} : \mathsf{atom};\ \mathsf{Year} : \mathsf{atom}),$$
$$\tau_2^2 = (\mathsf{Item} : \mathsf{atom}, \mathsf{Price} : \mathsf{atom}),\ \text{and}$$
$$\tau_2^3 = (\mathsf{Item} : \mathsf{atom}, \mathsf{Bid} : \mathsf{atom}, \mathsf{Time} : \mathsf{atom}).$$

The nested table in Figure 2.2 is an instance of the table scheme $\tau$ defined above. In this table scheme, for the first level, two columns are defined: StoreName, which is an attribute, and ItemList with three distinct possible structures (nested tables), each one corresponding to a type of store in Figure 1.1. We now precisely define the notion of an instance of a table scheme.

**Definition 2** *Let $\tau = (C_1 : [\tau_1^1;\ \ldots;\ \tau_1^{n_1}],\ C_2 : [\tau_2^1;\ \ldots;\ \tau_2^{n_2}], \ldots, C_m : [\tau_m^1;\ \ldots;\ \tau_m^{n_m}])$, with $m \geq 2,\ n_k \geq 1, 1 \leq k \leq m$, be a table scheme. An **instance** $T$ of $\tau$, denoted by $T : \tau$, is a set*

$$T = \{\langle C_1 : v_1^1, C_2 : v_2^1, \ldots, C_m : v_m^1\rangle, \ldots, \langle C_1 : v_1^n, C_2 : v_2^n, \ldots, C_m : v_m^n\rangle\}, (n \geq 0),$$

*where $v_j^k$ is: (i) an atomic value, if $\tau_j^k =$* atom*, (ii) a list of atomic values, if $\tau_j^k =$* {atom}*, or (iii) an instance of $\tau_j^k$ that is a table scheme. An instance of a table scheme is referred to as a **table**.*

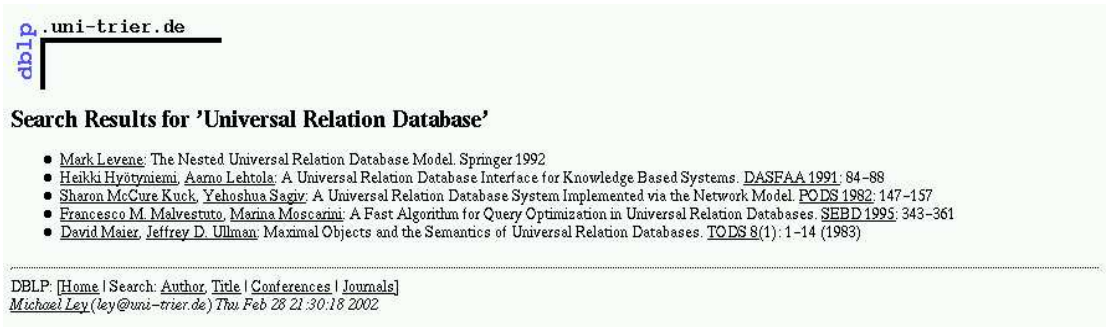
According to the notation introduced in Definition 2, a possible instance $S$ of our example table scheme $\tau$ is as follows:

$$S = \{\langle \mathsf{Store} : \text{“Popular Music”}, \mathsf{ItemList} : I_1\rangle, \ldots, \langle \mathsf{Store} : \text{“Auctions”}, \mathsf{ItemList} : I_5\rangle\}$$
$$I_1 = \{\langle \mathsf{Item} : \text{“Francis Albert...”}, \mathsf{By} : \text{“Frank Sinatra”}, \mathsf{Format} : \text{“Audio CD”}\rangle, \ldots\}$$
$$\vdots$$
$$I_5 = \{\langle \mathsf{Item} : \text{“ANTONIO CARLOS ...”}, \mathsf{Bid} : \text{“42.00”}, \mathsf{Time} : \text{“Ends in 6 days, 04:22:42”}\rangle, \ldots\}$$

Observe that the notation above incorporates structural information along with the data itself; thus we have a self describing representation for semistructured data. As a consequence, instead of using this notation, we could easily describe such data by means of XML, as we actually do in DEByE. In Section 2.4 we describe an XML implementation for our nested tables.

## 2.2 Expressiveness of Nested Tables for Representing Semistructured Web Data

In this section, we discuss the expressiveness of nested tables as a data model for representing semistructured Web data. In particular, we make a brief comparison between our nested tables and typical semistructured data models. For the discussion that follows, consider the Web page resulting from the query “Universal Relation Database” in the DBLP Web site[1], which is shown in Figure 2.3.


dblp.uni-trier.de

**Search Results for 'Universal Relation Database'**

- Mark Levene: The Nested Universal Relation Database Model. Springer 1992
- Heikki Hyötyniemi, Aarno Lehtola: A Universal Relation Database Interface for Knowledge Based Systems. DASFAA 1991: 84–88
- Sharon McCure Kuck, Yehoshua Sagiv: A Universal Relation Database System Implemented via the Network Model. PODS 1982: 147–157
- Francesco M. Malvestuto, Marina Moscarini: A Fast Algorithm for Query Optimization in Universal Relation Databases. SEBD 1995: 343–361
- David Maier, Jeffrey D. Ullman: Maximal Objects and the Semantics of Universal Relation Databases. TODS 8(1): 1–14 (1983)

DBLP: [Home | Search: Author, Title | Conferences | Journals]
*Michael Ley (ley@uni-trier.de) Thu Feb 28 21:30:18 2002*

Figure 2.3: A sample Web page from DBLP.

Figures 2.4 and 2.5 show the data extracted from this page organized into two distinct labelled trees according to OEM. In the following discussion, we refer to these trees as $M$ and $N$, respectively.

Trees $M$ and $N$ can be considered as semistructured databases and, intuitively, they are equivalent, since the relationship between atomic values is maintained. However, while in $M$ each Publication subtree is composed of distinct atomic components, in $N$ we introduce

[1] `http://www.informatik.uni-trier.de/~ley/db/indices/t-form.html`

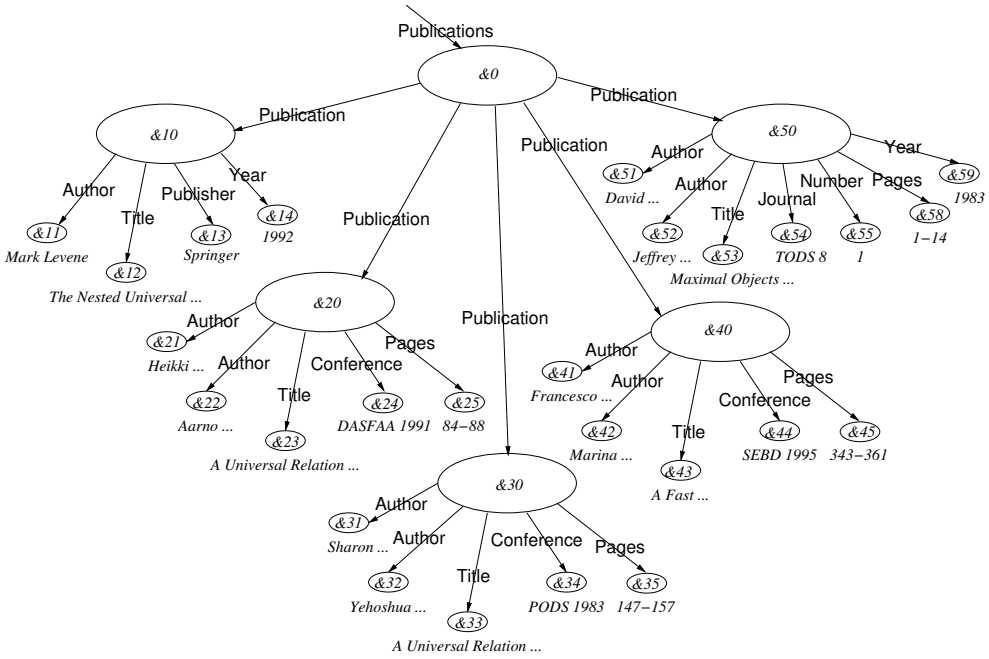



Figure 2.4: An OEM tree for the data in the page of Figure 2.3.

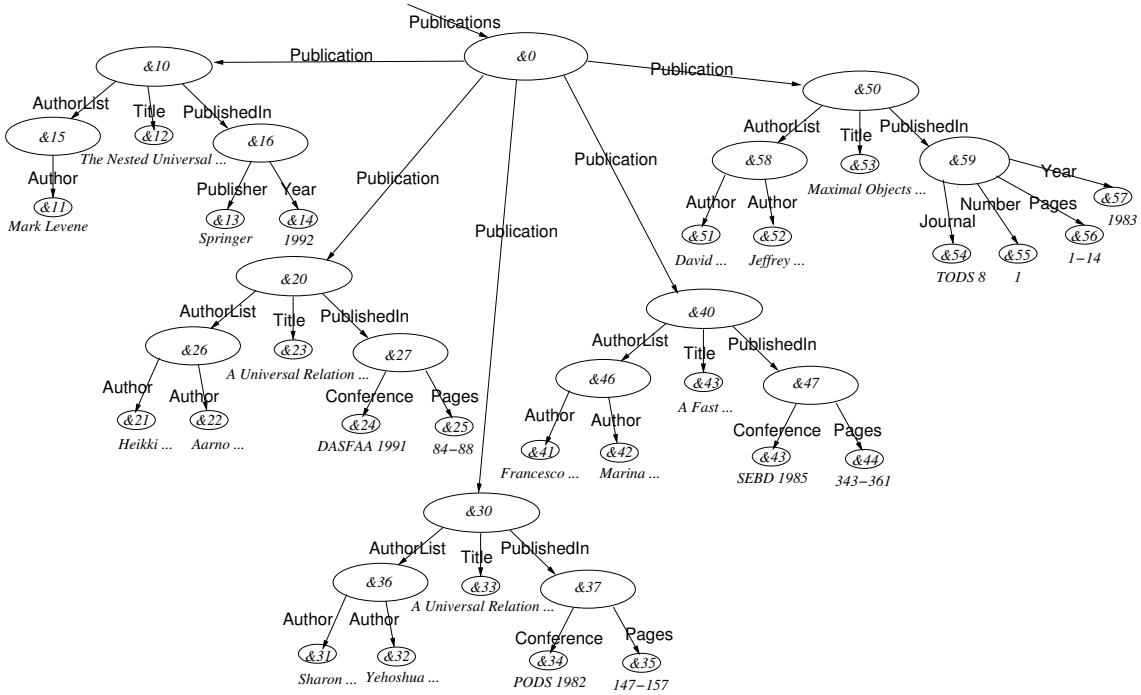



Figure 2.5: An alternative OEM tree for the data in the page of Figure 2.3.

two additional nodes (AuthorList and PublishedIn) in Publication sub-trees with the goal of making these sub-trees uniform in their first levels. This alternative representation preserves the semantics of the objects, but it is less concise than the first one. On the other hand, for our purposes, $N$ presents an important property: it can be directly mapped into a nested table, such as the one presented in Figure 2.6.

**Publication**

| # | AuthorList | Title | PublishedIn |
|---|---|---|---|
| 1 | Mark Levene | The Nested Universal Relation Database Mode | # \| Publisher \| Year<br>1 \| Springer \| 1992 |
| 2 | Heikki Hyötyniemi<br>Aarno Lehtola | A Universal Relation Database Interface for Knowledge Based Systems | # \| Conference \| Pages<br>1 \| DASFAA 1991 \| 84–88 |
| 3 | Sharon McCure Kuck<br>Yehoshua Sagiv | A Universal Relation Database System Implemented via the Network Model | # \| Conference \| Pages<br>1 \| PODS 1982 \| 147–157 |
| 4 | Francesco M. Malvestuto<br>Marina Moscarini | A Fast Algorithm for Query Optimization in Universal Relation Databases | # \| Conference \| Pages<br>1 \| SEBD 1995 \| 343–361 |
| 5 | David Maier<br>Jeffrey D. Ullman | Maximal Objects and the Semantics of Universal Relation Databases | # \| Journal \| Number \| Pages \| Year<br>1 \| TODS 8 \| 1 \| 1–14 \| 1983 |

Figure 2.6: Data from the DBLP page of Figure 2.3 organized into a nested table.

The table in Figure 2.6 makes explicit an interesting characteristic of nested tables for the representation of semistructured Web data. Traditionally, nestings have the role of representing in a single column complex objects, i.e., non-atomic values. In our approach, we "overload" this structural feature by using it to also accommodate structural variations. This is what happens for the column PublishedIn, in which rows 1 and 5 store tables that have a structure distinct from the structure of the tables stored in rows 2, 3 and 4. Notice that, for the case of this specific example, each row corresponds to a single publication. Thus, this representation cannot be considered precise, since all tables stored under PublishedIn will actually have one single tuple each. From this simple example, we can conclude that nested tables are indeed less expressive than OEM for representing semistructured data. However, in situations where typical Web data is to be represented, nested tables allowing variants constitute a viable representation alternative, since the semistructured data commonly found in data-rich Web pages are hierarchically organized and present a modest degree of variation that can be adequately handled by such tables.

To go further in this discussion, we now present a brief comparison with XML, which is currently the predominant formalism for representing Web data. We notice that most of the discussion we have presented so far in this section also applies to XML, since it is, essentially, a notation for representing labelled trees.

In Figure 2.7(a) we present a DTD that declares the structure of an XML document corresponding to the labelled tree $M$ of Figure 2.4. Similarly, in Figure 2.7(b) we present a DTD that declares the structure of an XML document corresponding to the labelled tree $N$ of Figure 2.5. Let us refer to these DTDs as $D_M$ and $D_N$, respectively.

Notice that $D_M$ and $D_N$ define XML documents that are equivalent in the same sense as trees $M$ and $N$ are. Considering that DTDs are indeed context-free grammars [Abiteboul

(a)

```
<!DOCTYPE dpub [
 <!ELEMENT Publications (Publication*)>
 <!ELEMENT Publication  (Author*, Title,((Publisher,Year)|
                        (Conference,Pages)|
                        (Journal,Number,Pages,Year))>
 <!ELEMENT Publisher    (#PCDATA)>
 <!ELEMENT Author       (#PCDATA)>
 <!ELEMENT Title        (#PCDATA)>
 <!ELEMENT Conference   (#PCDATA)>
 <!ELEMENT Journal      (#PCDATA)>
 <!ELEMENT Number       (#PCDATA)>
 <!ELEMENT Pages        (#PCDATA)>
 <!ELEMENT Year         (#PCDATA)>
]>
```

(b)

```
<!DOCTYPE dpub [
 <!ELEMENT Publications (Publication*)>
 <!ELEMENT Publication  (AuthorList,Title,PublishedIn)>
 <!ELEMENT Authorlist   (Author*)>
 <!ELEMENT PublishedIn  (PublishedIn1|PublishedIn2|PublishedIn3)>
 <!ELEMENT PublishedIn1 (Publisher,Year)>
 <!ELEMENT PublishedIn2 (Conference,Pages)>
 <!ELEMENT PublissheIn3 (Journal,Number,Pages,Year)>
 <!ELEMENT Title        (#PCDATA)>
 <!ELEMENT Author       (#PCDATA)>
 <!ELEMENT Publisher    (#PCDATA)>
 <!ELEMENT Year         (#PCDATA)>
 <!ELEMENT Conference   (#PCDATA)>
 <!ELEMENT Pages        (#PCDATA)>
 <!ELEMENT Journal      (#PCDATA)>
 <!ELEMENT Number       (#PCDATA)>
]>
```

Figure 2.7: Two DTDs for XML documents storing data extracted from the DBLP page of Figure 2.3.

*et al.*, 1999] and that XML documents (or OEM labelled trees) are derivations of such grammars, we can see $D_N$ as the grammar that results from including in $D_M$ a number of productions (or `ELEMENT` declarations) to ensure that the resulting documents or trees take a form similar to $N$ and, thus, can be mapped to nested tables. More precisely, such trees would be considered as tables, according to Definition 2.

Indeed, nested tables are less expressive than XML for representing Web data precisely because they can be described by a sub-class of DTDs such as $D_N$, which we refer to as *Tabular DTDs* or *TDTDs*. In TDTDs, `ELEMENT` declarations are restricted to some predefined forms that guarantee that XML documents (or labelled trees) correspond to nested tables. In particular, non-terminal `ELEMENT` declarations in TDTDs are restricted to be of one of the following forms.

- ***Aggregating*** (or ***tuple-generating***) declarations have the form

  $$\texttt{<!ELEMENT } X_0 \ (X_1 \dots X_n)\texttt{>}$$

  where $(n \geq 2)$, and $X_i \neq X_j$, for every $0 \leq i, j \leq n$ except $i = j$. Further, each $X_k$, $k = 1 \dots n$, must appear on the left-hand side of exactly one iterating or terminal declaration in the TDTD.

- ***Iterating*** (or ***list-generating***) declarations have the form

  $$\texttt{<!ELEMENT } X \ (Y*)\texttt{>}$$

  where $X \neq Y$. Further, $Y$ must appear on the left-hand side of exactly one aggregating, varying, or terminal declaration in the TDTD.

- ***Varying*** (or ***variant-generating***) declarations have the form

  $$\texttt{<!ELEMENT } X_0 \ (X_1 | \dots | X_n)\texttt{>}$$

  where $(n \geq 2)$, and $X_i \neq X_j$, for every $0 \leq i, j \leq n$ except $i = j$. Further, each $X_k$, $k = 1 \dots n$, must appear on the left-hand side of exactly one aggregating or iterating declaration in the TDTD.

It is easy to see that limiting the possible `ELEMENT` declarations as described above considerably restricts the possible derivations (i.e., labelled trees or XML documents) that can be generated. However, it must be observed that formats such as XML are intentionally non-restrictive, since they do not aim at any application in particular. Indeed, XML can be used to represent typical Web data, such as the data found in pages of Figures 1.1 and 2.3, but it is also flexible enough to describe, for instance, DNA sequences, communication protocols or stylesheets. In our work, we claim that, for representing data typically found in data-rich Web pages, it is possible to use nested tables as we define them, without compromising the accuracy of the representation. Indeed, despite their relative lack of expressiveness, nested tables are expressive enough to represent a vast collection of different data available in Web pages, such as those in Figures 1.1 and 2.3. As additional evidence, recent work on data extraction [Crescenzi *et al.*, 2001; Kushmerick, 2000; Muslea *et al.*, 2001] confirms that nested tables are an effective paradigm for describing Web data.

## 2.3 Deriving a Table Scheme from a Table Instance

A common task in semistructured data management is obtaining a scheme that is representative of a given set of instances. In the case of our work, this task is important because it is necessary to generalize the example table assembled by the user through the DEByE tool GUI. In this section, we describe how this is accomplished for the case of nested tables allowing variations.

We notice that there may be several possible table schemes that correspond to a given table instance. Thus, as described later, our strategy for obtaining a table scheme is biased towards the requirements of our data extraction approach.

Consider the example table assembled by the user through the DEByE tool GUI, such as the one presented in the screen shot of Figure 1.3. For convenience, Figure 2.8 shows the same table in detail. We may see that, in this figure, every row of the outermost table corresponds to an instance of a type $\tau$ defined as:

$$\tau = (\mathsf{StoreName} : \mathsf{atom}, \mathsf{ItemList} : [\tau_1;\ \tau_2;\ \tau_3])$$

where

$$\tau_1 = (\mathsf{Item} : \mathsf{atom}, \mathsf{By} : \mathsf{atom};\ \mathsf{Format} : \mathsf{atom};\ \mathsf{Year} : \mathsf{atom})$$

$$\tau_2 = (\mathsf{Item} : \mathsf{atom}, \mathsf{Price} : \mathsf{atom})$$

$$\tau_3 = (\mathsf{Item} : \mathsf{atom}, \mathsf{Bid} : \mathsf{atom}, \mathsf{Time} : \mathsf{atom})$$

Further, each of the rows in the internal tables correspond respectively to an instance of $\tau_1$,$\tau_2$ and $\tau_3$.

| | StoreName | ItemList | | | |
|---|---|---|---|---|---|
| 1 | Popular Music | Item | By | Format | Year |
| | | Francis Albert ... | Frank Sinatra | Audio CD | 1995 |
| 2 | zShops | Item | Price | | |
| | | Antonio Carlos... | 19.95 | | |
| 3 | Auctions | Item | Bid | Time | |
| | | ANTONIO CAR... | 42.00 | 6 days, 04:22:34 | |

Figure 2.8: Detail of an example table assembled through the DEByE tool.

Although $\tau$ can be regarded as a correct type for this table, for the purpose of data extraction it is too restrictive. Indeed, if we generate the structuring productions based on this type, we would constrain, for instance, the possibility of a Format value to appear along with a Price value. Thus, in our work, we prefer to use a less restrictive policy to derive a scheme from an example table. In the case of the table of Figure 2.8, we might

have derived the type $\tau'$, which can also be considered as a correct type for the example table, as

$$\tau' = (\mathsf{StoreName} : \mathsf{atom}, \mathsf{ItemList} : \tau'_1)$$

where

$$\tau'_1 = (\mathsf{Item} : \mathsf{atom}, \mathsf{By} : \mathsf{atom}, \mathsf{Format} : \mathsf{atom}, \mathsf{Year} : \mathsf{atom}, \\ \mathsf{Price} : \mathsf{atom}, \mathsf{Bid} : \mathsf{atom}, \mathsf{Time} : \mathsf{atom}).$$

The alternative type $\tau'$ opens the possibility of the same instance to have a value for Format and Year (as it actually occurs in some of the target pages from Amazon), even though no instance in the example table features both attributes. Also notice that it is now possible for an instance to have a value for both Bid and Price. However, such a situation is unlikely to occur on the target pages. So, there is no harm in allowing it, since it is not an obligation.

Type $\tau'$ illustrates one of the heuristics we use in our work for *type coercion*, that is, for combining distinct object types into a single object type. Originally adopted in the context of semistructured data management by the Lorel query language [Abiteboul *et al.*, 1997], type coercion is a useful mechanism to reconcile objects with a distinct structure.

In this particular case, notice that, distinct from type $\tau$, type $\tau'$ does not feature a variant. This is because, when combining two table schemes that have at least one column in common, we create a single table scheme containing all columns from both table schemes. Indeed, table schemes such as $\tau'_1$, generated according to our type coercion heuristics, provide a flexible structure, suitable for driving semistructured data extraction. This has been demonstrated by several experiments with the DEByE tool (see Chapter 5).

In the following, we present the *typing* function $\Upsilon$, which derives a table scheme from a given example table. This function is based on the *coercion* operator $\oplus$, which implements our type coercion heuristics. This operator is first defined.

**Definition 3** *The* ***coercion*** *operator $\oplus$ over two columns is defined as follows:*

- $\mathsf{X{:}atom} \oplus \mathsf{X{:}atom} = \mathsf{X{:}atom}$
- $\mathsf{X{:}\{atom\}} \oplus \mathsf{X{:}\{atom\}} = \mathsf{X{:}\{atom\}}$
- $\mathsf{X{:}\{atom\}} \oplus \mathsf{X{:}atom} = \mathsf{X{:}\{atom\}}$
- $\mathsf{X{:}atom} \oplus \mathsf{X{:}}(\mathsf{A}_1 : a_1, \ldots, \mathsf{A}_m : a_m) = \mathsf{X{:}}[\mathsf{atom};\ (\mathsf{A}_1 : a_1, \ldots, \mathsf{A}_m : a_m)]$
- $\mathsf{X{:}\{atom\}} \oplus \mathsf{X{:}}(\mathsf{A}_1 : a_1, \ldots, \mathsf{A}_m : a_m) = \mathsf{X{:}}[\{\mathsf{atom}\};\ (\mathsf{A}_1 : a_1, \ldots, \mathsf{A}_m : a_m)]$
- *Let* $\tau_a = (\mathsf{A}_1 : a_1, \ldots, \mathsf{A}_m : a_m)$, $\mathcal{T}_a = \{\mathsf{A}_1, \ldots, \mathsf{A}_m\}$, $\tau_b = (\mathsf{B}_1 : b_1, \ldots, \mathsf{B}_n : b_n)$ *and* $\mathcal{T}_b = \{\mathsf{B}_1, \ldots, \mathsf{B}_n\}$, *then:*
  - *If* $\mathcal{T}_a \cap \mathcal{T}_b = \emptyset$, $\mathsf{X{:}}\tau_1 \oplus \mathsf{X{:}}\tau_2 = [(\mathsf{A}_1 : a_1, \ldots, \mathsf{A}_m : \mathsf{A}_m);\ (\mathsf{B}_1 : b_1, \ldots, \mathsf{B}_n : b_n)]$
  - *If* $\mathcal{T}_a \cap \mathcal{T}_b \neq \emptyset$, $\mathsf{X{:}}\tau_1 \oplus \mathsf{X{:}}\tau_2 = (\mathsf{C}_1 : c_1, \ldots, \mathsf{C}_p : c_p)$, *where*

* $\mathsf{C}_k : c_k = \mathsf{A}_i : a_i \oplus \mathsf{B}_j : b_j$, *if* $\mathsf{C}_k \in \mathcal{T}_a \cap \mathcal{T}_b$ *and* $\mathsf{C}_k = \mathsf{A}_i = \mathsf{B}_j$, *or*
* $\mathsf{C}_k : c_k = \mathsf{A}_i : a_i$, *if* $\mathsf{C}_k \in \mathcal{T}_a - \mathcal{T}_b$ *and* $\mathsf{C}_k = \mathsf{A}_i$, *or*
* $\mathsf{C}_k : c_k = \mathsf{B}_j : b_j$, *if* $\mathsf{C}_k \in \mathcal{T}_b - \mathcal{T}_a$ *and* $\mathsf{C}_k = \mathsf{B}_j$.

*Further, the following properties apply:*

- $\mathsf{X}{:}\tau_1 \oplus \mathsf{X}{:}\tau_2 = \mathsf{X}{:}\tau_2 \oplus \mathsf{X}{:}\tau_1$
- $(\mathsf{X}{:}\tau_1 \oplus \mathsf{X}{:}\tau_2) \oplus \mathsf{X}{:}\tau_3 = \mathsf{X}{:}\tau_1 \oplus (\mathsf{X} : \tau_2 \oplus \mathsf{X} : \tau_3)$

Informally the coercion operator works as follows. When operating over equal types, the operator gives as a result this same type. When combining a list of atoms with an atomic value, it gives as a result a list of atoms. If any table scheme is to be combined with an atomic value or with a list of atoms, the result is a variant type over them. For the combination of two or more table schemes, two results are possible. If the table schemes do not have any column in common, the result is a variant type over them. If there exists at least one column in common, a single table scheme is built with the union of the columns from the schemes. Additionally, the columns in common are replaced by a single column whose type is the result of a coercion over the types of that columns. The coercion operator is used by the typing function $\Upsilon$ defined below.

**Definition 4** *The* ***typing*** *function* $\Upsilon$ *is defined as follows:*

- *Let* $a$ *be an atomic value, then* $\Upsilon(a) = \mathsf{atom}$
- *Let* $l$ *is a list of atomic values, then* $\Upsilon(l) = \{\mathsf{atom}\}$
- *Let* $T = \{t_1, \ldots, t_n\}$ *be a table and* $t_i = \langle \mathsf{C}_1 : v_1^i, \mathsf{C}_2 : v_2^i, \ldots, \mathsf{C}_m : v_m^i \rangle$ *then*
  - $\Upsilon(t_i) = (\mathsf{C}_1 : \Upsilon(v_1^i), \mathsf{C}_2 : \Upsilon(v_2^i), \ldots, \mathsf{C}_m : \Upsilon(v_m^i))$
  - $\Upsilon(T) = \Upsilon(t_1) \oplus \ldots \oplus \Upsilon(t_n)$

The typing function recursively operates through the structure of the given table. It derives a type for each row and combines these types using the coercion operator.

We now provide an example of how the typing function and the coercion operator work. To illustrate some cases not occurring in the example table of Figure 2.8, we use the example table shown in Figure 2.9.

According to Definition 4, the scheme for this table is given by

$$\Upsilon(\mathsf{T}) = \Upsilon(t_1) \oplus \Upsilon(t_2) \oplus \Upsilon(t_3).$$

Expanding $\Upsilon(t_1)$ we have:

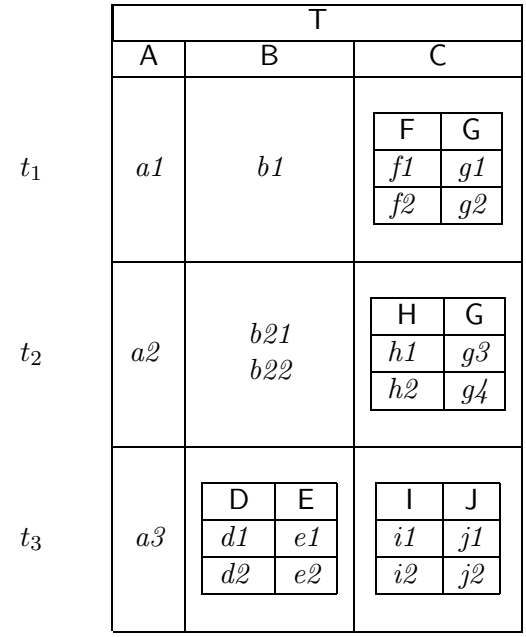



Figure 2.9: An example table for which coercion is needed.

$$\begin{aligned}
\Upsilon(t_1) = \quad & \langle \mathsf{A} : a1, \mathsf{B} : b1, \mathsf{C} : \{\langle \mathsf{F} : f1, \mathsf{G} : g1\rangle, \langle \mathsf{F} : f2, \mathsf{G} : g2\rangle\}\rangle \\
& (\mathsf{A} : \Upsilon(a1), \mathsf{B} : \Upsilon(b1), \mathsf{C} : \Upsilon(\{\langle \mathsf{F} : f1, \mathsf{G} : g1\rangle, \langle \mathsf{F} : f2, \mathsf{G} : g2\rangle\})) \\
& (\mathsf{A} : \mathsf{atom}, \mathsf{B} : \mathsf{atom}, \mathsf{C} : \Upsilon(\langle \mathsf{F} : f1, \mathsf{G} : g1\rangle) \oplus \Upsilon(\langle \mathsf{F} : f2, \mathsf{G} : g2\rangle)) \\
& (\mathsf{A} : \mathsf{atom}, \mathsf{B} : \mathsf{atom}, \mathsf{C} : (\mathsf{F} : \mathsf{atom}, \mathsf{G} : \mathsf{atom}) \oplus (\mathsf{F} : \mathsf{atom}, \mathsf{G} : \mathsf{atom})) \\
& (\mathsf{A} : \mathsf{atom}, \mathsf{B} : \mathsf{atom}, \mathsf{C} : (\mathsf{F} : \mathsf{atom}, \mathsf{G} : \mathsf{atom}))
\end{aligned}$$

Similarly, expanding $\Upsilon(t_2)$ and $\Upsilon(t_3)$ leads to:

$$\Upsilon(t_2) = (\mathsf{A} : \mathsf{atom}, \mathsf{B} : \{\mathsf{atom}\}, \mathsf{C} : (H : \mathsf{atom}, \mathsf{G} : \mathsf{atom}))$$
$$\Upsilon(t_3) = (\mathsf{A} : \mathsf{atom}, \mathsf{B} : (\mathsf{D} : \mathsf{atom}, \mathsf{E} : \mathsf{atom}), \mathsf{C} : (\mathsf{I} : \mathsf{atom}, \mathsf{J} : \mathsf{atom}))$$

Now, to obtain $\Upsilon(\mathsf{T}) = \Upsilon(t_1) \oplus \Upsilon(t_2) \oplus \Upsilon(t_3)$ we do the following:

| | $\Upsilon(t_1)$ | $\Upsilon(t_2)$ | $\Upsilon(t_3)$ |
|---|---|---|---|
| | A:atom | B:atom | C:(F:atom,G:atom) |
| $\oplus$ | A:atom | B:{atom} | C:(H:atom,G:atom) |
| | A:atom | B:(D:atom,E:atom) | C:(I:atom,J:atom) |
| $\Upsilon(\mathsf{T})$ | A:atom | B:[{atom};(D:atom,E:atom)] | C:[(F:atom,G:atom,H:atom);(I:atom,J:atom)] |

Finally, the scheme obtained for table T is:

$$\Upsilon(\mathsf{T}) = (\mathsf{A{:}atom},\ \mathsf{B{:}[\{atom\};(D{:}atom,E{:}atom)]},\ \mathsf{C{:}[(F{:}atom,G{:}atom,H{:}atom);(I{:}atom,J{:}atom)]})$$

## 2.4 DTOR – Implementing Nested Tables using XML

An important practical requirement of any data extraction tool or system is providing an output that can be easily consumed and processed by other tools and systems. In this section, we present *DTORF (DEByE Textual Object Repository Format)*, an XML-based format for encoding nested tables with structural variations. DTORF is the output format used by the DEByE tool. In practice, DTORF plays the role of a "bridge" between DEByE and many other tools and systems that process XML. This is important since XML is the most important standard for data representation and exchange on the Web.

To represent complex objects using XML, the usual solution consists of creating an XML element[2] for each object to be represented. In our case, if we simply do that, we lose the logical relationships between the objects described by the user while specifying the examples. Thus, in DTORF, as we show below, we made such logical relationships explicit by imposing constraints on the way of XML documents that are constructed. These constraints follow the definition of a table scheme presented earlier. The XML documents produced according to DTORF are called *DTOR (DEByE Textual Object Repositories)*. An example of a DTOR is presented in Figure 2.10.

In a DTOR, each XML element corresponds to an object, and special XML attributes are used to indicate the logical role of the objects. The set of elements used in a DTOR are limited to `<ATOM>`, `<TUPLE>`, `<LIST>`, `<VALUE>` and `<OBJECTS>`.

In any DTOR, all objects are nested in a root element `<OBJECTS>`. In this element, an attribute `sourcehref` indicates the source page from where the objects were extracted. In the case of the DTOR of Figure 2.10, as indicated in Line 2, the objects were extracted from the file `/home/alti/extrat/nested/jobim.html`.

The elements `<ATOM>`, `<TUPLE>` and `<LIST>` denote the logical role of the objects whose types are described by the value of the attribute `type` in each tag. This can be seen, for instance, in Lines 3, 4 and 7. In Line 3, we have an element representing an object of type Store, which is a tuple. In Line 4, there is an element representing an atom of type StoreName. Line 7 begins a list of type ItemList.

The objects represented by `<TUPLE>` elements are tuples, i.e., aggregations of objects of distinct types. Thus, all elements nested in `<TUPLE>` elements must represent objects of distinct types, i.e., they must have distinct values for the attribute `type`. To enforce the tabular structure of the objects, only `<ATOM>` and `<LIST>` elements are allowed inside `<TUPLE>` elements. This is exemplified by the `<TUPLE>` element beginning at Line 3.

The `<LIST>` elements denote homogeneous sets of objects. Thus, all elements nested

[2]We adopt here the DOM [Hégaret and Wood, 2002] terminology, in which pairs of matching tags of the form `<NAME> ...</NAME>` are termed *elements*, and pairs of the form `a="v"` occurring inside the opening tag of an element are called *attributes*.

```
<?xml version = "1.0" encoding = "iso-8859-1"?>
<OBJECTS sourcehref="file:/home/alti/extrat/nested/jobim.html">
  <TUPLE ipos="11514" type="Store">
    <ATOM ipos="11514" type="StoreName">
        <VALUE fpos="11527" ipos="11514"><![CDATA[Popular Music]]></VALUE>
    </ATOM>
    <LIST ipos="11819" type="ItemList">
      <TUPLE ipos="11819" type="ItemList">
        <ATOM ipos="11819" type="Item">
          <VALUE fpos="11900" ipos="11819"><![CDATA[Francis Albert ...]]]></VALUE>
        </ATOM>
        <ATOM ipos="11908" type="By">
          <VALUE fpos="11921" ipos="11908"><![CDATA[Frank Sinatra]]></VALUE>
        </ATOM>
        <ATOM ipos="11923" type="Format">
          <VALUE fpos="11931" ipos="11923"><![CDATA[Audio CD]]></VALUE>
        </ATOM>
      </TUPLE>
      <TUPLE ipos="12171" type="ItemList">
         ...
      </TUPLE>
    </LIST>
  </TUPLE>
  ...
  </TUPLE>
  <TUPLE ipos="21440" type="Store">
    <ATOM ipos="21440" type="StoreName">
      <VALUE fpos="21448" ipos="21440"><![CDATA[Auctions]]></VALUE>
    </ATOM>
    <LIST ipos="22067" type="ItemList">
      <TUPLE ipos="22067" type="ItemList">
        <ATOM ipos="22067" type="Item">
          <VALUE fpos="22145" ipos="22067"><![CDATA[ANTONIO CARLOS JOBIM ...]]></VALUE>
        </ATOM>
        <ATOM ipos="22167" type="Bid">
          <VALUE fpos="22172" ipos="22167"><![CDATA[42.00]]></VALUE>
        </ATOM>
        <ATOM ipos="22176" type="Time">
          <VALUE fpos="22200" ipos="22176"><![CDATA[Ends in 6 days, 04:22:42]]></VALUE>
        </ATOM>
      </TUPLE>
      ...
    </LIST>
  </TUPLE>
</OBJECTS>
```

Figure 2.10: Example of a DTOR.

in <LIST> elements must represent objects of a same type, i.e., they must have the same value for the attribute type. To enforce the tabular structure of the objects, only <ATOM> and <TUPLE> elements are allowed inside <LIST> elements. This is exemplified by the <LIST> element beginning at Line 7. As a convention, we make the value of the type attribute in a <LIST> element equal to the value of the attribute type in the elements immediately nested in it. This is exemplified in Lines 7 and 8. Notice that, as a format for representing semistructured data, DTORF does not impose any constraint in the way the internal members of lists are composed.

We notice that the root element <OBJECTS> denotes a collection of homogeneous objects in the same way as a <LIST> element. Thus, the same constraints applying to <LIST> elements also apply to the root element <OBJECTS>.

For representing atoms, <ATOM> elements are used. All <ATOM> elements must contain exactly one <VALUE> element, whose content is the value of the atom. This is illustrated, for instance, in Line 5. In <VALUE> elements, two attributes ipos="i" and fpos="f" must be included. They indicate, respectively, the initial and the final positions of the extracted string. Notice that this positional information is redundant with the content of the <VALUE> elements. This redundancy is tolerated, simply because it facilitates the automatic verification of extraction results in large scale experiments.

The attribute ipos is also used in elements <ATOM>, <TUPLE> and <LIST> with a specific role. For any of these elements, the value of ipos is equal to smallest value of ipos in any element nested in it. This allows the unique identification of each object represented in a DTOR, which is done using the values of ipos, type and the tag of the element itself. For instance, in Line 26 we have a tuple of type Store that can be identified by the triple ⟨<TUPLE>,Store,21440⟩.

# Chapter 3

# Wrapper Generation

We present in this chapter the techniques we have developed for generating wrappers based on examples of the data to be extracted. In all of these techniques, examples are taken from a *sample* page of a target Web source. Once generated, a wrapper is expected to work properly for all other pages from that target Web source, as long as the overall structure and presentation features remain the same. For instance, if we generate a wrapper based on examples taken from the Web page of Figure 1.1, this wrapper will also work for all answer pages returned from the same query interface from the Amazon Web site.

Generally, in practice, few examples are required for obtaining a good wrapper. Indeed, the effectiveness of the techniques presented here was demonstrated by experimental results presented in Chapter 5. Some of these results have also appeared in recent publications [Laender *et al.*, 2002b; 2000; Ribeiro-Neto *et al.*, 1999b; 1999a].

As discussed in Chapter 1, a wrapper is an implementation of a mapping that recognizes (among other uninteresting pieces of text) attribute values for objects of interest in a target page, associating an appropriate semantics to them. In our approach, such a mapping corresponds to a special type of grammar called a *tabular grammar*. Tabular grammars are context-free grammars that describe how to extract atomic values and how to assemble complex objects using these values. As we shall see, the parsing of a data-rich Web page containing semistructured complex objects using tabular grammars produces derivations that are, in fact, table instances. This means that tabular grammars are used to interpret implicit complex objects in data-rich Web pages as rows of a nested table with internal variations.

The productions of a tabular grammar can be divided into two distinct sets: a set of *extracting* productions, which are used to identify and extract atomic attribute values in a target page, and a set of *structuring* productions, which describe how these attribute values must be combined to form complex objects. Our approach for wrapper generation consists, in fact, in generating the extraction and the structuring productions that compose a tabular grammar. In Section 3.1 we present and formalize the concept of a tabular grammar, defining the types of production that comprise it.

The generation of tabular grammars has two distinct steps: (1) the generation of the extracting productions and (2) the generation of the structuring productions. The first

step requires the strings from the sample page to be identified as examples of values of attributes of interest (e.g., book titles). The second step requires the definition of a target structure from the complex objects to be built. The DEByE tool allows a user to provide all information required by both steps in a uniform manner. Through the DEByE tool GUI, the user assembles a nested table in which rows correspond to complex objects and whose columns contain strings taken from a sample page. The structure of the nested table assembled suggests the target structure for building new complex objects. Similarly, strings taken from the sample page serve as examples of values of attributes.

Our technique for generating extracting productions is presented in Section 3.2, while in Section 3.3 we present our technique for obtaining a tabular grammar from an example nested table. The process consists of first generating a table scheme that represents the example nested table and then mapping this table scheme into a set of structuring productions.

Finally, in Section 3.4, we discuss *oe-patterns*, a concise representation for tabular grammars. In practice, in the context of the DEByE tool, an oe-pattern corresponds to a wrapper.

## 3.1 Tabular Grammars

Consider a Web page containing implicit complex objects of interest. In many cases, it is possible to write a context-free grammar to extract complex objects from such pages. Using this grammar, a Web data extraction algorithm generates a parse tree that is used to recognize complex objects and store them using a suitable format (e.g., XML). This strategy for Web data extraction is adopted explicitly by tools such as ARANEUS [Mecca *et al.*, 1998] and W4F [Sahuguet and Azavant, 2001], but it is also implicitly adopted by several other tools described in the literature (e.g., Lixto [Baumgartner *et al.*, 2001], RoadRunner [Crescenzi *et al.*, 2001], XWRAP [Liu *et al.*, 2000], STALKER [Muslea *et al.*, 2001], etc.).

The derivation of grammars for Web data extraction may follow a two-step strategy. First, one writes terminal productions that drive the lexical analysis of the Web page. These productions are used to recognize the atomic values in the page (e.g., the title of a book, the name of an artist, etc.). Then, one writes non-terminal productions that parse the result of the lexical analysis and "assemble" complex objects from the atomic values derived by the terminal productions.

As an example, consider the Web page shown in Figure 3.1. We present in Figure 3.2 a simple grammar for extracting data from this page. In this grammar, Productions 10 to 20 drive the lexical analysis of the Web page, whereas Productions 1 to 9 parse the result of the lexical analysis and recognize the structure of the complex objects. For instance, Production 9 describes a tuple of attributes "Item", "Bid" and "Time", Production 8 describes a list of such tuples, and Production 4 describes two possible structural compositions for tuples that represent stores. By parsing the Web page in Figure 1.1, according to the grammar in Figure 3.2, we can, for instance, create an XML file that encodes the extracted

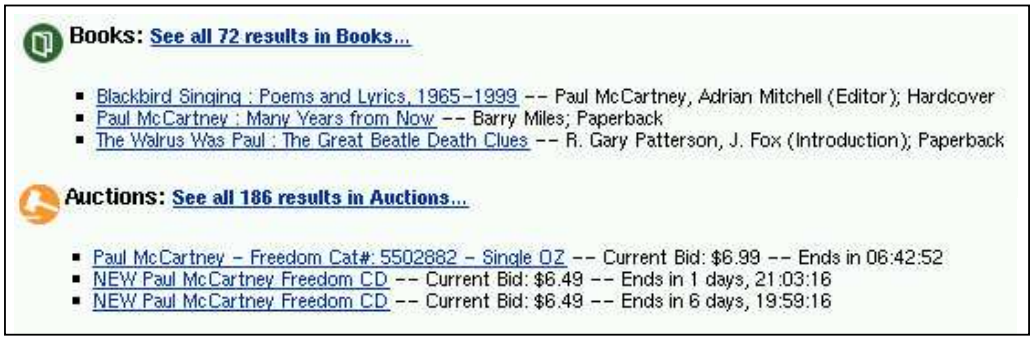



Figure 3.1: An excerpt of a Web page from **Amazon**.

data.

```
1.            ⟨Doc⟩ → ⟨StoreList⟩
2.      ⟨StoreList⟩ → ⟨StoreTuple⟩⟨StoreList⟩|⟨StoreTuple⟩
3.     ⟨StoreTuple⟩ → ⟨StoreName⟩⟨ProductVar⟩
4.     ⟨ProductVar⟩ → ⟨ProductList1⟩|⟨ProductList2⟩
5.   ⟨ProductList1⟩ → ⟨ProductTuple1⟩⟨ProductList1⟩|⟨ProductTuple1⟩
6.  ⟨ProductTuple1⟩ → ⟨Item⟩⟨AuthorList⟩⟨BookType⟩
7.     ⟨AuthorList⟩ → ⟨Author⟩⟨AuthorList⟩|⟨Author⟩
8.   ⟨ProductList2⟩ → ⟨ProductTuple2⟩⟨ProductList2⟩|⟨ProductTuple2⟩
9.  ⟨ProductTuple2⟩ → ⟨Item⟩⟨Bid⟩⟨Time⟩
10.     ⟨StoreName⟩ → "<b class=sans>"⟨String⟩":</b>"
11.          ⟨Item⟩ → "<a href="⟨String1⟩"> --"⟨String⟩"</a>"
12.        ⟨Author⟩ → "--"⟨String1⟩","⟨String⟩";"
13.      ⟨BookType⟩ → ";"⟨String⟩"<li>"
14.           ⟨Bid⟩ → "Bid:$"⟨String⟩"--"
15.          ⟨Time⟩ → "Ends in"⟨String⟩"<li>"
16.        ⟨String⟩ → ⟨Ch⟩|⟨Ch⟩⟨String⟩
17.       ⟨String1⟩ → ⟨Ch1⟩|⟨Ch1⟩⟨String1⟩
18.            ⟨Ch⟩ → ⟨Ch1⟩|⟨Sy⟩
19.           ⟨Ch1⟩ → "A" | ... | "Z" | "a" | ... | "z" | "1" | ... | "9"
20.            ⟨Sy⟩ → "!" | ... | "?"
```

Figure 3.2: A simple grammar for extracting data from excerpt the Web page in Figure 3.1.

The grammar in Figure 3.2 is by no means the only one that can be constructed for extracting data from our example page. Indeed, it was designed to extract a particular subset of the data present in our example page and to organize such data according to a certain structure. The resulting parse tree bares a structural resemblance to the data in the source Web page itself. In particular, the productions of the grammar were crafted so that the parse tree can be mapped into a *nested table with internal variations* such as the one in Figure 2.2. The grammar in Figure 3.2 is an example of what we call a *tabular*

*grammar.*

Tabular grammars are context-free grammars whose productions have specific formats that lead to parse trees that correspond to nested tables with variants. Our approach, detailed later, aims at generating tabular grammars for Web data extraction.

Let $G = \langle N, T, R, D \rangle$ be a grammar, where $N$ is the set of non-terminals, $T$ is the set of terminals, $R$ is the set of productions, and $D$ is the start symbol. Recall that $G$ is a *context-free grammar* iff every production $\alpha \rightarrow \beta$ is such that $\alpha$ is a single non-terminal and $\beta$ is non-empty [Hopcroft *et al.*, 2001]. We assume, without loss of generality, that $D$ does not appear on the right-hand side of any production.

We are now ready to define tabular grammars.

**Definition 5** *A context free grammar $G = \langle N, T, R, D \rangle$ is a **tabular grammar** iff $R$ can be partitioned into two sets $R_1$ and $R_2$ such that $R_1$ defines a regular language and $R_2$ contains only productions of the following classes:*

- ***aggregating** (or **tuple-generating**) productions of the form*

$$X_0 \rightarrow X_1 X_2 \ldots X_n$$

*where $n \geq 1$, and $X_i \neq X_j$, for every $0 \leq i, j \leq n$. Further, each $X_k$, $k = 1 \ldots n$, must appear on the left-hand side of either (a) exactly one iterating production in $R_2$ or (b) at least one extracting production in $R_2$;*

- ***iterating** (or **list-generating**) productions of the form*

$$X \rightarrow YX|Y$$

*where $X \neq Y$ and $Y$ must appear on the left-hand side of either (a) exactly one aggregating production in $R_2$, or (b) exactly one varying production in $R_2$, or (c) at least one extracting production in $R_2$;*

- ***varying** (or **variant-generating**) productions of the form*

$$X_0 \rightarrow X_1|X_2|\ldots|X_n$$

*where $(n \geq 2)$, and $X_i \neq X_j$, for every $0 \leq i, j \leq n$. Further, each $X_k$, $k = 1 \ldots n$, must appear on the left-hand side of exactly one iterating production in $R_2$;*

- ***extracting** (or **atom-generating**) productions of the form $X \rightarrow \rho$, where $\rho$ is a string of terminal and non-terminal symbols. Further, if there is some non-terminal symbol $A$ in $\rho$, it must appear on the left-hand side of a production in $R_1$.*

*In addition, the aggregating, iterating, and varying productions are called **structuring** productions and those in $R_1$ are called **lexical** productions.*

Consider the grammar in Figure 3.2. Productions 1, 3, 6 and 9 are aggregating productions; 2, 5, 7 and 8 are iterating productions; 4 is a varying production; and 10 to 15 are extracting productions. Further, Productions 16 to 20 define a regular language since they can be transformed into an equivalent set of right-linear productions.

The aggregating productions are invoked only if all symbols on their right-hand side derive some substring of the input Web page. This induces a very strict data extraction process in which all components of a given tuple must be present and must occur in a fixed order, which is not likely to occur in typical Web data. To circumvent this problem, we generalize single tuple-generating productions, by replacing them by a set of tuple-generating productions that capture all possible combinations of the attributes expected. For the sake of simplicity, instead of writing all these productions, we use the notation introduced by Definition 6.

**Definition 6** *For tabular grammars, the expression* $X_0 \rightarrow (X_1 \ldots X_n)^{\Delta}$ $(n \geq 2)$ *denotes the set of productions:*

$$\{X_0 \rightarrow X_1 | X_1 X_2 | X_1 X_2 \ldots X_n |, \ldots, X_0 \rightarrow X_n | X_n X_1 | X_n X_1 \ldots X_{n-1}\}$$

In Figure 3.3, we present the expressions that replace the aggregating productions of the grammar of Figure 3.2, using the notation of Definition 6. Notice that using such sets of aggregating productions adds great flexibility to the extraction process, but complicates the parsing of the target Web pages. In fact, depending on the table scheme, a tabular grammar can lead to a complex parsing process. However, by imposing the constraints described by Definition 5 on the grammar structure, it is possible to use a specific parsing algorithm for tabular grammars that makes the Web data extraction process feasible for practical purposes. This algorithm, called the *bottom-up algorithm*, was first presented in [Laender *et al.*, 2002b; Ribeiro-Neto *et al.*, 1999b] and is discussed in Chapter 4 as one of the contributions of the work developed.

3. $\langle$StoreTuple$\rangle \rightarrow (\langle$StoreName$\rangle\langle$ProductVar$\rangle)^{\Delta}$
6. $\langle$ProductTuple1$\rangle \rightarrow (\langle$Item$\rangle\langle$AuthorList$\rangle\langle$BookType$\rangle)^{\Delta}$
9. $\langle$ProductTuple2$\rangle \rightarrow (\langle$Item$\rangle\langle$Bid$\rangle\langle$Time$\rangle)^{\Delta}$

Figure 3.3: Expanded versions of the aggregating productions of the grammar of Figure 3.2.

## 3.2 Generating Extracting Productions

The first step in the derivation of a tabular grammar is the generation of extracting and lexical productions. In our approach, this is accomplished through the generation of avp-patterns.

**Definition 7** *Let $g$ be a Web page and let $s$ be a string in $g$. Let $A$ be an attribute. We define an* ***attribute-value pair (avp)*** *as a pair $A : s$ that assigns $s$ as a value of $A$.*

Given a set of example avps $\{A : s_1, \ldots, A : s_n\}$, our goal is to find contextual information (i.e., markups, symbols, keywords, etc.) common to the values of $A$ that occur in $g$. This information is represented by regular expressions, that we call *attribute-value pair patterns* or *avp-patterns*. Each avp-pattern found will be used to create an extracting production of the tabular grammar.

**Definition 8** *An **attribute-value pair pattern (avp-pattern)** is a pair $A : \rho$, where $A$ is an attribute and $\rho$ is a text pattern. The text pattern $\rho$ is used to match string values, in the domain of $A$, as they occur in a given text (i.e., a Web page).*

For generating an avp-pattern, with each given example avp, $A : s_i$, we associate a *local syntactic context* that can be derived from the strings surrounding the avp value $s_i$ in the text. We use the concept of a *passage* (or *window*) and techniques from information retrieval [Baeza-Yates and Ribeiro-Neto, 1999; Callan, 1994; Kaszkiel and Zobel, 1997], as follows. The tokens surrounding the avp value constitute a passage that can be used as its *local context*. For instance, Figure 3.4(a) illustrates the value 6.99 of the attribute Bid (which occurs in Figure 1.1) and passages that can be used as its local context. Using this context information, we build an avp-pattern that can be later used to identify other values for the attribute Bid. Figure 3.4(b) illustrates a possible representation of this avp-pattern. We refer to this avp-pattern as $A : s_{pre} * s_{suf}$, where the symbol * is used to match any sequence of characters (representing a value selected by the user to assemble an example), $s_{pre}$ refers to a string that is a prefix for * and $s_{suf}$ refers to a string that is a suffix for *.

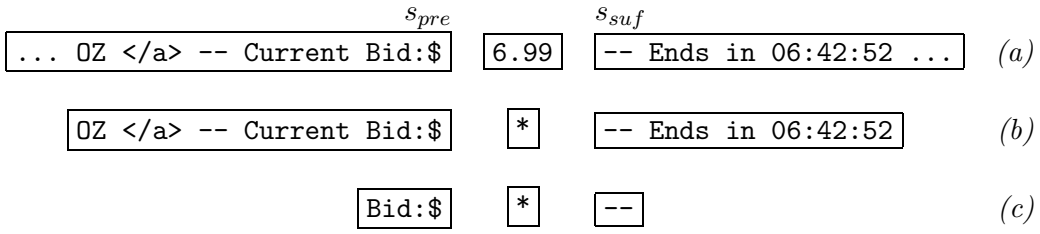



Figure 3.4: Examples of avp-patterns.

The avp-pattern in Figure 3.4(b) is too specific and will most likely not to match other values expected for the attribute Bid. The key problem is that this pattern includes too much information about the local context in which the value 6.99 appeared. Thus, to be able to effectively use this avp-pattern for recognizing and extracting new values for the attribute Bid in other Web pages, it must be transformed into a new pattern that is more general, in the sense that it contains less contextual information. Figure 3.4(c) shows a variation of the avp-pattern for the value 6.99, generated by reducing the length of the prefix $s_{pre}$ and of suffix $s_{suf}$. This new avp-pattern can now be used to effectively match other values for Bid in this page, i.e., it works as an extracting production.

In general, the generation of avp-patterns can be accomplished as follows. Given a string selected by the user, we determine a passage surrounding this the text. Initially, we adopt symmetric passages composed of $W$ text tokens to the right and $W$ text tokens to

the left of the string. Afterwards, due to simple heuristics applied to token identification (for instance, a run of spaces is considered a single token), an avp-pattern might become asymmetric as illustrated in Figure 3.4(c).

An avp-pattern is determined empirically as follows. We start with a small pattern composed solely of the symbol *, of a token to its right and of a token to its left. These tokens can be character strings or special symbols. For instance, in Figure 3.4(c) the initial avp-pattern would be composed of the symbol * surrounded by the prefix "Bid:$" and the suffix "– –". Notice that the prefix and the suffix are very useful, are present in the page, and can be recognized automatically once the user marks 6.99 as a value of interest. We then parse the sample page (which is displayed on the user's screen) looking for matches to the avp-pattern just defined, and count the number of matches. This count is compared with an estimate, provided by the user, for the number of Bid values in the sample page. If the number of matches counted exceeds the number of Bid values estimated by the user, we add additional terms to the pattern, increasing its width $W$ and the amount of contextual information attached to it. This process is repeated automatically until we have a good definition for the local context of the avp in consideration (we stop when the number of matches is smaller than the number of Bid values identified by the user).

Notice that, in the case of the DEByE tool, all users have to provide is a single number that indicates the number of occurrences (values) of Bid in the sample page they sees in front of them. In fact, the number provided by the user does not need to be the exact number of occurrences of values of a given attribute in the page. A rough approximation of this number is in most cases sufficient to adjust the width $W$ of the avp-pattern. Such information is quite simple to provide and presents little inconvenience to the user.

In practice, there are cases in which not all values of a given attribute present a same single common context in the target page. In cases like these, it is necessary to provide more than one example value, which results in more than one alternative avp-pattern.

It is important to notice that, so far in this section, we have discussed only avp-patterns generated for attributes. However, the same discussion remains valid for the case of lists of the form $L$:{atom}. In this case, the example strings provided are considered as examples of members of the list. As a consequence, avp-patterns will match atomic values that will later compose lists.

As avp-patterns are nothing more than regular expressions, the avp-patterns generated as described in this section can be converted into equivalent regular grammars. This is important since these regular grammars can be combined with structuring productions to compose a tabular grammar [Sudkamp, 1997].

## 3.3 Mapping Table Schemes into Tabular Grammars

In this section we describe how the structuring productions of a tabular grammar can be obtained from an example table. This process consists in first deriving a plausible scheme for this table and then mapping this scheme into the set of structuring productions.

In Section 2.3, we have described the typing function $\Upsilon$ that, when given a table

instance, obtains a plausible table scheme for this instance using a particular type of coercion policy. As the DEByE tool was built for guaranteeing that any example table assembled corresponds to a table instance according to Definition 2, we use the typing function for generating the table scheme.

For the generation of structuring productions, in the following we present a recursive procedure, **TabGram**, that maps this table scheme into a set of aggregating, iterating and varying productions of a tabular grammar. This procedure also includes the conversion of avp-patterns into equivalent extracting and lexical productions. The **TabGram** procedure is described in Figure 3.5.

```
1  TabGram(τ:type;<S>:symbol; R:set of productions)
2
3  begin
4    Let τ = (C_1 : [τ_1^1; ...; τ_1^{n_1}], C_2 : [τ_2^1; ...; τ_2^{n_2}], ..., C_m : [τ_m^1; ...; τ_m^{n_m}])
5    for i = 1 to m do
6        Create a new symbol <C_i>
7        for j = 1 to n_i do
8            Create a new symbol <C_i.j>
9            if τ_i^j = atom ∨ τ_i^j = {atom}
10             then RegGram(C_i,τ_i^j,<C_i.j>,R_i^j)
11             else TabGram(τ_i^j,<C_i.j>,R_i^j)
12           fi
13       end
14       if n_i = 1
15         then R ← R ∪ R_i^1 ∪ {<C_i> ⟶ <C_i.1>}
16         else R ← R ∪ R_i^1 ∪ ... ∪ R_i^{n_i} ∪ {<C_i> ⟶ <C_i.1>|...|<C_i.n_i>}
17       fi
18   end
19   R ← R ∪ {<S.tuple> ⟶ (<C_1><C_2>...<C_m>)^Δ}
20   R ← R ∪ {<S> ⟶ <S.tuple><S>|<S.tuple>}
21 end
```

Figure 3.5: A Procedure for mapping a table scheme into a tabular grammar.

This procedure is invoked over a type $\tau$ that is in fact a table scheme, and generates a tabular grammar whose productions are added to the set $R$. A parameter <S> indicates the start symbol of the generated grammar. We assume that the alphabet and the remaining symbols of the grammar are implicitly defined.

The procedure first iterates over each column $C_i$ of $\tau$ (Loop 5–18) and over each alternative type $\tau_i^j$ within these columns (Loop 7–13). For each $\tau_i^j$, two cases are possible:

1. if $\tau_i^j$ is an atomic type or an atomic list type, a procedure **RegGram** is invoked that generates a regular grammar corresponding to the set of avp-patterns generated for $C_i$, according to the description in Section 3.2. This grammar contains the extraction production and the corresponding lexical productions for extracting atomic values of

$C_i$. It has <C_i.j> as its start symbol and $R_i^j$ as the set of its productions. The exact description of **RegGram** is omitted here.

2. if $\tau_i^j$ is a table scheme, the procedure **TabGram** is recursively invoked to generate a tabular grammar for $\tau_i^j$. This grammar has <C_i.j> as its start symbol and $R_i^j$ as the set of its productions.

In Lines 14–17, the tabular and regular grammars previously generated are integrated into the tabular grammar being currently generated by adding their productions to $R$ and by "connecting" these productions by means of a varying production (Line–16) or by means of a simple auxiliary production (Line–15), if there is no more than one alternative type.

Finally, after the iteration over all columns $C_i^j$, an aggregating production is generated in Line 19, along with an iterating production in Line 20.

## 3.4 Object Extraction Patterns

Within the context of the DEByE approach, tabular grammars are represented by means of what we call *Object Extraction Patterns (oe-patterns)*. Essentially, an oe-pattern encodes two kinds of information necessary for guiding the process of data extraction: the structure of the objects of interest, in the form of a table scheme, and the textual surroundings (i.e., markups, symbols, keywords, etc.) by means of avp-patterns. A more precise definition of an oe-pattern follows.

**Definition 9** *An* oe-pattern *is a pair* $\langle \tau, \mathcal{P} \rangle$ *where* $\tau$ *is a table scheme and* $\mathcal{P} = A_1 : \rho_1 \ldots, A_n : \rho_n$ $(n \geq 1)$ *is a list where each* $A_i : \rho_i$ *is an avp-pattern such that* $A_i$ *is an attribute contained in* $\tau$.

In the DEByE tool, oe-patterns are encoded using XML, in a way very similar to what is done in a DTOR (see Section 2.4). Figure 3.6 shows an XML document encoding an oe-pattern for extracting data from the Web page of Figure 1.1. In this XML document, <ATOM>, <TUPLE> and <LIST> elements are used to describe the structure of the objects to be assembled. Nested inside <ATOM> elements, we place <PATTERN> elements that enclose regular expressions that encode extraction productions. As there can be more than one extraction production for a given attribute, an <ATOM> element can nest more than one <PATTERN> element. This is illustrated in Figure 3.6. In this figure, most of the regular expressions were truncated to fit the page.

```
<?xml version = "1.0" encoding = "iso-8859-1"?>
<OBJECTS mandatory="false">
  <TUPLE type="Store">
    <ATOM mandatory="false" type="StoreName">
      <PATTERN><![CDATA[<b class\="[^"]*?">...(?=[\s]*?: )]]></PATTERN>
    </ATOM>
    <LIST type="ItemList">
      <TUPLE type="ItemList">
        <ATOM mandatory="false" type="Item">
          <PATTERN><![CDATA[<li>[\s]*?<[aA]...</a>[\s]*?-- )]]></PATTERN>
          <PATTERN><![CDATA[<li>[\s]*?<[aA]...</a>[\s]*?\()]]></PATTERN>
        </ATOM>
        <ATOM mandatory="false" type="By">
           <PATTERN><![CDATA[</a>[\s]*?...)(?=[\s]*?; [\s]*?Audio)]]></PATTERN>
           <PATTERN><![CDATA[\)\n-- [\s]*?...(?=[\s]*?; )]]></PATTERN>
        </ATOM>
        <ATOM mandatory="false" type="Format">
           <PATTERN><![CDATA[; [\s]*?([...)]]></PATTERN>
           <PATTERN><![CDATA[\)\n-- [\s]*?([\x2...)]]></PATTERN>
        </ATOM>
        <ATOM mandatory="false" type="Year">
           <PATTERN><![CDATA[\([\s]*?([0-9]+)(?=[\s]*?\)\n-- )]]></PATTERN>
        </ATOM>
        <ATOM mandatory="false" type="Bid">
           <PATTERN><![CDATA[: \$[\s]*?([0-9]+\.[0-9]+)(?=[\s]*? -- )]]></PATTERN>
        </ATOM>
        <ATOM mandatory="false" type="Time">
           <PATTERN><![CDATA[ -- [\s]*?(...?<li>)]]></PATTERN>
           <PATTERN><![CDATA[ -- [\s]*?(...<br clear=left>)]]></PATTERN>
        </ATOM>
        <ATOM mandatory="false" type="Price">
           <PATTERN><![CDATA[: \$[\s]*?(...?<li>)]]></PATTERN>
           <PATTERN><![CDATA[: \$[\s]*?(...)]]></PATTERN>
        </ATOM>
      </TUPLE>
   </LIST>
 </TUPLE>
</OBJECTS>
```

Figure 3.6: Example of an oe-pattern.

# Chapter 4

# Data Extraction Strategies

Once a tabular grammar has been generated for pages of a given Web source, the process of extracting objects of interest from a target page consists of parsing this page and converting the parse tree obtained into a convenient format. Although any standard parsing procedure could be used for this task [Sudkamp, 1997], tabular grammars have, as seen in Section 3.1, very specific features that can be exploited to make the extraction process more efficient.

In this chapter we present two strategies we have developed for extracting complex objects of interest from Web pages given as input. Basically, these extraction strategies can be seen as specialized parsing procedures that parse the target Web page according to a tabular grammar previously generated. These procedures take advantage of the constraints imposed over the productions of the tabular grammar to make the Web data extraction process feasible for practical purposes. Thus, they offer suitable alternatives for implementing the Extraction module of the DEByE tool. The strategies presented here are called *top-down* and *bottom-up*. The top-down strategy was first presented in [Ribeiro-Neto *et al.*, 1999a], while the bottom-up strategy was introduced in [Ribeiro-Neto *et al.*, 1999b].

Both of these data extraction strategies rely on a user provided structural description for the objects to be assembled, which is implicitly described by the structuring productions of the tabular grammar. Also in this chapter, in Section 4.4, we present an algorithm that does not rely on such a structural description. Instead, this algorithm uses only the extracting productions for obtaining atomic attribute values and, analyzing the relative positions of these values in the target page, tries to uncover a plausible structure for assembling complex objects.

## 4.1 Top-down Extraction Strategy

The top-down extraction strategy consists of first locating text regions containing the objects of interest and successively decomposing the located text regions to extract these objects. The general functioning of this strategy is illustrated in Figure 4.1. In this figure, data on authors and their books occurring in a Web page must be extracted. According to the top-down extraction strategy, first, the text region that contains data on all authors

is located. Next, each region containing data on a single author is extracted, and so on, until each author name, book title and price has been extracted.

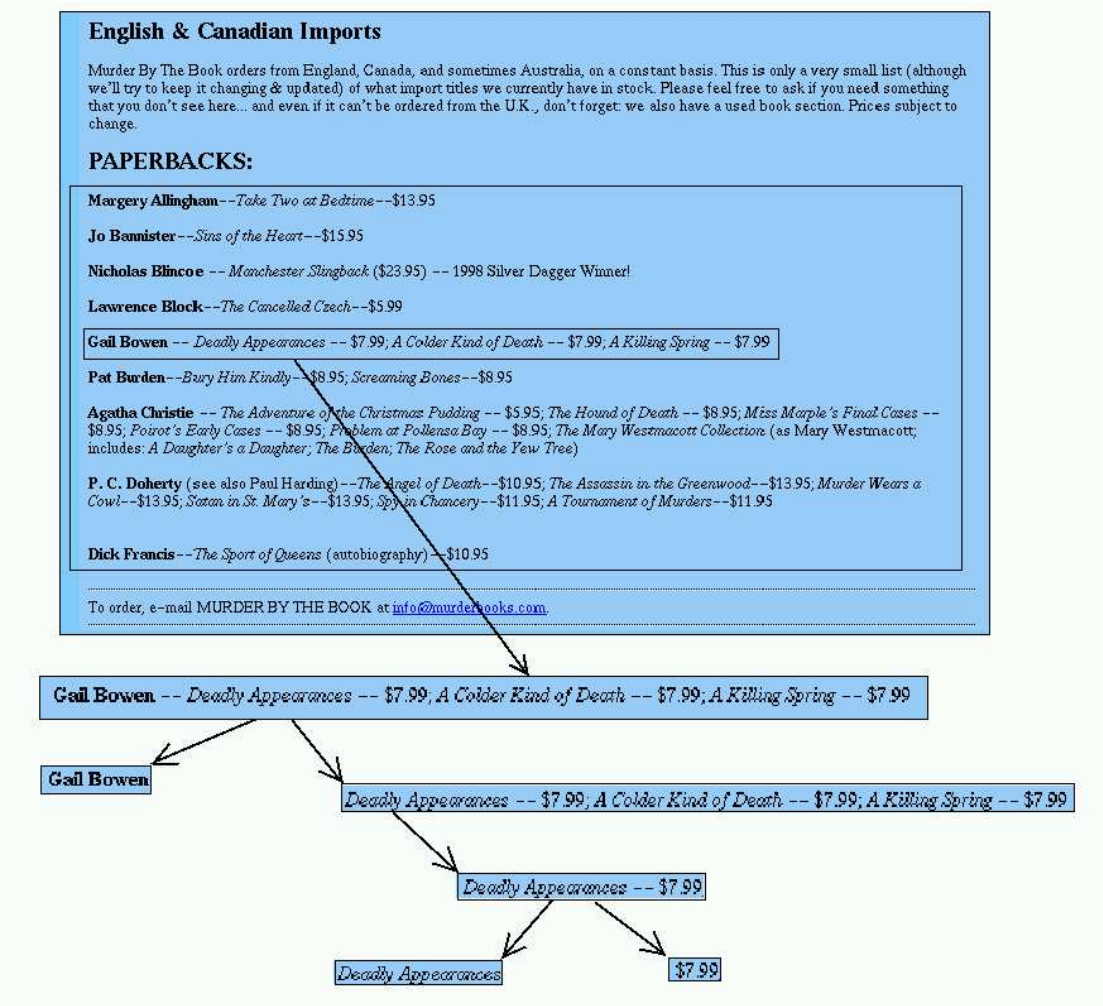



Figure 4.1: The top-down extraction strategy.

The recursive algorithm **Top-Down** in Figure 4.2 describes the top-down strategy. Given a column definition $C : \tau$ and a text region $g$, in each invocation, the algorithm extracts objects from it and returns them as an instance $T$. The algorithm distinguishes two cases: (1) the extraction of atoms and lists of atoms (Lines 4–10) and (2) the building of tables (Lines 11–23). For the first case, avp-patterns are simply used to extract atoms within the text portion $g$. For the second case, the algorithm generates an expression $\rho$ that extracts a whole text region that contains objects of interest (Line 12). This expression is derived from the tabular grammar using an aggregating production $< \mathtt{C} > \rightarrow (< \mathtt{C}_1 >< \mathtt{C}_2 > \ldots < \mathtt{C}_m >)^{\triangle}$. For instance, in Figure 4.1, expressions are generated for extracting the text region containing data on all authors and their books, each

text region containing data on a single author, each text region containing lists of book, etc. Once text regions have been identified, the algorithm is recursively invoked to extract objects they contain. This is performed by the loop in Lines 13–20. Finally, the results of the recursive executions are aggregated in a tuple (Line 20), and these tuples are added to a table (Line 21). We say that the algorithm is *top-down* because it recognizes complete objects which are then broken down into their components.

```
1  Top-Down(⟨C : τ⟩:column;g:text portion;T:instance)
2
3  begin
4    if τ = atom ∨ τ = {atom}
5      then foreach avp-pattern C : ρ do
6             foreach s that match ρ in g do
7               T ← T ∪ {s};
8             end
9           end
10   fi
11   if τ = (C1 : [τ1^1; ...; τ1^n1], C2 : [τ2^1; ...; τ2^n2], ..., Cm : [τm^1; ...; τm^nm])
12     Generate an expression ρ for matching C;
13     foreach s that match ρ in g do
14       for i = 1 to m do
15         for j = 1 to n_i do
16           Top-Down(⟨C_i : τ_i^j⟩,s,v_i^j)
17         end
18         v_i ← {v_i^j | v_i^j ≠ ∅, j = 1, ..., n_i};
19       end
20       t ← ⟨C1 : v1, C2 : v2, ..., Cm : vm⟩;
21       T ← T ∪ {t};
22     end
23   fi
24 end
```

Figure 4.2: The top-down algorithm.

Despite its simplicity, this top-down extraction strategy works well with pages that are well structured (i.e., that are data-rich and present none or little variation in their structure). Indeed, several works in the literature [Adelberg, 1998; Baumgartner *et al.*, 2001; Muslea *et al.*, 2001] deal with the extraction of complex objects presenting a hierarchical structure by using a top-down decomposition procedure similar to the algorithm of Figure 4.2. For pages with variable structure (which are quite common in the Web), a distinct *bottom-up* extraction strategy is more appropriate.

## 4.2 Bottom-up Extraction Strategy

This section presents the bottom-up extraction strategy. This extraction strategy recognizes and extracts atomic attribute values (i.e, avps that lie at the bottom of the hierarchical structure of a complex object), prior to the recognition of the object itself. The extracted avps are then used to assemble the object through a bottom-up composition operation. Before discussing the bottom-up algorithm, we introduce some necessary notation.

**Definition 10** *Let $\tau$ be a table scheme. We define an* ***occurrence list*** *a list $o_1, o_2, \ldots, o_n$, where each $o_i = C : v$ is an instance of a component $C : \tau_c$ of $\tau$. To any given $o_i = C : v$ we associate a value $\ell(o_i)$, which is called the* ***index*** *of the occurrence. Also, for any given $o_i = C : v$ we define the function $\mathcal{C}(o_i) = C$.*

Informally, the occurrence list stores values extracted from source Web pages and objects being assembled. The index provides the relative location of each object within the source page from where it originates.

Another important notion for our purpose is the sequencing of objects in the source page. This leads to the definition of a *sequence*, as follows.

**Definition 11** *Let $O$ be an occurrence list. We define a* sequence *in $O$ as any ordered subset $S = \langle o_1, o_2, \ldots, o_n \rangle$ $(n > 1)$ of $O$ such that: (1) $i < j$ iff $\ell(o_i) < \ell(o_j)$ and (2) there is no $o \in O$ such that $\ell(o_i) < \ell(o) < \ell(o_{i+1})$ $(i < n)$.*

In other words, sequences are simple occurrences of consecutive instances in $O$.

The description of the bottom-up algorithm is presented in Figure 4.3, while Figure 4.4 presents an example of the execution of this algorithm. Notice that, for this description, instead of following the productions of a tabular grammar, we consider the structure of a table scheme as a guide. This is only for the sake of convenience, since both representations are equivalent, as shown in Chapter 3.

The algorithm takes as input an oe-pattern $\langle \tau, \mathcal{P} \rangle$ and a Web page $g$. Initially, for each attribute or attribute list $A$ defined as a component of $\tau$, all avp-patterns $A : \rho$ are used to obtain all strings within the current page $g$ that match $\rho$. Each matching string $s$ is used to compose an atomic object $A : s$, for which the index value takes the position of the string $s$ in the page $g$. We call this step the *Extraction Phase* of the algorithm. At the end of the extraction phase, a list of occurrences $O$ corresponding to the extracted avps is obtained. These occurrences are then used to compose new objects in the *Assembling Phase* of the algorithm.

The first step in the assembling phase builds a list of atomic values using the avps generated in the extraction phase. This corresponds to the loop in Lines 16–24. Notice that such lists are identified as sequences of avps of list defined for the same attribute in the table scheme $\tau$ (Line 16). The avps that compose each list are removed from the occurrence list (Line 20) and replaced by a single occurrence representing the list as whole (Line 22). The index value for this occurrence is set to the smallest index value of its components (Line 23).

```
1  Bottom-Up-Extraction(⟨τ, P⟩:oe-pattern;g:web page)
2  begin
3    Extraction Phase:
4    foreach A such that A : [...; {atom}; ...] or A : [...; atom; ...] is component of τ do
5      foreach avp-pattern A : ρ do
6        foreach string s in page g that matches ρ do
7          Let l be the location of the string s in page g;
8          O ← O ∪ {A : s};
9          ℓ(A : s) ← ℓ;
10       end
11     end
12   end
13
14   Assembling Phase:
15   foreach sequence ⟨a1, a2, ..., ak⟩ (k ≥ 1) in O such that C(ai) = C
16             where C : [...; {atom}; ...] is a component of τ
18     do
19       O ← O − ⟨a1, a2, ..., ak⟩;
20       L ← {a1, a2, ..., ak};
21       O ← O ∪ {⟨C : L⟩};
22       ℓ(C : L) ← ℓ(a1);
23     end
24
25   while ‖O‖ > 1
26       foreach sequence ⟨o1, o2, ..., ok⟩ (k ≥ 1) in O such that
27                a) C(oi) ≠ C(oj) for every oi, oj and
28                b) C(oi) ∈ {C1, ... Cm} for every oi,
29                    where C : [...; (C1 : τ1, ..., Cm : τm); ...] is a component of τ
31         do
32           O ← O − ⟨o1, o2, ..., ok⟩;
33           T ← {o1, o2, ..., ok};
34           O ← O ∪ {⟨C : T⟩};
35           ℓ(C : T) ← ℓ(o1);
36         end
38       foreach sequence ⟨t1, t2, ..., tk⟩ (k ≥ 1) in O such that C(t1) = ... = C(tk) do
39         O ← O − ⟨t1, t2, ..., tk⟩;
40         S ← {t1, t2, ..., tp};
41         O ← O ∪ {⟨C : S⟩};
42         ℓ(C : S) ← ℓ(t1);
43       end
44   end
45 end
```

Figure 4.3: The bottom-up algorithm.

Next, the while loop of Lines 26–45 is responsible for iteratively assembling (nested) tables of increasing complexity. This assembling process requires two steps. The first, carried out by the loop of Lines 27–37, assembles tuples; and the second, carried out by the loop of Lines 39–44, groups tuples into tables. Tuples are identified as being sequences of occurrences that correspond to distinct columns of some table scheme that comprises the table scheme $\tau$. When a tuple is identified, it replaces all occurrences that comprise it, in a way similar to what was described above for atomic lists. Then, sequences of such tuples are replaced by sets of tuples, to form tables in the usual way. The assembling of tables stops when there are no more occurrences to be grouped. This occurs only when the outermost table is assembled.

We now provide an example of how the bottom-up strategy works for assembling a nested table. The assembly steps for this example are illustrated in Figure 4.4. In this figure, circles represent objects extracted or being assembled. Consider the instances of Author whose structure is given by Author:(Name,Book:(Title,Price)). Assembling such instances requires three assembling steps. In the first step, Book instances are assembled from avps of attributes Title and Price (obtained in the extraction phase). In the second step, Book instances related to a same instance of Author are collected together in a list (referred to as {Book}). In the third step, each one of those lists are combined with an instance of Name (previously extracted), to assemble Author instances. Notice that this order corresponds to a bottom-up traversal of the hierarchical structure of Book instances. In the figure, the labels $l_i$ correspond to the index value associated to each instance.

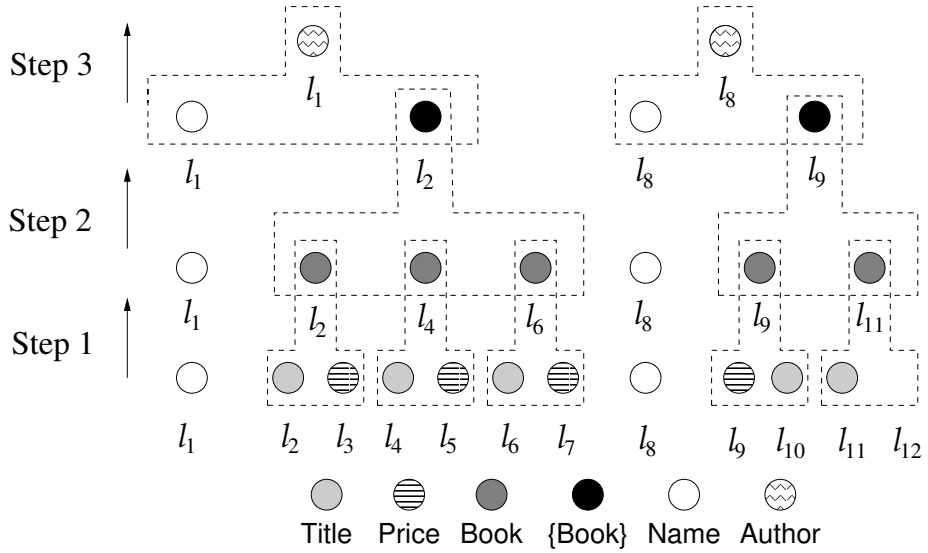



Figure 4.4: Execution of the bottom-up algorithm.

The lowest row of circles represents a set containing only avps. For these avps, the labels $l_i$ indicate the position in the source page of the string associated with the avp. If $j > i$ then $l_j > l_i$ (i.e., the list is ordered by the position $l_i$ of the string in the text).

Contiguous pairs of Title and Price values are combined to form Book instances. Each of these instances is labelled with the smallest index value. Notice that the Book instance labelled $l_9$ is built with components that appear in inverted order (relative to the order for previous objects). Also, the Book instance labelled $l_{11}$ misses its Price component. The capability of dealing with such situations is an important feature of our bottom-up strategy. The list of occurrences assembled after the completion of the first step is represented by the second row in Figure 4.4.

In the second step, runs of Book instances are grouped into lists of Book instances (indicated as {Book}). The third step simply combines these Book lists with Name instances (in our example, $l_1$ is combined with $l_2$ and $l_8$ is combined with $l_9$), as illustrated at the top of Figure 4.4.

The bottom-up algorithm is based on two fundamental assumptions. First, that avps can be correctly identified and extracted from a text (page), i.e., each avp-pattern determines a set of instances of an atomic attribute $A$. Second, that the presence of any component of an instance indicates the existence of such an instance. Therefore, if many of the avps correspond to incorrect strings (false positives), the assembling phase may form spurious objects. Further, if some values are not captured (false negatives), the assembly phase may create wrong complex objects containing, for instance, atomic objects that belong to other objects.

The problem of detecting false positives and false negatives in Web data extraction is, indeed, common to many approaches proposed on the literature. For instance, in [Kushmerick, 2000] the author describes a *corroboration algorithm* that uses simple domain-specific heuristics to verify the values extracted. In [Hsu and Dung, 1998] the authors propose the use of "negative examples" to make their extraction rules more effective.

In DEByE, many of the problems caused by imperfect avp-patterns can be alleviated by the features of the interface. That is, the user can provide new examples through feedback resources of the tool, change the estimated number of occurrences of instances in the source page, and mark some attributes as being mandatory in object instances. Obviously, there are cases for which this will not work at all. Further, it is frequently possible to build counter-examples that can break any heuristic one can devise. This is also the case of all the other approaches proposed on the literature. Thus, experimentation is a must to verify the spectrum of application of any semistructured data extraction algorithm. Our experimental results in Chapter 5 demonstrate that DEByE is an effective data extraction tool, which presents advantages (such as easiness of use, quick prototyping, and coverage of a variety of data sources with variations in structure) when compared to other approaches in the literature.

## 4.3 Top-down versus Bottom-up

The top-down strategy recognizes objects in their entirety. Thus, the recognition of partial objects (i.e., objects that are missing a component) and objects that contain components out of order require the generation of an specific extraction expression for each case. As

a consequence, to recognize partial objects, the top-down strategy depends on a potentially large set of example object patterns. To illustrate, consider that the user specifies as an example an object with two levels and three atomic components labelled $A_1$, $A_2$, and $A_3$. Retrieval of all possible partial matching objects would require seven distinct extraction expressions (one expression for the complete object, three expressions to indicate the absence of a single component, and three expressions to indicate the absence of two components). For large example objects, the number of cases might be exponential in the number of atomic components. This makes the top-down extraction procedure very inefficient in time (because each new page has to be processed independently for each extraction expression) and far less useful in practice.

The bottom-up strategy is more flexible than the top-down strategy because it assembles complex objects through a composition of simpler object components. Thus, this strategy is specially suitable for cases where missing components or components out of order are expected [Ribeiro-Neto *et al.*, 1999b]. This is corroborated by experimental results presented in Section 5.1. Because of this characteristic, we implemented the extractor module of the DEByE tool using the bottom-up strategy.

Figure 4.5 illustrates graphically how the bottom-up and the top-down strategies operate. The bottom-up strategy assembles complex objects through a composition of simpler component object matches, while the top-down strategy recognizes entire complex objects and decomposes them into simpler components.

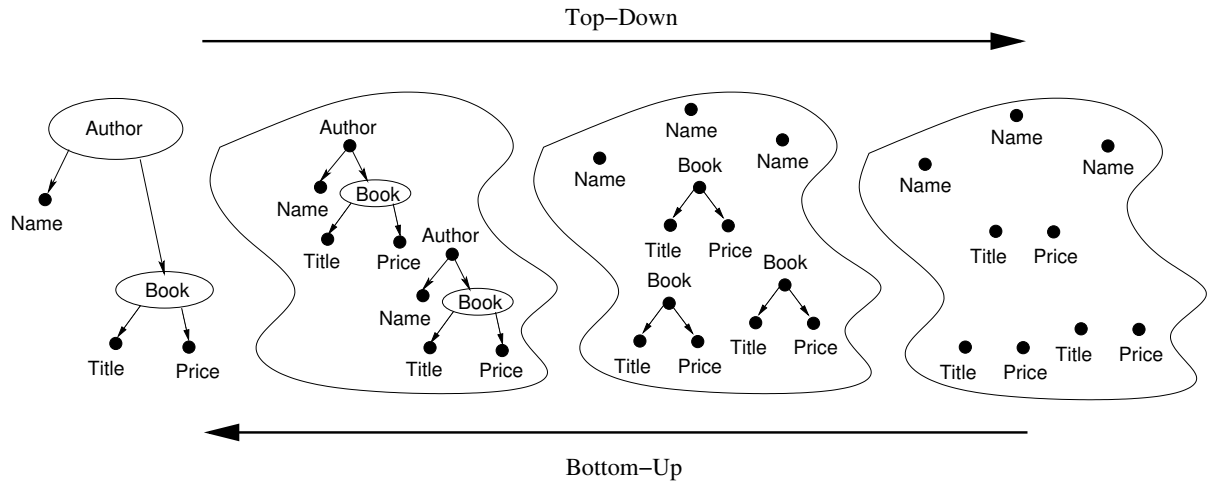



Figure 4.5: Comparison between the top-down and the bottom-up strategies.

## 4.4 Uncovering the Structure of Complex Objects

This section describes an alternative technique for obtaining complex objects that does not rely on the structuring productions of a tabular grammar. For this, all that is required

is a set of avp-patterns previously generated by identifying, in a sample page, sequences of atomic attribute values. These sequences are then processed by an algorithm we have developed, the *Hot Cycles* algorithm, that maps these sequences into a plausible structure building complex example objects with them.

The Hot Cycles algorithm first builds a graph for revealing frequent structural patterns (detected in the form of cycles) that resemble constructs, such as lists or tuples. The high frequency cycles, here called *hot cycles*, are used to map the list of avps into some tabular structure. The graph built by the Hot Cycles algorithm is a directed labelled *adjacency graph* that makes explicit use of adjacent occurrences of avps in the sample page. Each vertex in this graph corresponds to an attribute for the avps found in the source page. An arc connecting a pair of vertices for the attributes $A_i$ and $A_j$ is labelled with the number of avps of attribute $A_i$ that precedes avps of attribute $A_j$ in the page. Figure 4.6 illustrates a adjacency graph for a page similar to the page of Figure 1.1, but containing more objects.

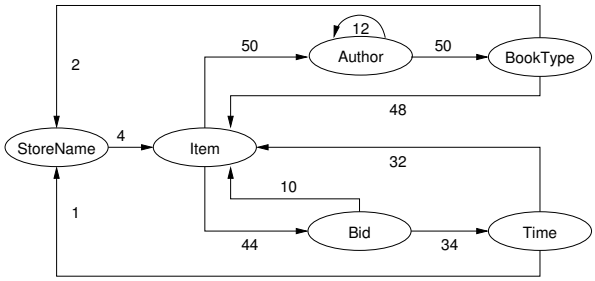



Figure 4.6: An example of an adjacency graph.

In the adjacency graph of Figure 4.6, we can see that 50 values of the attribute Item are found that preceded values of the attribute Author. We can also see that values of the attribute Author precede other values of the attribute Author 12 times, and so on. These concepts are more formally defined next.

**Definition 12** *Consider a set* $\{A_1 : s_1, \ldots, A_n : s_n\}$ *of avps extracted from a Web page* $g$*. Let* $L = \langle l_1, l_2, \ldots, l_n \rangle$ *be an ordered list of pairs* $l_i = \langle A_i, p_i \rangle$*, where* $p_i$ *is the location of the string* $s_i$ *in* $g$ *such that* $i > j$ *iff* $p_i > p_j$*. We call each* $l_i$ *an* ***occurrence*** *in* $g$ *and* $L$ *an* ***occurrence list*** *of* $g$*. Also, we say that* $l_i \prec l_j$*, that is,* $l_i$ *immediately precedes* $l_j$*, iff* $j = i + 1$*.*

**Definition 13** *Let* $L$ *be an occurrence list of a Web page* $g$*. An* ***adjacency graph*** *for* $g$ *is graph* $G = \langle V, N, \ell \rangle$*, where:* $V = \{A_i \mid \exists\ p_i(\langle A_i, p_i \rangle \in L\ )\}$ *is the set of vertices of* $G$ *and corresponds to the attributes in the elements of* $L$ *(one vertex for each attribute);* $N = \{\langle A_i, A_j \rangle \mid \exists\ p_i, p_j(\langle A_i, p_i \rangle, \langle A_j, p_j \rangle \in L$ *and* $\langle A_i, p_i \rangle \prec \langle A_j, p_j \rangle\ )\}$ *is the set of arcs of* $G$ *such that there is one such arc if at least one avp of attribute* $A_i$ *precedes an avp of attribute* $A_j$ *in* $L$*; and* $\ell(\langle A_i, A_j \rangle) = |\{\langle \langle A_i, p_i \rangle, \langle A_j, p_j \rangle \rangle \in L \times L \mid \langle A_i, p_i \rangle \prec \langle A_j, p_j \rangle\}|$ *is a*

*function that labels the arcs in $N$ with the number of avps of attribute $A_i$ that immediately precede avps of attribute $A_j$ according to $L$.*

The Hot Cycles algorithm operates iteratively in a bottom-up fashion. At each step, the algorithm examines the adjacency graph in an attempt to combine less complex components into more complex structures. This is accomplished by identifying cycles in the adjacency graph. When hot (or dominant) cycles are found, a piece of plausible structure has been uncovered. The less complex components in the hot cycle are then replaced by an element representing the new piece of structure uncovered. The algorithm then iterates and repeats this process.

To exemplify, in the graph of Figure 4.6, the algorithm looks for a cycle, such as [Item, Bid, Time, Item], formed by arcs with high valued numeric labels. Such a cycle signals that an aggregation of atoms was found. This aggregation can be represented by a tuple. As a second example, consider the situation in which the algorithm searches for a *loop* (i.e., a cycle from a node to itself) formed by a high value labelled arc. Such a loop signals an iteration, which can be mapped into a list (or an iterating production). In Figure 4.6, the loop [Author, Author] exemplifies this situation.

The Hot Cycles algorithm is described in Figure 4.7. Definition 14 introduces additional concepts required for properly understanding the algorithm.

```
1  Hot Cycles
2  begin
4    let L be an occurrence list.
5    Build an adjacency graph G = (V, N, ℓ) from L according to Definition 13.
7    while A ≠ ∅ do
8          foreach single-vertex cycle [A, A] ∈ G do
9            Replace the longest sequence ⟨A, p_i⟩, ..., ⟨A, p_{i+k}⟩(k ≥ 0) by a single element ⟨(A), p_i⟩ in L.
10         end
11         Adjust G to reflect the new value of L.
12         foreach hot cycle C = [A_1, ..., A_n, A_1] in G do
13           foreach sub-cycle C_j = [A_j, A_{j+1}, ..., A_{j+k}, A_j] (1 ≤ j, k ≤ n) of C do
14             foreach sequence S = ⟨A_j, p_j⟩, ..., ⟨A_{j+k}, p_{j+k}⟩ in L
15               Replace S by a single element ⟨(A_1, ..., τ_n), p_j⟩ in L.
16             end
17           end
19         end
20         Adjust G to reflect the new value of L.
21   end
```

Figure 4.7: The Hot Cycles Algorithm.

**Definition 14** *Let $G$ be an adjacency graph constructed from an occurrence list $L$ and let $C = [A_1, \ldots, A_n, A_1]$ $(n \geq 2)$ be a cycle in $G$. We define the* ***frequency*** *of $C$ as $f(C) = \ell(\langle A_n, \tau_1 \rangle)$. We say that $C$ is* ***sound*** *if there is at least one sequence $\langle A_1, p_1 \rangle, \ldots, \langle A_n, p_n \rangle$ in $L$. If there is a cycle $C_j = [A_j, A_{j+1}, \ldots, A_j]$ $(1 \leq j \leq n)$ in $G$, we say that $C_j$ is a*

***sub-cycle*** *of $C$. Note that $C$ is a sub-cycle of itself. A cycle $C$ of $g$ is said to be* ***hot*** *(or* ***dominant****) if it is sound and there is no sub-cycle of $C$ whose frequency is greater than $f(C)$.*

To exemplify such concepts, consider again the adjacency graph of Figure 4.6. If we examine this graph, we can see that the cycle $C_1 = [\mathsf{Item}, \mathsf{Bid}, \mathsf{Item}]$, with frequency 10, is a sub-cycle of $C_2 = [\mathsf{Item}, \mathsf{Bid}, \mathsf{Time}, \mathsf{Item}]$, with frequency 32. Furthermore, $C_2$ is a sub-cycle of $C_3 = [\mathsf{StoreName}, \mathsf{Item}, \mathsf{Bid}, \mathsf{Time}, \mathsf{StoreName}]$, that has frequency 1. Hence, $C_2$ is dominant (or hot), since it is "stronger" than $C_1$ and $C_3$. The hot cycle $C_2$ indicates that tuples of attributes Item, Bid and Time occur frequently. This is taken as an evidence that Item, Bid and Time form a plausible structure. The sub-cycle $C_1$ indicates that incomplete instances of such tuples occur, with missing values of Time, but this cannot be taken as the general case, since it is less often than $C_2$. The cycle $C_1$ is not as frequent as $C_2$, and we consider this as an indication that values of StoreName should not be aggregated with values of Item, Bid and Time. With respect to soundness, all cycles in the graph of Figure 4.6 are sound. However, unsound cycles can occur in adjacency graphs and we are forced to take them into account, since otherwise our algorithm would not work properly.

In what follows, we exemplify how the Hot Cycles algorithm works, using the example in Figure 4.6. In Figure 4.8 we show the successive graphs the algorithm generates, whereas Figure 4.9 contains a sequence of representations for the values of the occurrence list $L$. The representations for the values of $L$ are simplified in two directions. First, they contain just a short prefix of the values, since it would be infeasible to represent the full string that represents the sample Web page. Second, they use the letters S, I, B, T, A and K to indicate StoreName, Item, Bid, Time, Author and Book, respectively, and represent each pair $\langle A_i, p_i \rangle$ just by the type $A_i$.

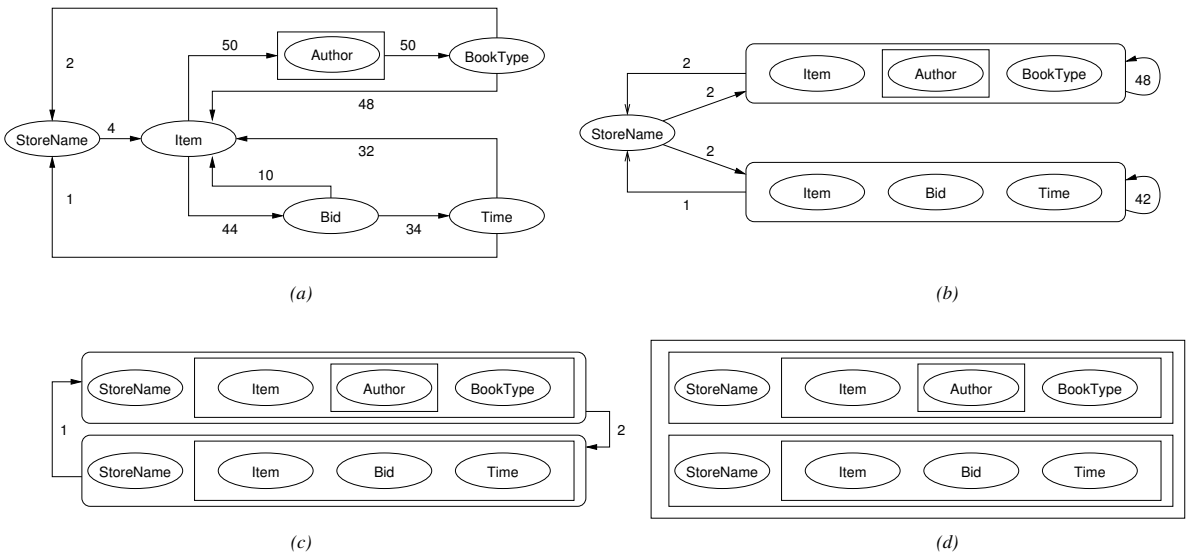



Figure 4.8: Illustration of the execution of the Hot Cycles Algorithm.

```
(a)  S IB IB IBT IBT IBT S IAK IAK IAAK IAK S ...
(b)  S IB IB IBT IBT IBT S I(A)K I(A)K I(A)K I(A)K S ...
(c)  S (IBT) (IBT) IBT IBT IBT S I(A)K I(A)K I(A)K I(A)K S ...
(d)  S (IBT) (IBT) (IBT) (IBT) (IBT) S I(A)K I(A)K I(A)K I(A)K S ...
(e)  S (IBT) (IBT) (IBT) (IBT) (IBT) S (I(A)K) (I(A)K) (I(A)K) (I(A)K) S ...
(f)  S ((IBT)) S (I(A)K) (I(A)K) (I(A)K) (I(A)K) S ...
(g)  S ((IBT)) S ((I(A)K)) S ...
(h)  (S ((IBT))) S ((I(A)K)) S ...
(i)  (S ((IBT))) (S ((I(A)K))) (S ...
(j)  ((S ((IBT))) (S ((I(A)K)))) ((S ((IBT))) (S ((I(A)K)))) ...
(k)  (((S ((IBT)))(S ((I(A)K)))))
```

Figure 4.9: Sequence of prefixes of simplified strings.

An initial version of the adjacency graph (Line 5 of the algorithm in Figure 4.7), shown in Figure 4.6, is first constructed. Next, a while loop (Lines 7–21) manipulates the occurrence list $L$ replacing atomic (or less complex) occurrences by more complex ones, according to the cycles found in the adjacency graph. At each iteration, the adjacency graph is reconstructed. The loop finishes when no more arcs are found in the graph, i.e., when there are no more occurrences to collapse.

The very first step in this loop is to replace all sequences of occurrences of the same attribute in $L$ by single occurrences that indicate lists (Lines 8–10). In our example, sequences of occurrences of Author will be replaced by a single occurrence, which indicates a list of values of Author. Figure 4.8(a) illustrates the new graph, and Figure 4.9(b) shows a prefix of the new value of $L$.

Next, hot cycles are identified (Line 12). There are two hot cycles in the graph of Figure 4.6: [Item, (Author), BookType, Item] and [Item, Bid, Time, Item]. Thus, all sequences of occurrences of Item, Author and BookType are replaced by a single occurrence, which indicates a tuple. The same applies to sequences of Item, Bid and Time, as well as to sequences of occurrences of just Item and Bid (Lines 13–17). Figures 4.9(c)-(e) show a prefix of the new values of $L$.

After we apply these changes to $L$, the adjacency graph is reconstructed (Line 20). The modified graph is illustrated in Figure 4.8(b). Notice that two distinct aggregations were introduced involving the same atomic type Item.

As this reconstructed graph still has arcs, the next iteration begins. Notice that this graph indicates (by an oval box with a loop) sequences of occurrences in $L$ that represent tuples of (Item,(Author),BookType). Thus, such sequences are replaced in $L$ by a single occurrence, which indicates lists of tuples (Lines 8–10). The same applies to sequences of tuples of (Item,Bid,Time). Figures 4.9(f)-(g) show a prefix of the new values of $L$.

Next, two hot cycles are found, [StoreName,(Item,(Author),BookType)] and [StoreName, (Item,Bid,Time)], which suggests that occurrences of StoreName must be aggregated to the lists of tuples previously generated. Thus, the occurrence list $L$ is modified and the adjacency graph $G$ is reconstructed accordingly. This is illustrated in Figure 4.8(c). In this figure, lists of tuples are indicated by square boxes and single tuples by an oval box. Figures 4.9(h)-(i) show the new values of $L$.

The last adjacency graph generated is illustrated in Figure 4.8(d). This graph depicts the aggregations, lists and nestings discovered by examining the atomic values found in the target page. Figure 4.9(k) shows the final value of $L$.

Notice that the main task accomplished by the Hot Cycles algorithm was converting the set of atomic attribute values in $L$ into a set of complex objects with a tabular structure. Indeed, at the end of the execution of the algorithm, $L$ is nothing more than a table instance whose rows are the complex objects assembled. Thus, the techniques described in Section 3.3 can be used to first derive a plausible scheme for this table and then map this scheme into a tabular grammar.

Intuitively, hot cycles uncover clues for structural formations left undeclared. The effectiveness of the Hot Cycles algorithm in assigning a plausible structure for a set of atomic attribute values was corroborated by several experiments we have performed with it. These experiments are reported in Chapter 5.

It is interesting to comment on what are the advantages of using the technique present in this section in comparison to relying on a user specification through the DEByE GUI. The Hot Cycles Algorithm is useful for applications where the user is not available for assembling example tables. In particular, if it is used in conjunction with a technique for automatically generating examples of attribute values, such as the one described in [Golgher *et al.*, 2001], it is very useful for dealing with the problems of wrapper adaptiveness and resilience. However, the structure assigned to objects by the Hot Cycles algorithm is a canonical one. Thus, it is possible that the suggested structure does not exactly match to a structure that would be specified by a user.

# Chapter 5

# Experimental Results

In this chapter, we present the results of experiments we carried out using the algorithms, techniques and extraction strategies developed in our work.

We begin by experimentally comparing the top-down and the bottom-up strategies for the extraction of complex objects. These experiments showed that, as expected, the bottom-up strategy is far superior than the top-down strategy for dealing with objects presenting structural variations.

Next, we present the results of experiments with the DEByE tool. For 15 Web data sources, including 3 of the most complex data sources in the RISE repository [Muslea, 1999], we fed the DEByE Extractor with oe-patterns generated using the DEByE GUI. The goal is to demonstrate the features of our bottom-up extraction strategy, as implemented in the DEByE tool, and the effectiveness of the whole approach. We analyzed various sample pages from each of the 15 sources used in our experiments and manually identified and counted the implicit objects in each of them. These objects were then used to verify the precision of our extraction procedure.

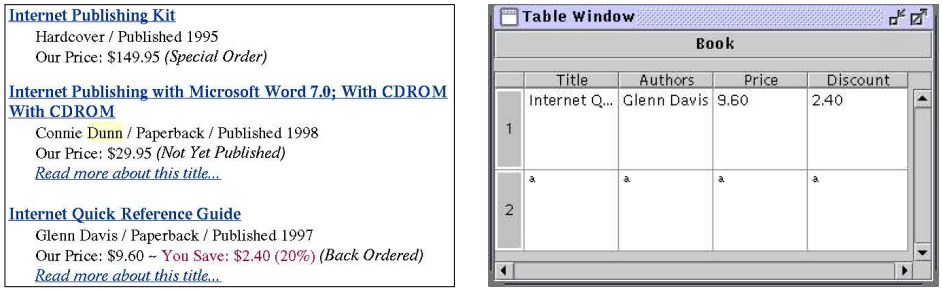
Finally, we report experimental results obtained with the Hot Cycles algorithm in the task of determining a plausible structure for objects from 24 Web data sources. For all of the sources considered, the algorithm succeeded in correctly uncovering plausible and reliable structures.

## 5.1 Comparison of the Top-down and Bottom-up Extraction Strategies

In this section, we draw a brief comparison between the top-down and the bottom-up strategies. This comparison consists of two parts: an experiment with objects presenting a flat structure and another experiment with objects presenting a nested structure.

For the first experiment, we use Web pages on books from the Amazon Web site, such as the one presented in Figure 5.1(a). Such pages are known to include objects which might be incomplete (i.e., some components might be missing). Using one of such pages as a sample, we assembled one single example object using the DEByE tool, as illustrated

in Figure 5.1(b). Then, we applied the top-down and bottom-up extraction strategies for extracting objects from a set of pages containing a total of 89 recognizable objects.




(a) (b)

Figure 5.1: A sample page (a) and an example table (b) for the Amazon Web site.

While the bottom-up strategy extracted 95% of them, the top-down strategy presented a poor extraction performance and was able to recognize only 28% of the objects. The main reason is that the objects might have different attributes which are missing. While the performance of the top-down strategy can be improved by increasing the number of example patterns, there is not much motivation to do so because the bottom-up strategy is already superior.

Now, consider again the same set of pages from the Amazon site and assume that the (implicit) objects on those pages are sorted according to their order of appearance. Consider this sorting as a ranking of the objects from these pages. Given this unusual ranking of the objects, we can plot curves of precision and recall [Baeza-Yates and Ribeiro-Neto, 1999] (in 11-standard recall levels) for the results of our bottom-up and top-down strategies. The curves are plotted as follows. Let $N$ be the total number of objects. We traverse the ranked objects starting with the first object and moving towards the last one. At the position $n$ of the ranking, we have traversed $n$ objects. For these $n$ objects, we count the number $\ell$ of objects that have been properly recognized by each extraction strategy. The precision $P$ indicates which percentage of the objects traversed has been recognized. The recall $R$ indicates which percentage of all objects has been recognized. Thus, the precision $P$ and the recall $R$ are given by: $P = \frac{\ell}{n}$ and $R = \frac{\ell}{N}$. The resulting precision and recall figures for the pages from Amazon are illustrated in Figure 5.2.

We first notice that the extraction performance of the top-down strategy deteriorates as it proceeds. This effect indicates that the top-down strategy fails to match objects early on and never recovers. The bottom-up strategy, on the other hand, is able to maintain high precision for levels of recall up to 80%. Then, its precision suddenly drops to zero. This indicates that the objects in the final sample pages have a distinct structure. To deal with this problem, we built a single additional example object (derived from one of the final sample pages). We then rerun our bottom-up strategy using now two example objects (the one that we had originally and the new one just built). The curve labelled Bottom-Up* illustrates the results which indicate a very nice improvement. In fact, the levels of precision are now very close to 100% for the various recall levels.

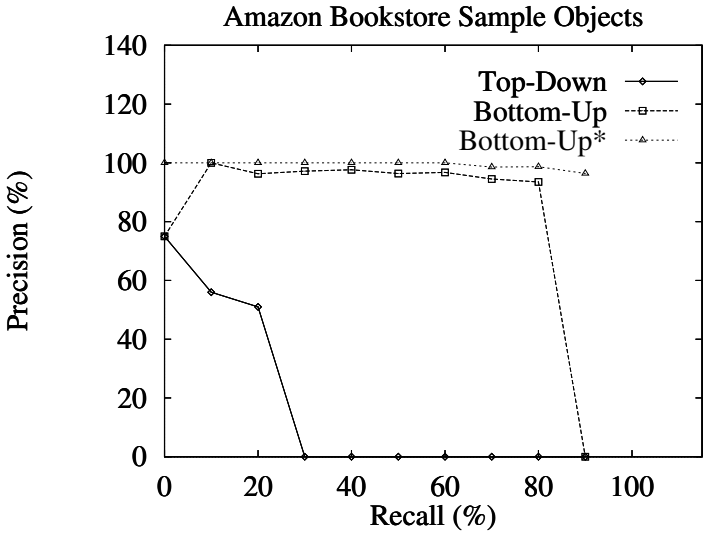



Figure 5.2: Precision and recall curves relative to extraction from Amazon Web pages.

For the second part of this comparison between the top-down and the bottom-up strategies, we use Web pages from ACM TODS at the DB&LP Web site. The objects in these pages have a two-level hierarchical structure as indicated in Figures 5.3(a) and 5.3(b). The example provided is shown in Figure 5.3(c).

To demonstrate the deficiencies of the top-down strategy in dealing with missing pieces of information in multi-level hierarchies, we deleted some component objects (particularly, Date and Pages) from the objects in our set of example pages. Following, we applied the top-down and bottom-up strategies on the modified pages. The results are summarized in Table 5.1.

| Object | Total | TD | BU |
|---|---|---|---|
| 1st. Level | | | |
| Volume | 20 | 15 (75%) | 20 (100%) |
| Number | 20 | 15 (75%) | 20 (100%) |
| Date | 15 | 15 (100%) | 15 (100%) |
| Edition | 20 | 15 (75%) | 20 (100%) |
| 2nd. Level | | | |
| Title | 76 | 45 (60.5%) | 76 (100%) |
| Author | 188 | 110 (58.5%) | 187 (99.5%) |
| Page | 66 | 45 (68.2%) | 66 (100%) |
| Article | 76 | 45 (60.5%) | 76 (100%) |

Table 5.1: Number of objects extracted by our top-down (TD) and bottom-up (BU) strategies for the DB&LP TODS pages.

We observe that, at the first level, there are 20 complex objects and that 5 of them do not include a Date component. Because of that, these 5 objects are not extracted by the

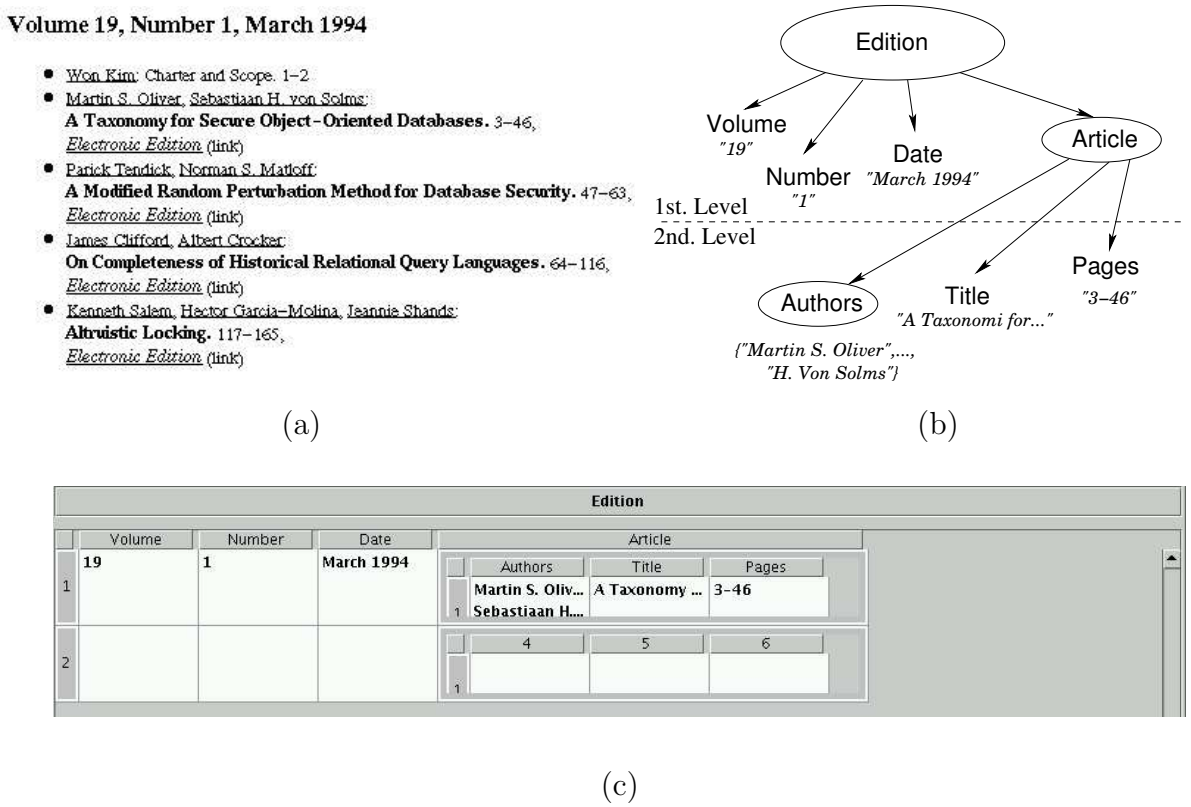



Figure 5.3: A sample page (a), the structure description (b), and an example table (c) for the DB&LP TODS pages.

top-down strategy. Additionally, the top-down strategy also fails to recognize second level components (generated by a decomposition operation) even after properly recognizing the first level components. The bottom-up strategy, on its turn, is able to compose complex objects, even when some of their component objects are missing in the Web source. As a result, it presents a very nice extraction capability.

## 5.2 Extraction Experiments with Representative Web Sources

This section presents a more comprehensive experimental evaluation carried out with the DEByE tool, using the bottom-up algorithm for implementing the Extractor module. We performed extraction experiments with 12 data-rich Web sources – the focus of our interest. From each one of these sources, we collected a subset of pages which we used to test DEByE. The majority of the sources are listed in the top positions in the list of the 100Hot Web

site[1] in their respective categories. The selected sites are the best ranked data-rich Web sites in their categories. In the immediate following, we briefly describe the main features of each Web data source used. To describe the structure of the objects found in the Web sources, we use a notation slightly different from the one introduced in Chapter 2, in which A:atom and L:{atom} are respectively written as A and {L}.

| | | |
|---|---|---|
| Placebo Without You I'm Nothing | $16.97 | $11.88 |
| Portishead Pnyc | $16.97 | $11.88 |
| Louis Prima Collectors Series | $11.97 | $8.38 |
| Queen Greatest Hits I & II | $29.97 | $20.98 |
| R.E.M. Up | $16.97 | $11.88 |

(a) CDNow

| Date | Location | Job Title | Company |
|---|---|---|---|
| Jun 7 | US-TX-Austin | Software Developer - Web | ESG Consulting |
| Jun 7 | US-CA-San Ramon | Web Interface Engineer - 99 Stock | IMS Net Corporation |
| Jun 7 | US-FL-Ft. Lauderdale | GREAT VC++/INTERNET OPPORTUNITY !!! | Maxim Group |
| Jun 7 | US-IN-Indianapolis | System Engineer | Wang Global |
| Jun 7 | US-New Jersey | Systems Analyst II | A-R-C (Alternative Resources) |
| Jun 6 | US-CA-San Jose | Senior Network Engineer | Infonent, Inc. |
| Jun 6 | US-WI-Milwaukee | System Engineer | Wang Global |
| Jun 6 | US-Illinois | Web Developer | IMI Systems |
| Jun 6 | US-IL-Chicago | System Engineer | Wang Global |
| Jun 6 | US-CA-San Francisco | Web Developer For Large Financial Site | IMI Systems |

(b) Monster

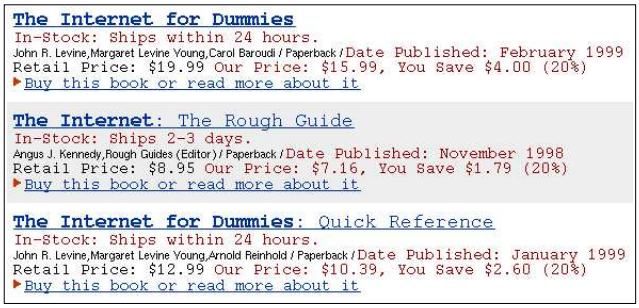
Figure 5.4: Excerpts of pages from CD Now and Monster.

## Web Data Sources

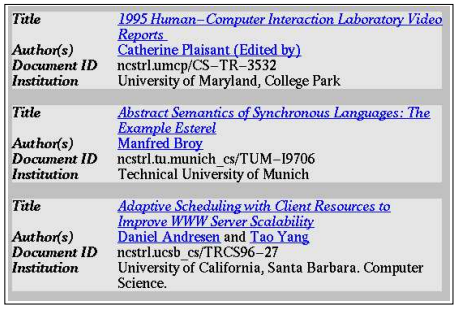
CDNow (`http://www.cdnow.com`). We collected the "30% discount" pages of the site. In these pages, the structure of the CD entries are very regular with no components missing (see Figure 5.4(a)). We modeled the CD entries in these pages as instances of the type CD:(ArtistName,Title,Price,Discount). All 219 CD entries in the source pages were recognized as instances of the CD type and extracted with just one example provided. CDNow is the 5th most popular site under the category *music* in the 100Hot ranking and is the best ranked CD store in this list.

Monster (`http://www.monster.com`). The Monster site is, according to the 100Hot ranking, the most popular site in the category *job*. We did a keyword search using "WWW" as an argument and used the first 500 job offer entries returned in our experiments. We modeled the entries in these pages as instances of the type Job:(Date,Location,Title,Company).

[1]This Web site (`http://www.100hot.com`) maintains a list of the most popular Web sites in many categories such as shopping, jobs, entertainment, etc. Whenever we refer to the popularity of a Web site in the 100Hot ranking, we are considering its position at the time the experiments were carried out.




(a) Barnes & Noble




(b) NCSTRL

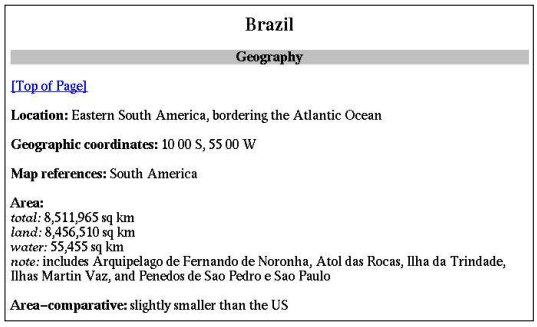

(c) Travelocity

Figure 5.5: Excerpts of pages from Barnes & Noble, NCSTRL, and Travelocity.

With just one example provided, we were able to retrieve all 500 entries as instances of this type. An excerpt of one of these pages is presented in Figure 5.4(b).

Barnes & Noble(`http://www.bn.com`). For this electronic bookstore, the second most popular site in the *book* category of 100Hot, we collected the set of pages returned by using the keyword "Internet". We used the first 200 entries returned in our experiments. An excerpt of one of these pages is shown in Figure 5.5(a). We modeled the book entries as objects of the type Book:(Title,Authors,Price). The entries present some variations in their structure (there are missing components in some of them), which required us to provide two separate examples. As a result, we were able to retrieve all the 200 book entries as instances of Book.

NCSTRL (`http://www.ncstrl.org`). The NCSTRL (Networked Computer Science Technical Reference Library) is a very popular repository of computer science technical reports from many institutions all over the world. We issued a query using "WWW" as a keyword and extracted a page with 177 entries. Some entries are shown in the excerpt of the page presented in Figure 5.5(b). The entries of these pages were modeled as instances of the type Report:(Title,AuthorName,Institution). One single example was provided. Despite the poor HTML formatting exhibited by the page, we were able to recognize all 177 entries as instances of Report.



**Brazil**

**Geography**

[Top of Page]

**Location:** Eastern South America, bordering the Atlantic Ocean

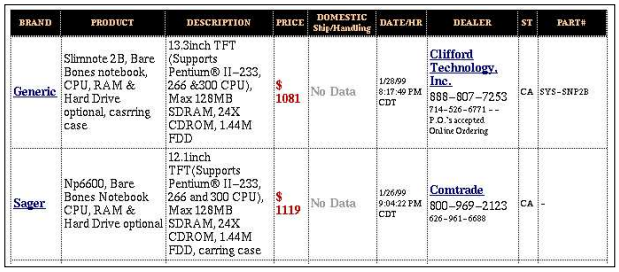
**Geographic coordinates:** 10 00 S, 55 00 W

**Map references:** South America

**Area:**
*total:* 8,511,965 sq km
*land:* 8,456,510 sq km
*water:* 55,455 sq km
*note:* includes Arquipelago de Fernando de Noronha, Atol das Rocas, Ilha da Trindade, Ilhas Martin Vaz, and Penedos de Sao Pedro e Sao Paulo

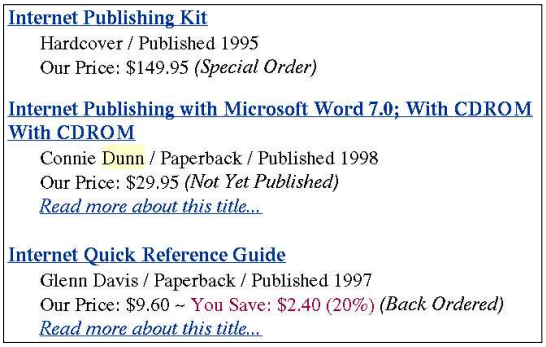
**Area-comparative:** slightly smaller than the US

(a) CIA Factbook



| BRAND | PRODUCT | DESCRIPTION | PRICE | DOMESTIC Ship/Handling | DATE/HR | DEALER | ST | PART# |
|---|---|---|---|---|---|---|---|---|
| Generic | Slimnote 2B, Bare Bones notebook, CPU, RAM & Hard Drive optional, casrring case | 13.3inch TFT (Supports Pentium® II-233, 266 &300 CPU), Max 128MB SDRAM, 24X CDROM, 1.44M FDD | $ 1081 | No Data | 1/28/99 8:17:49 PM CDT | Clifford Technology, Inc. 888-807-7253 714-526-6771 - - P.O.'s accepted Online Ordering | CA | SYS-SNF2B |
| Sager | Np6600, Bare Bones Notebook CPU, RAM & Hard Drive optional | 12.1inch TFT(Supports Pentium® II-233, 266 and 300 CPU), Max 128MB SDRAM, 24X CDROM, 1.44M FDD, carring case | $ 1119 | No Data | 1/26/99 9:04:22 PM CDT | Comtrade 800-969-2123 626-961-6688 | CA | - |

(b) Price Watch



Internet Publishing Kit
Hardcover / Published 1995
Our Price: $149.95 *(Special Order)*

Internet Publishing with Microsoft Word 7.0; With CDROM With CDROM
Connie Dunn / Paperback / Published 1998
Our Price: $29.95 *(Not Yet Published)*
*Read more about this title...*

Internet Quick Reference Guide
Glenn Davis / Paperback / Published 1997
Our Price: $9.60 ~ You Save: $2.40 (20%) *(Back Ordered)*
*Read more about this title...*

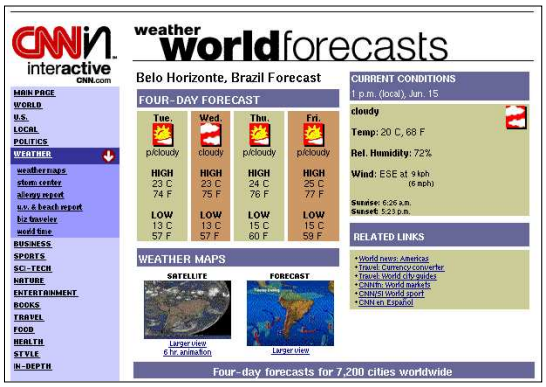
(c) Amazon

Figure 5.6: Excerpts of pages from CIA Factbook, Price Watch, and Amazon.

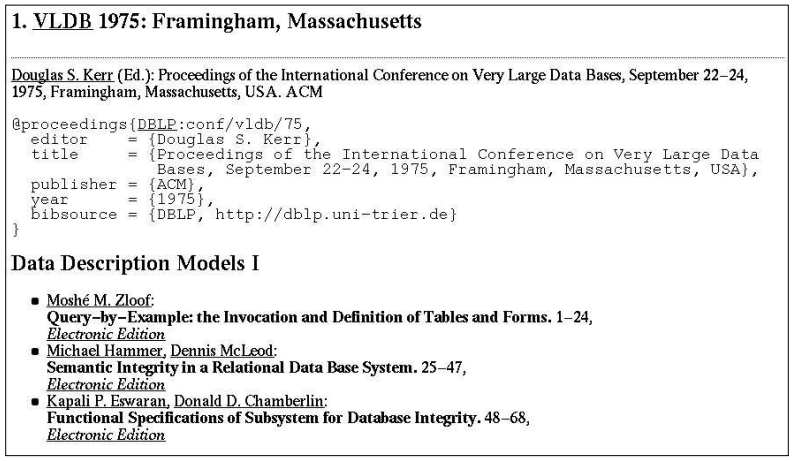
Travelocity (`http://www.travelocity.com`). We collected the page on vacation packages from the Travelocity site, ranked as the most popular site in the *travel* category of 100Hot. The page on vacations packages contained 162 entries. We modeled the objects as instances of the type Package:(Place,NoOfNigths,Price). One single example was provided. We intentionally ignored a free text description of each vacation package, present in each entry. For some of the entries, the information on the number of nights was missing. With this single example provided, all 162 entries were recognized as instances of Package and extracted. Figure 5.5(c) shows an excerpt of the Travelocity page.

CIA Factbook (`http://www.cia.gov/cia/publications/factbook/menugeo.html`). This site provides a detailed profile of 266 political entities, most of them countries. Every political entity has a profile, in the form of an HTML page, including information on geography, people, economy, government, etc., organized in the form of labelled fields. Figure 5.6(a) shows an excerpt of the page corresponding to the profile of Brazil. As there are hundreds of fields in each profile, we chose just a few of them which were modeled as instances of the type Country:(Name,Location,Coordinates,Area:(Total,Land,Water),Population, NationalCapital). We considered the first 50 profiles (in the order they appear in the main site's page)

in our experiments. With one single example given, we obtained 36 complete instances (72%), 13 incomplete instances (26%) (with one attribute missing in comparison with the profile), and 1 incorrect instance (2%) (which included an incorrect attribute instance). We then provided one additional example and obtained as a result 43 complete instances (86%), 6 incomplete instances (12%), and 1 remaining incorrect instance (2%). With two more examples (thus using a total of 4 examples), we got 49 complete instances (98%), but the incorrect instance remained. This instance corresponds to the profile of *Antarctica*, which has some features quite distinct from those found in the profile of the other countries.




(a) CNN World Whether



**1. VLDB 1975: Framingham, Massachusetts**

Douglas S. Kerr (Ed.): Proceedings of the International Conference on Very Large Data Bases, September 22–24, 1975, Framingham, Massachusetts, USA. ACM

```
@proceedings{DBLP:conf/vldb/75,
  editor    = {Douglas S. Kerr},
  title     = {Proceedings of the International Conference on Very Large Data
               Bases, September 22-24, 1975, Framingham, Massachusetts, USA},
  publisher = {ACM},
  year      = {1975},
  bibsource = {DBLP, http://dblp.uni-trier.de}
}
```

**Data Description Models I**

- Moshé M. Zloof:
  **Query-by-Example: the Invocation and Definition of Tables and Forms.** 1–24,
  *Electronic Edition*
- Michael Hammer, Dennis McLeod:
  **Semantic Integrity in a Relational Data Base System.** 25–47,
  *Electronic Edition*
- Kapali P. Eswaran, Donald D. Chamberlin:
  **Functional Specifications of Subsystem for Database Integrity.** 48–68,
  *Electronic Edition*

(b) VLDB page at DB&LP

Figure 5.7: Excerpts of pages from CNN World Whether and VLDB at DB&LP.

Price Watch (`http://www.prwatch.com`). The Price Watch site provides information on prices of computers and computer parts from several vendors. It occupies the 11th place in the category *hardware* of 100Hot, but it is the best ranked site that is not a site of a specific vendor. We collected a set of pages on Pentium II 266 Mhz notebooks. The

dblp .uni-trier.de

**Jeffrey D. Ullman**

List of publications from the DBLP Bibliography Server

Homepage

| | | |
|---|---|---|
| | | **1999** |
| 199 | EE | Ramana Yerneni, Chen Li, Jeffrey D. Ullman, Hector Garcia–Molina: Optimizing Large Join Queries in Mediation Systems. ICDT 1999: 348–364 |
| 198 | | Jeffrey D. Ullman: Some Advances in Data–Mining Techniques (Abstract). NGITS 1999: 1 |
| 197 | EE | Ramana Yerneni, Chen Li, Hector Garcia–Molina, Jeffrey D. Ullman: Computing Capabilities of Mediators. SIGMOD Conference 1999: 443–454 |
| 196 | | Alon Y. Levy, Anand Rajaraman, Jeffrey D. Ullman: Answering Queries Using Limited External Query Processors. JCSS 58(1): 69–82 (1999) |
| | | **1998** |
| 195 | EE | Shalom Tsur, Jeffrey D. Ullman, Serge Abiteboul, Chris Clifton, Rajeev Motwani, Svetlozar Nestorov, Arnon Rosenthal: Query Flocks: A Generalization of Association–Rule Mining. SIGMOD Conference 1998: 1–12 |
| 194 | EE | Chen Li, Ramana Yerneni, Vasilis Vassalos, Hector Garcia–Molina, Yannis Papakonstantinou, Jeffrey D. Ullman, Murty Valiveti: Capability Based Mediation in TSIMMIS. SIGMOD Conference 1998: 564–566 |
| 193 | EE | Min Fang, Narayanan Shivakumar, Hector Garcia–Molina, Rajeev Motwani, Jeffrey D. Ullman: Computing Iceberg Queries Efficiently. VLDB 1998: 299–310 |

(a) Author page at DB&LP

**Agatha Christie** — *The Adventure of the Christmas Pudding* — $5.95; *The Hound of Death* — $8.95; *Miss Marple's Final Cases* — $8.95; *Poirot's Early Cases* — $8.95; *Problem at Pollensa Bay* — $8.95; *The Mary Westmacott Collection* (as Mary Westmacott; includes: *A Daughter's a Daughter*; *The Burden*; *The Rose and the Yew Tree*)

**Dorothy Dunnett**—*Niccolo Rising*—$15.95; *Race of Scorpions*—$15.95; *Scales of Gold*—$15.95; *The Spring of the Ram*—$15.95
*Missing Person* (1993)—$11.95; *No Fixed Abode* (1994)—$11.95; *Identity Unknown* (1995)—$13.95

**Dick Francis**—*The Sport of Queens* (autobiography)—$10.95

(b) Murder by The Book

Figure 5.8: Excerpts of pages from (a) an Author Page at DB&LP and (b) a page from Murder by the Book.

resulting pages are formatted as HTML tables with one row for each item. Figure 5.6(b) shows an excerpt of one of these pages. There were a total of 113 items. Instances of the type Item:(Brand,Model,Description,Price,Date,Dealer:(Name,Phone),State,PartNo) were provided as examples of the objects found in this site. With one example, we extracted 57 complete items (50%) and 56 incomplete items (50%) (with one or two attributes missing in comparison with the item in the page). With two examples, we extracted 90 complete items (80%) and 23 incomplete items (20%) (with just one attribute missing in comparison with the item in the page).

Amazon (`http://www.amazon.com`). For this electronic bookstore, the most popular site in the *book* category according to 100Hot, we collected the pages satisfying the title search whose argument is the word "Internet". An excerpt of one of these pages is presented in Figure 5.6(c). As a rule, book entries present in the returned pages were very poorly

formatted, which is particularly true for the information on authors. Interestingly, the first pages resulting from this query present book entries with very regular structure, but the entries in the last pages are increasingly less regular and poorly formatted. This is an unexpected finding, since the pages are supposed to be ordered by the title of the books. The query returned a total of 4421 book entries in 89 pages. We analyzed 5 book entries in each page, to take into account the degradation in the structure of the entries. Thus a total of 445 entries were analyzed (our sample would be far more regular in structure if we had used the 445 best ranked entries). The examples were provided as instances of the type Book:(Title,Authors,Price) (as in the Barnes & Noble experiment). With one single example taken from the first page, 205 instances (46%) were completely extracted and 240 (54%) were extracted with missing attributes. Using an additional example, we obtained 289 complete instances (65%) and 156 incomplete (35%) instances. Increasing the number of examples to 3, 4, and 5, the fraction of instances extracted in their entirety also increased to 74%, 93%, and 96%, respectively.

CNN World Forecast (`http://www.cnn.com`). We collected weather forecast pages from the CNN site, one of the three most popular sites in the 100Hot *news* category, relative to 100 cities (i.e., we obtained 100 object instances). Figure 5.7(a) shows an excerpt of a city's page. We use the table scheme Forecast:(City,WeekDay:(Day,Temp:(High:(HC,HF)), Low:(LC,LF))) which reflects our interest in the maximum and minimum temperatures (in Celsius and Fahrenheit) for each day of a given week. With just one example, we were able to recognize and correctly extract the 100 instances of this type.

VLDB Pages at DB&LP (`http://dblp.uni-trier.de/db/conf/vldb`). We collected from the DB&LP site the pages with the contents of the VLDB proceedings from 1975 to 1983. There was one page per proceedings, except for the proceedings of the 1978's conference that had two pages. Examining just the page corresponding to the 1975 proceedings, illustrated in Figure 5.7(b), we specified a single example with the following type: Proceedings:(Year,Place,Section:(SecTitle,Article:(ArtTitle,{AuthorName},Pages))). The example for this type includes one instance of Proceedings, two instances of Section, and three instances of Article. We then used this single example to retrieve object instances from all the 9 source pages and obtained the following results. From 461 instances of Article manually identified, 424 (92%) were extracted with all their attributes and 37 (8%) were extracted with one or two attributes missing (values of AuthorName were totally extracted). From 144 instances of Section manually identified, 134 (93%) were extracted with all components and 10 (7%) were extracted with missing components. Note that this takes into account the components missing in instances of Article contained in a Section instance. From the 10 instances of Proceedings, 4 were completely extracted and 6 were extracted with missing components. Examining these results, we notice that most of the problems were found in objects related to the page of the 1976 proceedings and to the second page of the 1978 proceedings. This occurred because the 1976 proceedings were not organized in sections, thus having a structure very distinct from the example given, and because the second page of 1978 proceedings, although having the same structure, had very different

formatting features for the article entries.

Author Page at DB&LP (`http://www.dblp.de/db/indices/a-tree`). For this experiment, we took Professor Jeffrey Ullman's page at DB&LP. An excerpt of this page is in Figure 5.8(a). To capture the variations in the implicit structure of the objects in this page, we used a type definition the includes a variant as follows: Article:(Title,{AuthorName},HowPublished:[(Pages,Journal,Number);(Pages,Proceedings)]). We provided 5 examples of article entries. As a result, we were able to correctly extract 554 instances of AuthorNames (100%) and 196 complete instances of Article (98%). The remaining 3 Article instances (less than 2%) were extracted with a single missing attribute.

Murder by the Book (`http://www.mbb.com`). Murder by The Book is a bookstore specialized in mystery books. Its pages are very poorly formatted (for instance, the data on book prices is formatted in many different ways). For that reason, we chose this site as a particular case of interest. As in the previous experiment, for this site there was the need of variants for the type Author. For this, we provided example objects that were instances of the type Author:(Name,Book:[(Title,Price);(UnitPrice,{BookTitle})]). We then extracted object instances from a page (the "English Imports" page) containing 147 book entries grouped into 49 author entries. With respect to Book instances, we extracted 129 of them completely (89%) and 16 (11%) with missing attributes (Price), when 5 examples were provided. Regarding the 49 Author instances, we extracted 40 of them (82%) completely and 9 (18%) with some missing component, when 2 examples were provided.

## Analysis of our Results

Table 5.2 summarizes our results. The percentage figures for the number of objects extracted are relative to the total number of objects identified manually in the sources pages. The number of examples used in the extraction is determined by trial and error. When the number of examples is insufficient, the results of the extraction process include many objects with missing attribute values. It is then necessary to provide more examples to obtain more complete (and precise) results. The number of examples is increased until the quality of the results improves. In Table 5.2, we only show the points at which the results were improved considerably. For instance, for the Murder by the Book site, we only show results when 2 and 5 examples are provided. As we can see, the DEByE tool was able to recognize and retrieve most objects in the given pages.

In Table 5.3 we present the time (in seconds) spent generating oe-patterns (i.e., tabular grammars), extracting avps (i.e., atomic values), and assembling complex objects using the extracted avps. These figures refer to the extraction carried out using the number of examples shown in Table 5.2. The column Size refers to the size of the sample page used to generate the avp-patterns and the column Total Size refers to the total size (in bytes) of the pages from which the complex objects were extracted. The column Total indicates the total time spent in each case. As we can observe, the whole extraction procedure can be completed in a matter of seconds, even for complex sites.

| Web Source | Object Type | Total | Examples | Objects Extracted | |
|---|---|---|---|---|---|
| | | | | Complete | Incomplete |
| CDNow | CD | 219 | 1 | 219 (100%) | – |
| Monster | Job | 500 | 1 | 500 (100%) | – |
| Barnes & Noble | Book | 200 | 2 | 200 (100%) | – |
| NCSTRL | Report | 177 | 1 | 177 (100%) | – |
| Travelocity | Package | 162 | 1 | 162 (100%) | – |
| CIA Factbook | Country | 50 | 1 | 36 (72%) | 13 (26%) |
| | | | 2 | 43 (86%) | 6 (12%) |
| | | | 4 | 49 (98%) | – |
| Price Watch | Item | 113 | 1 | 57 (50%) | 56 (50%) |
| | | | 2 | 90 (80%) | 23 (20%) |
| Amazon | Book | 445 | 1 | 205 (46%) | 240 (54%) |
| | | | 2 | 289 (65%) | 156 (35%) |
| | | | 3 | 327 (74%) | 118 (26%) |
| | | | 4 | 417 (93%) | 28 (7%) |
| | | | 5 | 429 (96%) | 16 (4%) |
| CNN | City | 100 | 1 | 100 (100%) | – |
| | Weekday | 400 | 1 | 400 (100%) | – |
| VLDB Pages at DB&LP | Proceedings | 10 | 1 | 4 (40%) | 6 (60%) |
| | Section | 144 | 2 | 134 (93%) | 10 (7%) |
| | Article | 461 | 3 | 424 (92%) | 37 (8%) |
| Author Page at DB&LP | Article | 196 | 5 | 199 (98%) | – |
| | Author | 554 | 12 | 554 (100%) | – |
| Murder by the Book | Author | 49 | 2 | 40 (82%) | 9 (18%) |
| | Book | 147 | 5 | 130 (89%) | 16 (11%) |

Table 5.2: Number of objects extracted by DEByE for various Web sources.

Although we have limited our experiments to a few hundred objects of each type, it is worth noting that almost all sites considered contain a large amount of pages which have the same overall structure and formatting features. Thus, many more objects could have been extracted from these sites. For instance, for the Amazon pages, we recall that the number of objects actually extracted (4421) was equal to the total number of book entries extracted by the query, although we have analyzed only 445 book entries.

As a final observation, the fact that our extraction strategy was very effective with data available in popular Web sites (such as the ones we used in our experiments) confirms our hypothesis on how data is usually made available on the Web. In fact, the pages of many very popular and interesting Web sources present an inherent implicit structure that can be recognized and modeled accurately. Furthermore, data contained in these pages are usually surrounded by contextual information that allows their effective recognition and extraction.

| Web Source | oe-pattern Generation | | Object Extraction | | | Total (sec.) |
|---|---|---|---|---|---|---|
| | | | Time (sec.) | | | |
| | Time (sec.) | Size | avp | Assembling | Total Size | |
| CDNow | 0.223192 | 31654 | 0.395016 | 0.095113 | 301892 | 0.713321 |
| Monster | 0.683688 | 39521 | 0.397513 | 0.233642 | 363686 | 1.314843 |
| Barnes & Noble | 0.526398 | 51565 | 2.782157 | 0.100536 | 1342861 | 3.409091 |
| NCSTRL | 3.200000 | 160531 | 0.221553 | 0.094289 | 160531 | 3.515842 |
| Travelocity | 0.311210 | 76067 | 0.345100 | 0.048700 | 76067 | 0.705010 |
| CIA Factbook | 3.004563 | 42045 | 0.364568 | 0.051170 | 4288460 | 3.420301 |
| Price Watch | 1.520407 | 17460 | 0.325296 | 0.149700 | 151488 | 1.995403 |
| Amazon | 1.917623 | 33551 | 7.242873 | 1.497996 | 2613915 | 10.658492 |
| CNN | 0.523155 | 32127 | 0.165441 | 0.107794 | 727400 | 0,796390 |
| VLDB Pages at DB&LP | 0.942920 | 18173 | 0.702879 | 0.573590 | 532403 | 2.219389 |
| Authors Page at DB&LP | 3.601458 | 73814 | 0.220965 | 0.229656 | 73814 | 4.052079 |
| Murder by The Book | 0.338172 | 10853 | 0.041062 | 0.058324 | 10853 | 0.437558 |

Table 5.3: Time spent generating oe-patterns, extracting avps, and assembling complex objects in the extraction experiments.

## 5.3 Extraction Experiments with Sources from the RISE Repository

*RISE (Repository of Online Information Sources Used in Information Extraction Tasks)* is a repository of online information sources that were used for data extraction experimentation by various authors [Muslea, 1999], most of them from the machine-learning community. Among these, we distinguish the research on the WIEN [Kushmerick, 2000], Stalker [Muslea *et al.*, 2001] and SoftMealy [Hsu and Dung, 1998] systems. Our purpose here is to provide a preliminary comparison of DEByE with these three other systems, using data sources from the RISE repository.

In Table 5.4 we present experimental results obtained by running DEByE on three of the most complex data sources in RISE: Okra, BigBook, and IAF. For comparison, we also present published results obtained for these three sources by WIEN, Stalker and SoftMealy tools. We first observe that, contrary to the other tools, WIEN takes whole pages as examples. Thus, when considering the number of examples used by WIEN, we take the number of pages given as examples and multiply it by the average number of objects in each page. This is what Table 5.4 shows. We further observe that these three sources include only pages containing flat (i.e., single level) objects, contrary to most sources we use in Section 5.2. As a result, DEByE was as effective as any of the other three tools for these three data sources, while frequently using a smaller number of examples and always requiring less user effort for the specification of examples. Further, DEByE is conceptually simpler than the other three tools and requires a less complex implementation. Because of that, it tends to be faster on conventional Web data sources.

While the results in Table 5.4 shed some light on the relative performance of DEByE, they do not allow a full and direct comparison between DEByE and the three other tools.

| | DEByE | | WIEN | | Stalker | | SoftMealy | |
|---|---|---|---|---|---|---|---|---|
| Web Source | Ex. | Extracted | Ex. | Extracted | Ex. | Extracted | Ex. | Extracted |
| OKRA | 2 | 100% | 46 | 100% | 1 | 97% | 1 | 100% |
| BigBook | 1 | 99% | 274 | 100% | 8 | 97% | 6 | 100% |
| IAF | 3 | 99% | – | – | 10 | 85% – 100% | 1 | 99% |

Table 5.4: Results of experiments with the DEByE *Extractor* for RISE Web sources.

There are several reasons. First, WIEN, Stalker and SoftMealy are all based on machine-learning techniques, while DEByE is based on the identification of context through passage analysis [Baeza-Yates and Ribeiro-Neto, 1999; Callan, 1994; Kaszkiel and Zobel, 1997], a well known information retrieval technique. Second, the experimental protocol used by the three other tools was much more mechanical than ours. While in WIEN, Stalker and SoftMealy the examples where randomly chosen in several trials (30 for WIEN, 500 for Stalker and 300 for SoftMealy), in DEByE the users choose the examples at their will. In fact, a key point in the DEByE approach is to extract objects according to the users' preferences. The fact that most often the users are able to specify a useful example in the first trial indicates the validity of the DEByE approach. Third, the goals of the experimentation performed were distinct in the case of each tool. In the WIEN experiments, the goal was to obtain 100% accuracy in the extraction. Thus, the number of examples was increased until this accuracy level was reached. In Stalker, the authors stopped the experiments after reaching 97% of accuracy or after 10 examples were given. SoftMealy, as DEByE, reached almost perfect accuracy with few examples, so that no limits needed to be imposed on the accuracy and in the number of examples given. However, in the experiments with SoftMealy, the authors used three alternative extraction strategies, while in DEByE the same algorithm was used for the three sources. This is an important point. In DEByE, there is no knowledge base or set of heuristics to guide the extraction procedure. All variations found are treated by the same extraction algorithm, using solely the evidences presented in the user specified examples. Fourth, in DEByE we used only the sets of pages available from RISE (which were also used in the experiments with WIEN and Stalker) while in the experiments with SoftMealy the authors used an extended set with many more pages for each source.

Examining Table 5.4, we observe that the number of examples required by DEByE for each source does not vary much, as it occurs with the other tools. This may be explained by the fact that, in DEByE, we only generate patterns for extracting single atomic values that are assembled afterwards, according to the structure of the examples provided. As we have already discussed, assembling objects in DEByE does not rely on the ordering of the component objects. As a result, there is no need to generate alternative patterns for capturing distinct ordering of components.

These results with the RISE repository suggest that the DEByE approach is as effective as the known alternatives based on wrapper induction.

| # | Source | L | MA | NA | Var | Scheme |
|---|---|---|---|---|---|---|
| 1 | CDNow | 1 | No | 4 | No | (ArtistName,Title,Price,Discount) |
| 2 | Monster | 1 | Yes | 4 | No | (Date,Location,Title,Company) |
| 3 | Barnes & Noble | 1 | Yes | 4 | No | (Title,Authors,Price,RetailPrice) |
| 4 | NCSTRL | 2 | Yes | 3 | No | (Title,{AuthorName},Institution |
| 5 | Travelocity | 1 | Yes | 3 | No | (Place,NoOfNigths,Price) |
| 6 | CIA Factbook | 1 | Yes | 8 | No | (Name,Location,Coord,Area,Land,Water,Pop,Capital) |
| 7 | Price Watch | 2 | Yes | 9 | No | (Brand,Model,Desc,Price,Date,(Dealer,Phone),State,Part) |
| 8 | Amazon | 1 | Yes | 3 | No | (Title,Authors,Price) |
| 9 | CNN | 3 | No | 6 | No | (City,Day,((HighC,HighF),(LowC,LowF))) |
| 10 | VLDB | 4 | No | 6 | No | (Year,Place,(SectionName,(Title,{Author},Pages))) |
| 11 | Authors | 3 | Yes | 6 | Yes | [(Title,{Author},Pages,Journal,Number); (Title,{Author},Pages,Proceedings)] |
| 12 | MBB | 3 | Yes | 2 | Yes | [(Author,(BookTitle,Price)); (Author,(UnitPrice,{BookTitle}))] |
| 13 | Okra | 1 | No | 4 | No | (Score,Name,Email,FirstEntered) |
| 14 | BigBook | 1 | No | 6 | No | (Name,Address,City,State,AreaCode,Phone) |
| 15 | IAF | 1 | Yes | 6 | No | (Name,Email,Nick,Update,Organization,Provider) |
| 16 | LA Weekly | 2 | Yes | 5 | No | (Name,Address,Phone,Review,{CreditCard}) |
| 17 | Amazon/Cars | 2 | No | 5 | No | (BrandHead, Type, (Model,Brand,PriceRange)) |
| 18 | Buy/Product | 1 | No | 8 | No | (Product,OurPrice,Price,Save,Avail,Type,Platform,Part) |
| 19 | Buy/Subcategory | 2 | No | 6 | No | (Category,(Product,Vendor,Platform,Media,Price)) |
| 20 | RPM/Distribution | 2 | No | 4 | No | (DistributionName,(RPM,Description,Distribution)) |
| 21 | RPM/Maintainer | 2 | No | 4 | No | (Provider,(RPM,Description,Distribution)) |
| 22 | UEFA/Teams | 1 | No | 9 | No | (Country,Association,FoundedIn,UEFA-Aff,FIFA-Aff,President,Gen-Sec,Press-Off,Coach) |
| 23 | UEFA/Playes | 2 | Yes | 2 | No | (Country,{Players}) |
| 24 | Rise/PharmaWeb | 1 | Yes | 13 | No | (Faculty,Univ.,Att,Address,City,ZIP,Province,Country, Phone,Fax,URL,Update,UpBy) |

Table 5.5: Results of the experiments with the Hot Cycles algorithm

## 5.4 Experiments with the Hot Cycles Algorithm

In this section, we present the results of experiments we carried out using an implementation of the Hot Cycles algorithm. The goal was to verify if indeed the algorithm was able to suggest a plausible structure for a given data source from which we wish to extract data. For this, besides the 15 data sources used in the experiments of Section 5.2, we also consider a set of Web sources used by RoadRunner, a well know data extraction tool recently presented [Crescenzi *et al.*, 2001].

For each source, we first provided examples of attribute values to be extracted, which are used for generating the corresponding avp-patterns. Next, these avp-patterns were used to extract all atomic values from a set of pages of the source. The resulting list of avps was then input to the Hot Cycles algorithm.

Table 5.5 lists the sources used in the experiments, along with a summary of the results obtained. This table presents the number of levels (L) and the number of attributes (NA) found by the Hot Cycles algorithm. It also describes whether the algorithm identified

missing attributes (MA) and variants (Var). The last column in this table (Scheme) describes the table scheme generated by the Hot Cycles algorithm (according to the notation introduced in Section5.2), for each Web source.

Inspecting the tabular schemes found by the algorithm, we see that these schemes match exactly the schemes originally proposed in Section 5.2 and by others researchers [Kushmerick, 2000; Muslea *et al.*, 2001] for sources 1 to 16. For the remaining sources (17 to 24), which were originally proposed in [Crescenzi *et al.*, 2001] for the experiments with the RoadRunner extraction tool, there were no target schemes defined. Thus, we compared our results with the structure of the nested tables available in the RoadRunner Web site[2]. In some cases we disregarded non-meaningful columns generated by RoadRunner, which contain, for instance, headers and footers of pages. This comparison shows that the table schemes generated match the structure of the RoadRunner results.

Thus, for all the 24 Web data sources we considered, the Hot Cycles algorithm correctly uncovered plausible and reliable structural formations. This suggests that identification of hot cycles is a promising direction towards reliable automated Web data extraction.

[2] `http://www.dia.uniroma3.it/db/RoadRunner`

# Chapter 6

# Conclusions and Future Work

In this work we have proposed, implemented and evaluated strategies and techniques that address the problem of extracting semistructured data from Web data sources, within the context of the DEByE approach. The results we have reached assign to this approach a number of features that are very important for Web data extraction. They have been used in the implementation of the DEByE tool and have their effectiveness verified through experiments. In this chapter we present our conclusions and discuss directions for future work.

## 6.1 Conclusions

DEByE falls into the category of semi-automatic data extraction approaches, since the role of the user is limited to providing examples of the data to be extracted. Indeed, this is a most desirable feature, because it shields users (i.e., wrapper developers) from being aware of specific formatting features of the target pages. Although fully automated approaches have been proposed in the literature, such approaches usually have a great dependency on specific features of HTML, which does not happens with DEByE. Further, fully automated approaches have difficulties in selecting data of interest mixed with uninteresting pieces of data occurring in target pages.

The idea of using examples is not new and has been applied in other state-of-the-art data extraction approaches in the literature [Hsu and Dung, 1998; Kushmerick, 2000; Muslea *et al.*, 1999]. However, in all of these approaches, the extraction process relies on an implicit knowledge or on a separate description of the structure of the target Web page. In DEByE, the examples provided by the users describe the structure of the objects being extracted (which, frequently, constitutes a small portion of the structure of the whole page). This adds great flexibility to the extraction process and provides a natural and efficient way of handling nested objects and structural variations, that is, to the best of our knowledge, unique to our approach.

To allow users to specify complex multi-level objects as examples, the DEByE tool adopts nested tables, which are simple, intuitive, and can be used effectively for modeling

data available in distinct Web sources. In fact, even early experiments with the DEByE tool GUI have demonstrated the effectiveness of nested tables for the process of example specification [Silva, 1999; Laender *et al.*, 2000].

Actually, nested tables constitute the fundamental data representation paradigm within the DEByE approach, since they are also used for manipulating and storing the data extracted from target pages. To deal with typical variations of semistructured data, the concept of nested table had to be extended by relaxing the original assumption that all values in a column should have a same internal structure. Indeed, we regard this extensions as an interesting result, since it provided a natural and elegant solution for representing complex semistructured data. Latter on, we have verified the effectiveness of such tables not only for representing data extracted from the Web [Laender *et al.*, 2000; 2002b], but also for querying [da Silva *et al.*, 2002] and storing [Magalhães *et al.*, 2001a] such data.

Based on this extended form of nested tables, we have formalized the concept of wrappers by means of tabular grammars. Such context-free grammars are formed by productions that lead to parse trees that can be directly mapped to nested tables. We have developed strategies for generating tabular grammars from a set of example objects provided by a user from a sample page. This includes: (1) the generation of extracting productions from single values identified by the user as belonging to a specific domain (e.g., an item description, a price, etc.) and (2) the generation of structuring productions by capturing the structure of the objects to be extracted from the nested tables assembled by the users.

The extraction of data from target pages is accomplished by parsing these pages using a tabular grammar. For this parsing process, we have developed an efficient bottom-up strategy. This strategy includes two distinct phases: an extraction phase, in which atomic attribute values are extracted based on local context information available in the extraction productions, and an assembling phase, in which such values are assembled to form complex objects according to the target structure supplied by the user through examples, which is encoded in the structuring productions. The bottom-up strategy is one of our most important results, since all previous similar wrapper generation approaches in the literature adopt a top-down decomposition procedure for data extraction. As we have experimentally demonstrated, the bottom-up strategy is far superior than the top-down strategy for dealing with multi-level objects presenting structural variations.

The general principle used by the bottom-up algorithm, that is, first extracting atomic values and then grouping these values to assemble complex objects, has been further exploited by the Hot Cycles algorithm we have developed. This algorithm aims at uncovering a plausible tabular structure for assembling complex objects with a given set of atomic values extracted from a target page. Although not integrated into the DEByE tool, the Hot Cycles algorithm is entirely based on the framework of the DEByE approach. We regard this algorithm as another important result of our work, since it can be used for adapting the approach to applications where the user is not available for assembling example tables.

# Future Work

## Additional Strategies for Dealing with Ambiguous Context

Although our extraction strategies work well for a variety of situations commonly found in Web sources, they can be improved, both in efficacy and in efficiency, for some specific situations of ambiguous local context.

As an example we may cite the case of pages containing tables with a large number of columns (e.g., more than 10). In such pages, each column in a table is generally regarded as a type. The problem is that it is most likely that data on two or more distinct column share large common contexts (i.e., column delimiters), what leads to a great computational effort in establishing a context that is selective enough for each column (or type) in the table. As tables are often of interest for users, we may work on specific heuristics for dealing with such situations.

Another important example of situation of ambiguous context we plan to better address are large text portions, such as movie reviews, newspaper news, etc., which in some cases are not surrounded by distinctive textual context (e.g., markups, keywords, etc.).

## Incorporating the Hot Cycles Algorithm in the DEByE Tool

A natural step to follow from the current stage of our work is to incorporate the Hot Cycles algorithm into the DEByE Tool. This would offer to users the option of simply specifying attributes by selecting atomic values from the target page and asking the tool to suggest a plausible structure for accommodating them. In this case, it would be important to allow users to revise the suggested structure according to their needs.

## Extraction Improvement Through Corroboration

As many other approaches in the literature, our extraction strategies are also subject to the occurrence of false positives (i.e., data wrongly extracted) and false negatives (i.e., data that should be extracted but that were not extracted). Currently, in DEByE, such problems can be alleviated by the features of the GUI of the DEByE tool. That is, the user can provide new examples, change the estimated number of occurrences, and mark some attributes as being mandatory in object instances.

However, there are some post-extraction actions that can be automatically performed to deal with false positives and false negatives, which are called *corroboration* [Kushmerick, 2000]. For instance, in [Kushmerick, 2000], the author describes a *corroboration algorithm* that uses simple domain-specific heuristics to verify the values extracted. In [Hsu and Dung, 1998], the authors propose the use of "negative examples" to make their extraction rules more effective. A similar strategy is adopted in [Baumgartner *et al.*, 2001].

In our work we plan to incorporate some corroboration actions, in a way orthogonal to the extraction strategies, hoping to improve the quality of the extracted data in cases were it is needed. For instance, we plan to use statistical methods to identify average features

in the set of the objects extracted, so that objects that do not conform to these features (outliers) can be detected.

## Integrating DEByE Wrappers with Page Collecting Agents

An issue we do not directly addressed in our work is how to automatically obtain target Web pages from which data will be extracted. Although this may be seen as a problem that is orthogonal to what we have discussed here, we have also designed and implemented a tool for assisting the user in the tasks of generating agents for collecting Web pages containing data of interest, possibly produced as results of form submission (i.e., dynamic pages). This tool, called ASByE (Agent Specification By Example) is described in [Golgher *et al.*, 2000a]. An interesting direction for future work is to further explore the integration between wrappers generated by DEByE and agents generated by ASByE.

## Resilience and Adaptiveness

As the structural and presentation features of Web pages are prone to frequent changes, a most needed property of wrappers is *resilience*, i.e., the ability of continuing to work properly in the occurrence of changes in the pages to which they are targeted. It is also desirable that a wrapper built for pages of a specific Web source on a given application domain could work properly with pages from another source in the same application domain. Such a property is called *adaptiveness*.

In [Golgher *et al.*, 2001], a method is described that can automatically select, from a sample page of a data source, strings that can serve as examples of attribute values for generating avp-patterns. The general idea is, given a repository $R$ containing data extracted by a pre-existing wrapper $W$, to use the values of attributes in $R$ to match the examples strings in the sample page.

By coupling this method with the Hot Cycles algorithm, we plan to work on an new and more automated data extraction method that can be used to generate wrappers that are resilient and adaptive.

# Bibliography

[Abiteboul *et al.*, 1995] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, Reading, Massachusetts, 1995.

[Abiteboul *et al.*, 1997] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[Abiteboul *et al.*, 1999] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, San Francisco, California, USA, 1999.

[Abiteboul, 1997] Serge Abiteboul. Querying Semi-Structured Data. In *Proceedings of 6th International Conference on Database Teory*, pages 1–18, Delphi, Greece, 1997.

[Adelberg, 1998] Brad Adelberg. NoDoSE – A tool for semi-automatically extracting structured and semistructured data from text documents. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 283–294, Seattle, Washington, USA, 1998.

[Arantes *et al.*, 2001a] Alisson R. Arantes, Alberto H. F. Laender, Paulo B. Golgher, and Altigran S. da Silva. Managing web data through views. In *Proceedings of the 2st International Conference on Eletronic Commerce and Web Technologies*, pages 154–165, Munich, Germany, 2001.

[Arantes *et al.*, 2001b] Alisson R. Arantes, Alberto H. F. Laender, Paulo B. Golgher, and Altigran S. da Silva. An environment for building and maintaining web views. In *Procedings of the First Workshop on Information Integration on the Web*, pages 172–178, Rio de Janeiro, Brasil, 2001.

[Arocena and Mendelzon, 1998] Gustavo O. Arocena and Alberto O. Mendelzon. WebOQL: Restructuring documents, databases, and webs. In *Proceedings of the 14th International Conference on Data Engineering*, pages 24–33, Orlando, Forida, USA, 1998.

[Baeza-Yates and Ribeiro-Neto, 1999] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, Harlow, England, 1999.

[Baumgartner *et al.*, 2001] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web information extraction with Lixto. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 119–128, Rome, Italy, 2001.

[Bray *et al.*, 2002] Tim Bray, Jean Paoli, and Michael Sperberg-McQueen. Extensible markup language (XML) 1.0. http://www.w3.org/TR/REC-xml, 2002.

[Buneman *et al.*, 1996] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A Query Language and Optimization Techniques for Unstructured Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[Buneman *et al.*, 1999] P. Buneman, A. Deutsch, and W. Tan. A Deterministic Model for Semistructured Data. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, Jerusalem, Israel, 1999.

[Buneman, 1997] P. Buneman. Semistructured Data. In *Proceedings of the Sixteenth ACM ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117–121, Tucson, Arizona, USA, 1997.

[Califf and Mooney, 1999] Mary Elaine Califf and Raymond J. Mooney. Relational Learning of Pattern-Match Rules for Information Extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI'99)*, pages 328–334, Orlando, Florida, USA, 1999.

[Callan, 1994] J. P. Callan. Passage-Level Evidence in Document Retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 302–309, Dublin, Ireland, 1994.

[Crescenzi and Mecca, 1998] Valter Crescenzi and Giansalvatore Mecca. Grammars have exceptions. *Information Systems*, 23(8):539–565, 1998.

[Crescenzi *et al.*, 2001] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Towards automatic data extraction from large Web sites. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 109–118, Rome, Italy, 2001.

[da Silva *et al.*, 2002] Altigran S. da Silva, Irna M. R. Evangelista-Filha, Alberto H. F. Laender, and David W. Embley. Representing and querying semistructured web data using nested tables with structural variants. In *Proceedings of the 21st International Conference on Conceptual Modeling ER 2002*, Tampere, Finland, pages 135–151, 2002.

[Embley *et al.*, 1999a] David W. Embley, Douglas M. Campbell, Y. S. Jiang, Stephen W. Liddle, D. W. Lonsdale, Yiu-Kai Ng, Dallan Quass, and Randy D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data & Knowledge Engineering*, 31(3):227–251, 1999.

[Embley *et al.*, 1999b] David W. Embley, Y. S. Jiang, and Yiu-Kai Ng. Record-boundary discovery in Web documents. In *Proceedings ACM SIGMOD International Conference of Management of Data*, pages 467–478, Philadelphia, Philadelphia, USA, 1999.

[Evangelista-Filha *et al.*, 2001] Irna M. R. Evangelista-Filha, Alberto H. F. Laender, and Altigran S. Silva. Querying Semistructured Data By Example: The QSByE Interface. In *Proceedings of the International Workshop on Information Integration on the Web*, pages 156–163, Rio de Janeiro, Brazil, 2001.

[Florescu *et al.*, 1998] Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. Database techniques for the World-Wide Web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.

[Freitag, 2000] Dayne Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(2/3):169–202, 2000.

[Golgher *et al.*, 2000a] Paulo B. Golgher, Alberto H. F. Laender, Altigran S. da Silva, and Berthier Ribeiro-Neto. An example-based environment for wrapper generation. In *Proceedings of the 2nd International Workshop on The World Wide Web and Conceptual Modeling*, pages 152–164, Salt Lake City, Utah, USA, 2000. Held in conjunction with ER 2000.

[Golgher *et al.*, 2000b] Paulo B. Golgher, Alberto H. F. Laender, Altigran S. da Silva, and Berthier A. Ribeiro-Neto. ASByE: uma ferramenta baseada em exemplos para especificação de agentes para coleta de documentos web. In *Anais do XV Simpósio Brasileiro de Banco de Dados*, pages 217–231, João Pessoa, Brasil, 2000.

[Golgher *et al.*, 2001] Paulo B. Golgher, Altigran S. da Silva, Alberto H. F. Laender, and Berthier A. Ribeiro-Neto. Bootstrapping for Example-Based Data Extraction. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management*, pages 371–378, Atlanta, Georgia, USA, 2001.

[Hammer *et al.*, 1997] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. Breunig, and V. Vassalos. Template-based wrappers in the TSIMMIS system. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 532–535, Tucson, Arizona, USA, 1997.

[Hégaret and Wood, 2002] Philippe Le Hégaret and Lauren Wood. The Document Object Model (DOM). http://www.w3.org/DOM, 2002.

[Hopcroft *et al.*, 2001] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, Reading, MA, 2nd. edition, 2001.

[Hsu and Dung, 1998] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the Web. *Information Systems*, 23(8):521–538, 1998.

[Huck *et al.*, 1998] Gerald Huck, Peter Fankhauser, Karl Aberer, and Erich J. Neuhold. Jedi: Extracting and synthesizing information from the Web. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems*, pages 32–43, New York City, New York, USA, 1998.

[Kaszkiel and Zobel, 1997] M. Kaszkiel and J. Zobel. Passage Retrieval Revisited. In *Proceedings of the 20th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 178–185, Philadelphia, USA, 1997.

[Kushmerick, 2000] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[Laender *et al.*, 2000] Alberto H. F. Laender, Berthier Ribeiro-Neto, Altigran S. da Silva, and Elaine Spinola Silva. Representing Web Data as Complex Objects. In K. Bauknecht, S. K. Mandria, and G. Pernul, editors, *Electronic Commerce and Web Technologies*, pages 216–228. Springer, Berlin, 2000.

[Laender *et al.*, 2002a] Alberto H. F. Laender, Altigran S. da Silva, Paulo B. Golgher, Berthier Ribeiro-Neto, Irna M. R. Evangelista-Filha, and Karine V. Magalh aes. The DEByE environment for web data management. *IEEE Internet Computing*, 6(4):60–69, 2002.

[Laender *et al.*, 2002b] Alberto H. F. Laender, Berthier Ribeiro-Neto, and Altigran S. da Silva. DEByE – Data Extraction by Example. *Data and Knowledge Engineering*, 40(2):121–154, 2002.

[Laender *et al.*, 2002c] Alberto H. F. Laender, Berthier Ribeiro-Neto, Altigran S. da Silva, and Juliana Santiago Teixeira. A Brief Survey of Web Data Extraction Tools. *SIGMOD Record*, 31(2):84–93, 2002.

[Libkin, 1991] L. Libkin. A Relational Algebra for Complex Objects Based on Partial Information. In *Proceedings of the Third Symposium on Mathematical Fundamentals of Database and Knowledge Systems*, pages 29–43, Rostock, Germany, 1991.

[Liu *et al.*, 2000] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for Web information sources. In *Proceedings of the 16th International Conference on Data Engineering*, pages 611–621, San Diego, California, USA, 2000.

[Ludäscher *et al.*, 1998] Bertram Ludäscher, Rainer Himmeröder, Georg Lausen, Wolfgang May, and Christian Schlepphorst. Managing semistructured data with FLORID: A deductive object-oriented perspective. *Information Systems*, 23(8):589–613, 1998.

[Magalhães *et al.*, 2001a] Karine V. Magalhães, Alberto H. F. Laender, and Altigran S. da Silva. Storing semistructured data in relational databases. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval*, pages 143–152, Laguna de San Rafael, Chile, 2001.

[Magalhães *et al.*, 2001b] Karine V. Magalhães, Alberto H. F. Laender, and Altigran S. da Silva. Uma abordagem para armazenamento de dados semi-estruturados em bancos de dados relacionais. In *Anais do XVI Simpósio Brasileiro de Banco de Dados*, pages 140–154, Rio de Janeiro, Brasil, 2001.

[Makinouchi, 1977] Akifumi Makinouchi. A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model. In *Proceedings of the 3rd International Conference on Very Large Data Bases*, pages 447–453, Tokyo, Japan, 1977.

[Mecca *et al.*, 1998] Giansalvatore Mecca, Paolo Atzeni, Alessandro Masci, Paolo Merialdo, and Giuseppe Sindoni. The Araneus Web-Base Management System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 544–546, Seattle, WA, 1998.

[Miller *et al.*, 2002] Eric Miller, Ralph Swick, and Dan Brickley. Resource Description Framework (RDF). http://www.w3.org/RDF/, 2002.

[Miller, 2002] Eric Miller. Semantic Web Activity Statement. http://www.w3.org/2001/sw/Activity, 2002.

[Muslea *et al.*, 1999] Ion Muslea, Steven Minton, and Craig Knoblock. An Hierarchical Approach to Wrapper Induction. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 190–197, Seattle, WA, 1999.

[Muslea *et al.*, 2001] Ion Muslea, Steven Minton, and Craig Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[Muslea, 1999] Ion Muslea. RISE: Repository of online information sources used in information extraction tasks. http://www.isi.edu/ muslea/RISE/, 1999.

[Papakonstantinou *et al.*, 1995] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object Exchange Across Heterogenous Information Sources. In *Proceedings of 11th International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, 1995.

[Ribeiro-Neto *et al.*, 1999a] Berthier Ribeiro-Neto, Alberto H. F. Laender, and Altigran S. da Silva. Top-down Extraction of Semi-Structured Data. In *Proceedings of the 6th International Symposium on String Processing and Information Retrieval*, pages 184–191, Cancun, Mexico, 1999.

[Ribeiro-Neto *et al.*, 1999b] Berthier Ribeiro-Neto, Alberto H. F. Laender, and Altigran S. da Silva. Extracting semi-structured data through examples. In *Proceedings of the 1999 ACM CIKM International Conference on Information and Knowledge Management*, pages 94–101, Kansas City, Missouri, USA, 1999.

[Sahuguet and Azavant, 2001] Arnaud Sahuguet and Fabien Azavant. Building intelligent Web applications using lightweight wrappers. *Data and Knowledge Engineering*, 36(3):283–316, 2001.

[Silva, 1999] E. S. Silva. Extraction of Semi-Structured Data Based on Examples. Master's thesis, Departament of Computer Science, Federal University of Minas Gerais, 1999. In Portuguese.

[Soderland, 1999] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[Sudkamp, 1997] Thomas A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science*. Addison-Wesley, second edition edition, 1997.

[Zloof, 1977] Moshé M. Zloof. Query-by-Example: A Data Base Language. *IBM Systems Journal*, 16(4):324–343, 1977.