Poznań University of Economics
Department of Information Systems

Doctor of Philosophy Dissertation

Semi-Automatic Web Information Extraction

by

Dominik Flejter

Supervisor: Prof. dr hab. Witold Abramowicz

Poznań, 2011

Contents

Contents **iii**

Acknowledgments **ix**

Introduction **xi**

1 Role of Information in E-commerce **1**

1.1 Data, Information, Knowledge and Wisdom 2

1.1.1 Static View of Data, Information, Knowledge and Wisdom 2

1.1.2 Dynamic View of Data, Information and Knowledge 3

1.1.3 Level of Information Structure 4

1.2 Data-Intensive Web Sites 6

1.2.1 Basic Data-Intensive Web Sites 6

1.2.2 Deep Web 7

1.2.3 Personalized Web Sites 9

1.2.4 Adaptive Web Sites 9

1.2.5 Collaborative Filtering Web Sites 10

1.2.6 Web Applications 10

1.2.7 GUI-centric Web Sites 11

1.2.8 E-Commerce Sites 12

1.2.9 Social Networking Web Sites 12

1.2.10 Generalized Data-Intensive Web Sites 12

1.2.11 Basic and Complex Data-Intensive Web Sites 13

1.3 Web Information in Electronic Commerce 14

1.3.1 Electronic Commerce Activities 14

1.3.2 The Role of Information in E-commerce 17

1.3.3 Scenarios of Using Web Information in E-Commerce 19

1.4 Summary 22

2 Web Information Extraction **23**

2.1 Bringing Structure to the Web 23

2.1.1 Web Content Mining . . . 25
2.1.2 Web Structure Mining . . . 26
2.1.3 Web Usage Mining . . . 27
2.2 Information Extraction . . . 28
2.2.1 Information Extraction and Web Information Extraction . . . 28
2.2.2 Wrappers and Extraction Rules . . . 29
2.3 Challenges of IE from Data-Intensive Web Sites . . . 30
2.3.1 Literature Review . . . 31
2.3.2 User Interaction Model . . . 33
2.3.3 Real-Life Web Sites Analysis . . . 36
2.3.4 WIE Application Scenarios . . . 39
2.3.5 Identified Challenges . . . 41
2.4 Summary . . . 42

3 Comparison of Existing WIE Systems 43
3.1 Comparison Scheme for WIE Systems . . . 43
3.1.1 Existing Classification Schemes . . . 43
3.1.2 Classification Used in this Thesis . . . 46
3.2 Comparison of Existing Solutions . . . 47
3.2.1 Feature-based Analysis . . . 47
3.2.2 Genetic Analysis . . . 56
3.2.3 Comparison of Performance . . . 57
3.3 Challenges to Address . . . 60
3.3.1 Information Extraction from Web Documents . . . 61
3.3.2 Extracting Information from Complex Web Sites . . . 63
3.3.3 Large-Scale Information Extraction . . . 65
3.4 Summary . . . 65

4 Proposed Web Information Extraction Method 67
4.1 Objectives of Proposed Solution . . . 67
4.1.1 Focus of Our Solution . . . 67
4.1.2 Detailed Objectives . . . 68
4.2 Basic Components of Proposed Model . . . 70
4.2.1 Navigation State and Web Actions . . . 70
4.2.2 Web Action Templates . . . 74
4.2.3 Data Extraction Rules . . . 76
4.2.4 Extraction Rules Composition . . . 78
4.2.5 Extraction Rules Templates . . . 79
4.2.6 Data Records and Data Records Schema . . . 80
4.2.7 Attribute Sets . . . 82
4.3 Data Extraction Graph . . . 84
4.3.1 Data Extraction Nodes, Instances and Atoms . . . 84
4.3.2 Input Mappings . . . 88

4.3.3 Data Extraction Graph . . . 89
4.3.4 Input Configurations . . . 91
4.3.5 Executable Data Extraction Graphs . . . 95
4.3.6 Graph Inputs and Outputs . . . 97
4.3.7 Cyclic Data Extraction Graphs . . . 97
4.4 Data Extraction Graph Execution . . . 99
4.4.1 Intuitive Overview . . . 99
4.4.2 Graph Analysis . . . 100
4.4.3 Graph Execution . . . 104
4.4.4 Basic Subroutines . . . 106
4.4.5 Queue Management . . . 107
4.4.6 Provenance-Aware Join . . . 108
4.4.7 Data Storing . . . 112
4.5 Navigation State Management . . . 113
4.5.1 Basic Elements of Navigation State Management . . . 114
4.5.2 State-Aware Graph Execution Challenges . . . 116
4.5.3 Modeling State-Dependence and State Modifications . . . 117
4.5.4 State-Aware Graph Execution Algorithms . . . 118
4.5.5 State-Aware Queue Management . . . 122
4.5.6 State Flow Correctness . . . 123
4.5.7 Current Query Model . . . 126
4.6 Query Planning . . . 127
4.6.1 Basic Query Planning Tasks . . . 128
4.6.2 Optimizing Data Filtering . . . 130
4.6.3 Query Planning Algorithm . . . 136
4.7 Summary . . . 142

5 Implementation of Proposed Method 143
5.1 System Architecture . . . 143
5.1.1 Data Model . . . 145
5.1.2 Data Extraction Graph Execution Component . . . 146
5.1.3 Web Actions Execution Environment . . . 148
5.1.4 User Interface . . . 150
5.2 Implemented Nodes . . . 152
5.2.1 Web Actions Templates . . . 153
5.2.2 Extraction Rules . . . 155
5.2.3 Request Capture Template Node . . . 156
5.2.4 Other Nodes . . . 156
5.3 Implemented Algorithms . . . 156
5.3.1 Queue Management Algorithms . . . 156
5.3.2 State Management Approaches . . . 157
5.3.3 Nodes Merger Algorithms . . . 158
5.4 Summary . . . 158

6 Discussion and Evaluation of Proposed Approach 159
6.1 Comparison of Our Approach with Previous Systems . . . 159
6.1.1 Analysis Based on Comparison Scheme . . . 160
6.1.2 Similarities and Differences Between Used Approaches . . . 162
6.2 Evaluation Based on Test Web Sites . . . 165
6.2.1 Created Web Sources . . . 165
6.2.2 Wrapping Test Web Sites with Our Solution . . . 166
6.3 Generality of Proposed Solution . . . 167
6.3.1 Research Objectives . . . 167
6.3.2 Addressed Challenges . . . 178
6.3.3 Applicability in Different Types of Data-Intensive Web Sites . . . 182
6.3.4 Applicability in Different Business Usage Scenarios . . . 185
6.3.5 Applicability in Different Technical Usage Scenarios . . . 186
6.4 Performance of Proposed Data Extraction Algorithm . . . 187
6.4.1 Experimental Setup . . . 188
6.4.2 Results . . . 190
6.5 Future Research Directions . . . 192
6.6 Summary . . . 194

Conclusions 195

Bibliography 199

A User — Web Site Interaction 221

B Analyzed Web Sites 229

C List of Information Extraction Challenges 233

D Presentation of Existing WIE Systems 243
D.1 TSIMMIS . . . 243
D.2 W3QS . . . 244
D.3 Information Manifold (IM) . . . 244
D.4 WebLog . . . 245
D.5 WebSQL . . . 245
D.6 Ashish and Knoblock Work (AK) . . . 246
D.7 Webfoot . . . 246
D.8 ShopBot/Jango . . . 246
D.9 WIEN . . . 247
D.10 Araneus . . . 248
D.11 WebOQL . . . 249
D.12 FLORID . . . 249
D.13 Jedi . . . 250
D.14 NoDoSE . . . 250

D.15 Stalker . . . 250
D.16 SoftMealy . . . 251
D.17 Junglee VDB . . . 251
D.18 WHIRL . . . 252
D.19 W4F . . . 252
D.20 DEByE . . . 253
D.21 XWrap . . . 254
D.22 WebVCR . . . 255
D.23 Lixto . . . 255
D.24 Hidden Web Exposer (HiWE) . . . 256
D.25 Omini . . . 257
D.26 IEPAD . . . 257
D.27 EXALG . . . 258
D.28 WL2 . . . 259
D.29 RoadRunner . . . 259
D.30 Denodo Integration Platform . . . 260
D.31 DeLa . . . 261
D.32 Cameleon . . . 261
D.33 VIPS . . . 262
D.34 DEPTA . . . 262
D.35 IDE . . . 263
D.36 ViNTs . . . 263
D.37 Thresher . . . 264
D.38 DWDI . . . 264
D.39 SSF . . . 265
D.40 Turbo . . . 265
D.41 Senellart et al. Approach . . . 266
D.42 DEQUE / I-Crawler . . . 267
D.43 Other Systems . . . 268

E Combining CsExWB and FiddlerCore Events 271

F List of Developed Test Web Sites 275

List of Used Symbols 279

List of Figures 281

List of Tables 282

List of Definitions 284

List of Examples 286

List of Algorithms **288**

Acknowledgments

This thesis was made possible by the support and assistance of a number of people, to whom I would like to express my sincere gratitude. First of all, I would like to thank my supervisor, Prof. Witold Abramowicz, for introducing me to the research world, for his confidence in me, and for the research independence I received while working on this thesis.

I am very grateful to the whole research team of the Department of Information Systems at Poznań University of Economics. My special thanks go to Dr. Tomasz Kaczmarek. Many of the ideas central to this thesis were born in our countless discussions. I would like to thank Dr. Agata Filipowska, Dr. Krzysztof Węcel, Dr. Marek Kowalkiewicz and Karol Wieloch for all the research that we have done together. Working on different projects with you helped me fit my thesis into a "bigger picture" and helped me learn about applying various research techniques. I am also grateful to all researchers and students that took part in Integror project.

My most personal thanks go to my family. I would like to thank my parents Elżbieta and Marian for underlining the importance of education and continuous self-development. If not for you, I would never have started my PhD research. I am endlessly grateful to my wonderful wife Monika for fully accepting and supporting my work on this dissertation, for keeping me focused on finishing the work, and for being the first reviewer of this dissertation. My love, if not for you, I would not be able to finish my work. I would also like to thank my sister Oleńka for all the positive emotions she expressed towards my research work.

There are a few other people that more or less directly inspired some aspects of my work. I would like to thank Roman Hryniewiecki and Damian Ciesielczyk for multiple discussions on the business role of information and information extraction. I am grateful to the participants and PC Members of the workshops on Advances in Accessing Deep Web [111, 8], Exploiting Structured Information on the Web [133] and Mashups, Enterprise Mashups and Lightweight Composition on the Web [119, 116, 117] I co-organized for multiple novel ideas concerning advanced Web information extraction challenges and solutions. My final thanks go to the authors of FiddlerCore, CsExWB, libcurl, QuickGraph and GraphViz. I used these free tools for developing my prototype application. I also wish to thank the LaTeX community for making DTP work on this thesis feasible.

Introduction

Motivation

The World Wide Web is a very diversified information space. While it contains a large amount of textual and multimedia content, a significant part of Web information is semi-structured and contains data of business value.

Electronic commerce is an example of a domain where a multitude of Web sites exposing semi-structured information exist, including manufacturers' Web sites (containing the specification and classification of offered products, sales statistics and catalogue prices), suppliers, competitors and auction Web sites (containing valuable and continuously evolving information on product assortment, prices, availability, user reviews and delivery options), user opinion and professional product assessment Web sites (containing user ratings and different kinds of benchmark measurements), as well as search engines and different types of social Web sites (providing information on product and brand popularity).

Acquiring all categories of data cited above is a critical task for most of market participants in many branches of the contemporary economy. At the same time, due to a high number of diverse and quickly evolving data sources on the Web, it is a complex research problem. As a result, the task of accessing semi-structured information available on the Web, called Web information extraction, has been an active field of both research and business activities.

The previous work in the area of *data extraction from data-intensive Web sites* covered a few specific problems, including data extraction from static documents, learning extraction rules based on training data, user interaction or similarity of multiple documents, and acquisition of data from Deep Web sources, i.e. Web databases hidden behind query forms.

Most of these problems have been already studied in depth, and well-performing methods exist. However, the WWW continued to evolve and brought in new information extraction challenges. Over time, many of Web sites have started to use composite Web documents (using (i)frames and JavaScript to combine multiple HTML document), complex server interaction paradigms (such as AJAX), advanced user interface elements (based on a wide use of JavaScript and Flash components, as well as on the rich event model) and different types of stateful design patterns. The problem of extracting information from Web sites using these technologies and interaction models (further referred to as *complex data-intensive Web sites*), which is central to this thesis, has not been addressed by any previous solution.

Scope of Presented Research

The **research area** of this PhD thesis concerns Web information extraction, i.e. the extraction of structured data objects from semi-structured Web sources.

The **research problem** studied in this thesis concerns Web information extraction from complex data-intensive Web sites, i.e. Web sites that are stateful, use advanced server-interaction patterns such as asynchronous communication, or rely on client-side dynamic technologies, including frames and JavaScript, in their user interfaces. While this problem shares many traits with extensively studied problems of performing data extraction from static documents and Deep Web sources, it is significantly more challenging. Only a few of the challenges specific for data extraction from complex data-intensive Web sites have been previously partially addressed, and many of them have not been even explicitly defined in the previous work. Therefore, Web information extraction from complex data-intensive Web sites should be treated as a new research problem.

Our **research objective** is to propose a data extraction model and algorithms capable of performing data extraction from previously handled basic data-intensive Web sites and Deep Web sources, as well as from complex data-intensive Web sites that were not handled by existing approaches. The proposed model and algorithms should be characterized by low overhead, i.e. by a low proportion of irrelevant files downloaded from HTTP servers during information extraction.

The proposed solution consists of a few elements, and in our opinion well addresses the stated objective. Therefore, the thesis that we will defend in this dissertation is that

> using query planning and execution algorithms based on an extensible, state-aware data extraction graph model and on a rich representation of Web documents, enables a low-overhead information extraction from a significantly larger set of complex data-intensive Web sites than in the case of previous solutions.

Methodology

In this thesis we follow the information system research methodology defined by Hevner and colleagues in [152]. The authors distinguish between two key paradigms of information systems research: the behavioral-science paradigm and the design-science paradigm. The former focuses on developing and verifying theories of organizational behavior, and its objective is truth. The latter "seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts", and its objective is utility. While one cannot exist without the other, the focus of Hevner et al.'s methodology, as well as of our research, is on designing information systems.

The applied methodology considers the design of information systems as a complex activity that is framed by two key external components: the environment and the knowledge base. Analysis of the environment, consisting of people, organizations and technologies, is necessary to assure that research is relevant, i.e. that it corresponds to business needs and that it can be applied in real-life business scenarios. On the other hand, by analyzing knowledge base, which consists of existing approaches and research tools (foundations and methodologies), one assures research rigor and "the clear identification of a contribution to the archival knowledge base of foundations and methodologies". Internally, the information system research consists of two interconnected activities of designing theories or artifacts and evaluating their utility. The relation between these external and internal information systems research components is schematically depicted in Figure 1.

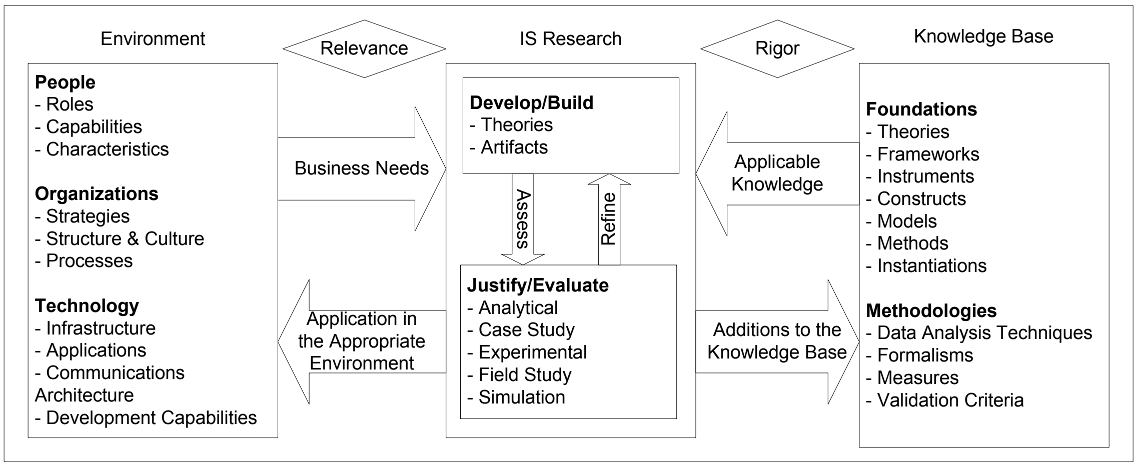


Figure 1: Applied Methodology (source: [152])

To make following the methodology more simple, Hevner and colleagues provide seven guidelines that should be followed by design-science research. The first of them specifies that the output of information systems research should consist of artifacts, such as constructs (basic terms and objects used in communicating problems), models (abstract representations of problem and solution spaces), methods (processes or algorithms) and instantiations (exemplary implementations of models and methods, demonstrating feasibility). The second guideline states that research results should consist in developing "technology-based solutions to important and relevant business problems". Moreover, it needs to address "important unsolved problems in unique or innovative ways or solved problems in more effective or efficient ways". The third guideline is that research results should be rigorously evaluated in terms of utility, quality, and efficacy of designed artifacts, using well-recognized observational, analytical, experimental, testing or descriptive evaluation methods. The fourth guideline concerns the provision of "clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies". The fifth guideline states that rigorous data collection and analysis methods (in the case of the behavioral-science paradigm), and formalization (in the case of the design-science paradigm), re-using the existing knowledge base, should be applied. The sixth guideline is that research should be an iterative process of improving of artifacts and methodologies – both those designed by the researcher and those belonging to the existing knowledge base. Finally, the seventh guideline is that research results should be communicated both to the practitioners and to the researchers.

Structure of this Thesis

This thesis consists of six chapters and six appendices. Chapters 1–3 cover the non-original part of our research, while chapters 4–6 correspond with our original contribution to the field of Web information extraction.

The structure of this thesis directly reflects the adopted methodology. In the first two chapters we study the environment, introduce key constructs and define the general research problem. In Chapter 1 we analyze the organizational environment. The study concerns different types of Web information sources (1.2) and their usability in different types of electronic commerce business entities and usage scenarios (1.3). It combines a literature review and an analysis of specific business scenarios, assuring relevance of the designed solution to the business needs. At the same time, it illustrates the motivation behind our research.

Chapter 2 is devoted to the technological environment, and concerns the challenges of acquiring information from Web sources described in Chapter 1. It provides a top-down presentation of the research area (2.1), defines the general research problem of Web information extraction from data-intensive Web sites (2.2) and discusses its challenges (2.3). It combines the techniques of literature review (2.3.1), case-based research (a review of real-life Web sites (2.3.3) and technical usage scenarios (2.3.4)) and a modeling-based approach (2.3.2).

In Chapter 3 we study the knowledge base in the research area of this thesis. Based on a literature review, we propose a multi-criteria analytical scheme for comparing different information extraction solutions (3.1), and apply it to 39 state-of-the-art data extraction systems (3.2). We also provide a critical analysis of the gap between environment (i.e. business needs and technological challenges) and the existing knowledge base (3.3).

In chapters 4 and 5 we describe our own contribution to the area of information extraction from data-intensive Web sites. We start by defining our research problem and detailed research objectives (4.1), based on the identified gap between environment and knowledge base. Then, we propose our own approach to information extraction from complex data-intensive Web sites, consisting of models and methods (presented in Chapter 4) as well as their instantiations (described in Chapter 5). Description of the proposed models, including basic static components (4.2), data extraction graph (4.3) and its extensions dealing with stateful Web sites (4.5), consists of intuition, examples and formal definitions. All introduced methods, covering data extraction graph execution (4.4), state management in stateful Web sites (4.5) and query planning (4.6), have a form of formalized algorithms. All models and methods are instantiated as software components, according to a flexible system architecture (5.1).

Chapter 6 provides a discussion and evaluation of the proposed solution. The evaluation focuses mostly on the generality of our approach with respect to attained research objectives, addressed challenges, capability of dealing with different types of data-intensive Web sites, and applicability in different business and technical usage scenarios (6.3). However, we also analyze the performance of our approach in terms of limiting data extraction overhead (6.4). Thus, our evaluation combines the evaluation techniques of simulation, scenarios analysis and informed argument. To complete the thesis, we briefly discuss some of challenges that have not been addressed by our current solution, and introduce some ideas for handling them in our future work (6.5).

Chapter 1

Role of Information in E-commerce

The Web is undoubtedly the biggest repository of information in the history of humanity. According to recent estimates, the contemporary World Wide Web contains up to 20 billion[1] indexed documents, available at over one trillion unique URLs[2], exposed by over 131 million unique domains[3]. At the same time, so-called Deep Web (consisting of on-line databases available through simple query interfaces) is believed to be even 500 times bigger than the indexable Web [48] and to include up to 25 billion Web sources [287].

In parallel, the Web has become a huge marketplace offering a wide variety of goods and services to businesses and customers all around the world. In recent years we observed a constant growth of electronic commerce in both developed and developing countries. According to a Forrester report from 2010[4], U.S. on-line retail sales are expected to attain almost $250 billion in 2014, accounting for 8% of total retail sales. Moreover, it expects that by 2012 over a half of all U.S. retail sales will be "Web-influenced", i.e. start by on-line research. The same study forecasts that on-line retail spending will continue grow in Western Europe by over 10% a year in the next few years. At the same time, Polish on-line retail sales accounted for 3.3 billion euros in 2010 and are expected to almost double within five year period, attaining the value of 5.9 billion euros in 2015, according to a study by ResearchFarm[5].

Electronic commerce uses to a large extent automated processes, resulting in the large availability of information in a digital form. On the other hand, a high level of automation means that electronic commerce is specially dependent on information. The efficient acquisition and processing of information about business environment is one of the major sources of competitive advantage; conversely, the inability to acquire and understand crucial information can lead to spectacular failures both in the operational and strategic level.

[1]Source: Tilburg University's http://www.worldwidewebsize.com/, accessed on 10.07.2011.

[2]Figure announced by Google in 2008, including multitude of auto-generated content, see: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html, accessed on 15.01.2011.

[3]Source: http://www.domaintools.com/internet-statistics/, accessed on 10.07.2011.

[4]See: http://tinyurl.com/ecomUS2014, accessed 19.09.2010

[5]See: http://www.rp.pl/artykul/2,668643_Internetowy_boom_w_Polsce.html, accessed on 02.06.2011.

In this chapter we analyze the organizational environment of our research problem [152] in the area of electronic commerce. Before discussing the scope of on-line information that is useful in e-commerce activities (Section 1.2) and how can it be used (Section 1.3.3), we introduce basic terms, i.e. the distinction between data, information, knowledge and wisdom, and between structured, semi-structured and unstructured documents and data (Section 1.1).

1.1 Data, Information, Knowledge and Wisdom

Data, information, knowledge and wisdom are terms with multiple definitions and different usage modes derived from a number of fields, including mathematics, philosophy, sociology, physics and computer science; even if we consider them solely in the context of information technology, various meanings are used by different authors.

1.1.1 Static View of Data, Information, Knowledge and Wisdom

The most classic IT representation of the relation between these terms is the "pyramid of wisdom" (see Figure 1.1),[6] which captures the interdependencies between these terms in a form of four[7] distinct layers laying one atop of one another.

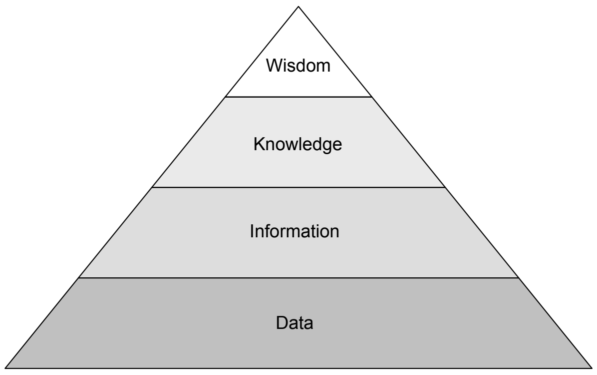


Figure 1.1: Pyramid of Wisdom

The higher layers of the hierarchy are based on and extend the lower layers. It is typically assumed that the relative volume of the higher levels is smaller than in the case of the lower levels[8], i.e. that the higher levels are more "condensed" or "abstracted". In general, the higher levels are considered better suited for usage in an unchanged form; in other words, less effort is required to use higher-level resources in decision making as compared with lower levels.

For example, we can treat records stored in a data warehouse as data, specific textual reports using aggregated data warehouse records as information, generalizations over these data as knowledge and statistical or intuition-based forecasting models as wisdom. If the reports are organized around a product's dimension, they are almost directly applicable to

[6]This hierarchy is typically attributed to Ackoff [10], however a similar hierarchical view was previously used by other researchers, and can be even tracked down to poetry [245].

[7]The number of layers differs. Some researchers add the level of capability, expertise, insight, judgment or decision, and some skip the level of wisdom [246].

[8]That is why the triangular shape is used rather than rectangular levels

sales decisions. However, if the reports do not consider the geographical dimension of sales, decisions regarding the regional differentiation of sales can be only taken based on data analysis. Similarly, the intuition acquired while taking overall sales decisions cannot be easily applied to predicting sales in specific regions.

In this thesis, we focus on the data and information layers of the pyramid of wisdom. **Data** can be defined as basic facts “communicated using numbers, texts, figures or sounds” resulting from “observation of phenomena, things and people” [7]. **Information** can be defined as “a collection of data concerning specific objects, organized and presented in a systematic way, enabling their understanding” [7], as data interpreted by people that enable or cause decision taking [139], or as data assigned context, which enables interpretation [129].

In this thesis we take an approach similar to [7], defining data as all facts that are the individual property values of specific objects, and information as schema-based organization imposed on individual data. Notions of knowledge and wisdom are outside of our interest in this thesis. For a few definitions of knowledge we refer the interested reader to [44].

1.1.2 Dynamic View of Data, Information and Knowledge

So far, we have presented the quantitative and qualitative relation between data, information, knowledge and wisdom, without analyzing corresponding dynamic relations i.e. the transformations of one resource into another.

Wisdom and knowledge are by definition well-suited for taking decisions and solving unknown and known problems, respectively. Information needs to be organized, analyzed and understood before it becomes useful in decision making. Finally, a number of processes, including the acquisition, selection, cleansing, transformations (including aggregation) and assignment of precise meaning are required to transform data into information, which can be in turn transformed into knowledge. A concise formulation of these transformations can be found in [45]. In this view, data becomes information thanks to understanding relations, information is transformed into knowledge by understanding patterns; and finally, thanks to understanding principles knowledge becomes wisdom (see: Fig 1.2.a).

It is assumed that the transformation from information to knowledge is a deterministic process of collecting information needed to understanding specific fragment of reality. By contrast, passing from knowledge to wisdom is non-deterministic and involves the interpolation of previously acquired knowledge, enabling the solving of unknown problems [10].

While Fig 1.2 captures well typical transformations, it ignores the continuous character of both data acquisition and the transformations themselves. However, as many data sources act as streams of rapidly changing data or information, continuous transformations to other forms are required. For example, whenever new prices of suppliers and competitors are acquired from the market, new information on trends needs to be discovered, and decisions regarding positioning, assortment, pricing and purchases need to be taken. Similarly, new documents (e.g. news or reports) regarding the legal, economical or competitive environment can be used both to continuously re-create company policies (reflecting the knowledge or wisdom of people and organizations) and to acquire individual data that can be used in other analyses (i.e. in generating more information).

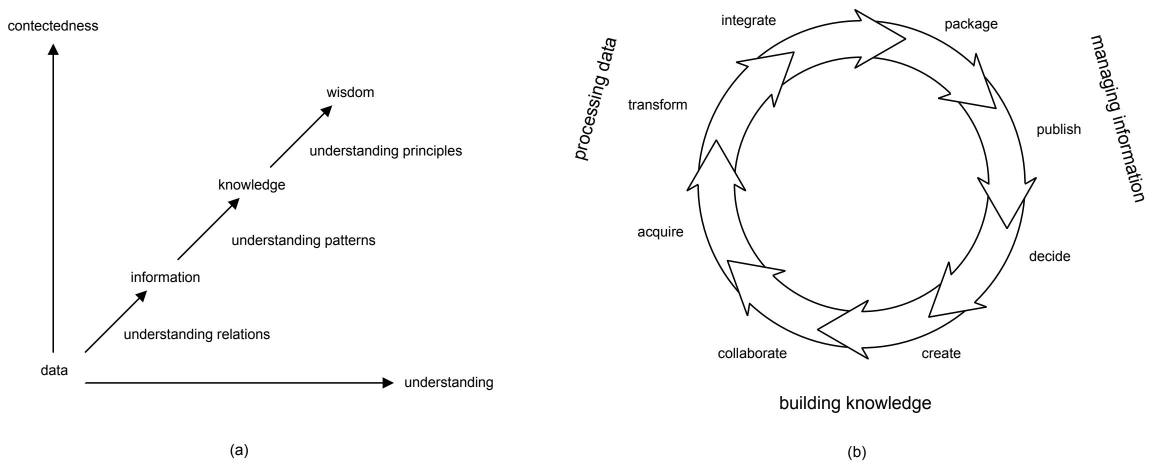


Figure 1.2: Transformation of Data Into Wisdom: a) linear view (source: [45]), b) cyclic view (Source: [139])

An example of a such continuous approach is proposed in [139] in the form of a cycle connecting data, information and knowledge (wisdom is not present in this approach). The cycle depicted in Fig 1.2.b), is composed of three distinct sub-processes: processing data, managing information and building knowledge.

The starting point for this cycle is data acquisition, followed by data transformation and integration. This tripartite cycle relates to processing data stage. Next stage – managing information – starts with the packaging of data, which required to obtain information. Next, information is published and used in decision taking, becoming knowledge. It is here that the final stage of the cycle, called building knowledge, starts. Using information in decision making enables the creation of knowledge that may be further shared by collaboration. As a result of acquiring new knowledge, new requirements for data acquisition are defined. This mechanism motivates closing of the cycle.

1.1.3 Level of Information Structure

Data or information usefulness is determined partially by how it is structured, i.e. how precise the separation of individual facets of information is. Following this dimension, both data and their presentation (i.e. documents that contain them) are typically divided into structured, semi-structured and unstructured. While all three terms are widely used, there are significant differences regarding their understanding.

Database researchers draw the distinction between structured, semi-structured and unstructured data, related to how well data records are restricted by a schema definition. [9] According to [100], data contained in documents with no schema or type definition (e.g. free text or HTML documents) are unstructured, data contained in documents that mix data

[9]We use here the word "data" in the way it is used in the database community; however, referring it to the "pyramid of wisdom" we would rather use the term "information", as we mean multiple interconnected relational tables or classes.

records with schema information (e.g. XML) are semi-structured, and data in repositories fully separating records from schema (e.g. in databases) are structured.

In this thesis, we define level of information structure as the extent to which it can be represented by a relational schema containing a few interconnected tables. The information is **structured** if its relational representation is lossless. It is **semi-structured** when its core can be represented in a relational way, but some characteristics of described entities are hard to capture in a structured way (e.g. they cannot be expressed by simple types). Finally, it is **unstructured** if it is not possible to represent its core as relational records.

For example, typical economic data are structured as they can be represented as a spreadsheet or a number of relational tables; information on products is typically semi-structured as it mixes product attributes (such as dimensions or price) with hard-to-capture characteristics (such as design, usability or personal experiences); finally, for example, emotional states are unstructured information as they are hard to express as data.

It is important not only how well structured the data are, but also how much structure is contained in their presentation format, i.e. what the level of structure of document containing data is. One classification of documents is based on the structure hints they contain. In this view, free text documents are unstructured, ungrammatical/telegraphic documents with rather stable data presentation patterns and vocabulary are semi-structured (examples include medical records, equipment maintenance logs or classifieds), while documents with explicit extra structure and formatting hits (such as HTML tags) are structured [254, 97]. Classifications of documents into having rigorous, semi-rigorous, semi-relaxed and relaxed structure, as well as into low and high-quality HTML [72], also fit into this direction.

Another classification focuses on the effort needed to structure the content of documents. According to this division, free text documents are unstructured as they require substantial natural language processing, the majority of HTML documents are semi-structured "since the embedded data are often rendered regularly via the use of HTML tags" and XML files are structured since their interpretation is possible based on DTD or XML Schema [66].

In this thesis, we relate the level of structure of Web pages only to the level of their actual representation as structured objects that can be processed by computers. Thus, the page is **structured** when all entities, relation and attributes are explicitly separated and described in a machine-readable language (as in case of databases of RDF files); it is **semi-structured** when it contains some structural hints that help extract majority of entities, relations or attributes of contained information (as in case of most HTML files); and is **unstructured** when the extraction of needed information requires an understanding of contained entities (concepts) and their relations (as in the case of free text or graphical files).

In this setting, any information which is unstructured by nature can be represented only as unstructured Web pages; in the meantime, well-structured information may be represented as a well-structured or mostly unstructured Web page. It is to be noted that semi-structured information and semi-structured pages may be as useful as their structural counterparts, if specific usage scenario is not affected by missing details. Moreover, in the majority of real-life scenarios semi-structured documents may by the "best available" and would typically still be preferred over less structured documents or no documents at all.

1.2 Data-Intensive Web Sites

While a lot of information made available on-line is unstructured (e.g. text or multimedia documents), a significant percent of Web sites aim at publicizing structured or semi-structured information in a structured or semi-structured way. These Web sites, further called *data-intensive Web sites*, are typically focused on a few classes of objects and posses stable schema of records presented on-line. In the following sections we review and compare their most significant categories.

1.2.1 Basic Data-Intensive Web Sites

The way data are presented on-line in data-intensive Web sites is strongly influenced by the characteristics of the dominant language of on-line documents, which is HTML. While HTML is a tag-based language representing documents as trees of nested elements, in its basic form it implements no separation of content and definition of its presentation and provides no constructs for data structure definition.

In an attempt to separate presentation from data, the standard of Cascading Style Sheets (CSS) was proposed [1], enabling the external definition of presentation features (such as margins, colors and fonts) of specific tags. However, due to a number of legacy systems, Web designer conservativeness and some shortcomings of CSS, a large number of Web sites still use a mixture of formatting based on the typical rendering of tags and using cascading styles.

Out of valid HTML tags only few (e.g. `<P>` for paragraph or `<H1>`-`<H6>` tags for headings) are intended to be used as structure markers; all of them are useful rather for encoding the structure of (hyper)textual data than data records [230]. Moreover, as Web pages are mostly prepared for human readers, in the majority of cases specific tags are used because of their display by popular Web browsers rather than because of associated meaning. As a result, even structure-oriented tags are used in a very inconsistent way by different Web sites.

As a result of HTML features and their actual usage, Web sites typically provide data in a semi-structured format. They use few types of tags to provide structured information. One group of tags used for this purpose are tags related to ordered, unordered and definition lists. Another group contains basic text-structuring tags such as paragraphs and headers. Another group consists of tags focused on formatting (such as `<B>`, `<I>` or `<EM>`). Finally, the most important tags used in providing data on-line are related to tables. HTML tables are omnipresent and are used for a variety of purposes. While one of them is providing tabular view of some data, it is surely not the most popular one. Tables in Web sites are used mostly for layout preparation purposes – the majority of them are not directly visible, but still they remain the easiest way to implement layout with multiple distinct elements (such as left menu, top menu, page content, footer, news box etc.)[184, 270].

Typical organizations of data in a Web page include lists (generated with list tags, tables or DIVs), hierarchical lists (with multiple embedded lists, tables or DIVs with specific visual features for different levels of hierarchy) and complex tables (with data organized both along rows and columns, often with complex table headers). While individual attributes are often surrounded by specific tags, in many cases multiple attributes are concatenated within a single tag, making their extraction much harder.

In typical data-intensive Web sites data, are not only organized within HTML documents but are also split into multiple pages connected by a network of hyperlinks. In many cases (for example in all kinds of directory and in the majority of e-commerce Web sites) hyperlinks connecting pages are related to a specific attribute of records; by following links, a user accesses page(s) containing data with this attribute set to a specific value. We assume that in basic data-intensive Web sites all data are available solely by using link-based navigation, and that at a specific time the same URL gives different users access to the same data, independent of their navigation session or login information (if required).

In the case of data-intensive Web sites, both pages' content and some parts of their navigation structure (e.g. categories links in directory-styled sites) are generated from underlying data repositories. While in the majority of cases the content of the Web page to be displayed is generated when the HTTP request is issued, other solutions including partial caching or generating static pages upon database changes. In all these approaches, dynamically generated Web pages follow some regularities; while there may be some differences between the structures of similar pages (e.g. the number of records, attributes present only in some records, blocks of content appearing conditionally), they share a common template. Similarly, the parts of the navigation structure that are generated dynamically follow some patterns related to data attributes. While these structures (template and general navigation structure) are typically hidden, we adopt an assumption that they exist and can be somehow discovered algorithmically.

To access all data in data-intensive Web sites one can use typical crawler software. By contrast, query answering within data-intensive Web sites typically requires the indexing of their whole content.

1.2.2 Deep Web

Deep Web [48] (as opposed to Surface Web) is a huge part of the Web, consisting of data-intensive Web sites advertising their content only via query interfaces (rather than hyperlinks) [148, 113, 114]. As its content remains mostly non indexed by general purpose search engines, it is also referred to as Invisible Web [187, 185] or Hidden Web [232, 36, 18].

HTML forms are the active components of Web pages used to acquire user input. In the case of Deep Web sites, user input is used to generate Web pages containing responses to a specific user queries. Web forms may contain the following types of fields [230]:

- Text fields (`<INPUT TYPE="text">` and `<TEXTAREA>` tags) that allow users to enter any textual content (single or multiple lines, respectively).
- List fields (`<SELECT>` or `<SELECT MUTLIPLE>` tags) that allow users to select one or multiple predefined values for a given attribute.
- Radio fields (`<INPUT TYPE="radio">`) that enable users to select exactly one of the possible attribute values.

- Checkbox fields (`<INPUT TYPE="checkbox">`) that enable users to select a boolean (yes/no; true/false) value for a specific attribute.
- Hidden fields (`<INPUT TYPE="hidden">`) used to send attributes not visible to users.

While from a technical point of view form-based navigation may be perceived as being somehow similar to link-based navigation, with links corresponding to specific attributes (in both cases an HTTP request is used to move from the current page to another dynamically generated page), there are few differences that make Deep Web sites require other (typically more challenging) access methods. The first difference is purely technical and of rather limited impact. While all hyperlinks are bounded to HTTP GET requests, forms may use both GET and POST requests. POST requests make it possible to send more data to the server; however, they cannot be easily accessed from Web browsers (as a result they are also not "bookmarkable" and "linkable"). This characteristic has a direct impact on accessibility by users but virtually no influence on automatic access to Deep Web. More important challenges concern the way information in HTML forms is organized.

There are two important classes of forms giving access to Deep Web sites. The first category are the forms that use only closed-set fields (list, radio and checkbox fields); in this setting each combination of these fields' value is a functional counterpart of a separate hyperlink corresponding to a few attributes at once. Deep Web sites that use solely closed-set fields require an extra mechanism for parsing forms and HTTP POST requests support; apart from that they behave as basic data-intensive Web sites, with the exception that form filling typically allows users to set the value of more attributes at once than a typical hyperlink. Form parsing, which is trivial in the majority of cases, may require more effort if the accepted values of one attribute change dynamically (using JavaScript or AJAX) after the values of other attributes were selected (e.g. in most flight ticket sales Web sites, after you choose an origin airport the list of accessible destination airports is dynamically filled in) [271].

The usage of open-domain fields (i.e. text fields with no restricted vocabulary) in Deep Web forms is much more game-changing. Typically it results in an infinite or unknown list of possible attribute values. As a result it is infeasible to find all combinations of fields' values and to access all possible result pages. Moreover, while result sets for different combinations of closed-set queries are typically disjoint[10], result sets for open-set queries often overlap.[11]

Accessing all data stored in a Deep Web site requires specific crawling software (in the case of closed-set forms) or may be infeasible (in the case of open-set forms). Moreover, Deep Web sites often provide limited querying support, enabling users to query just a few attributes with no logical and comparison operators. Additionally, sometimes the granularity of values in query forms is not adequate (for example, users need to choose predefined price ranges rather than provide specific range themselves). Thus, the querying of Deep Web data requires query rewriting with respect to source querying capabilities (that need to be automatically

[10]For example cars may belong just to one make or price range. However, in some cases also closed-set queries generate overlapping results; for examples by checking "include results from partner sites" checkbox one receive results that include all result from the set when the checkbox is not checked.

[11]It happens for full text queries when records are returned for any keyword present in the record. However, when open-set fields expect an exact value of attribute (e.g. in case of city or specific production year), no overlap happens.

discovered [47]) and the extra filtering of results acquired from the site [280, 288, 9].

Similarly, as in the case of basic data-intensive Web sites, we assume that at a specific moment of time, the same HTTP request sent to an HTML form provides the same results to different users with different browsing session states.

1.2.3 Personalized Web Sites

The personalization of Web sites, adopted only a few years after the WWW was created [200], is still one of the key trends in Web site design in recent years [35]; while it concerns mostly the sites providing unstructured content or information services (such as Web portals and news Web sites), it can be also applied to data-intensive Web sites [64]. Data-intensive Web sites that may be subject to personalization include topic directories [86], on-line shops [25], and Deep Web sites [51, 9]. The main difference between personalized data-intensive Web sites and the two previously described categories is related to how data are presented to different users. While basic data-intensive Web sites and Deep Web sites provide the same content to different users, personalized Web sites accommodate both navigation structure and displayed data to the profile of the authenticated user. The profile may contain information directly provided by the user (e.g. his demographic data or preferences) or induced from his previous behavior. By contrast, we assume that in pure personalized Web sites, information collected during the current session has no influence on navigation and the range of displayed data.

To download all data provided by a personalized Web site for a specific user, some extra capabilities of data extraction systems (as compared with basic data-intensive Web sites or Deep Web sites) are required. First of all, user authentication (varying from simple HTML forms and HTTP authentication to using the dynamic encryption of authentication data and password masking) needs to be supported. Secondly, authentication information needs to be maintained while accessing data, which may require sending authentication information along each HTTP request or preserving Cookies set by the Web site. It is to be noted that in personalized Web sites it is not possible to access complete data unless we are able to log in as a few representative Web site users.

1.2.4 Adaptive Web Sites

Adaptive Web sites (being a direct descendant of adaptive hypertext research [58]) seem more popular as an object of research than real-life applications. However, adaptiveness is used to some extent by a growing number of Web sites. In adaptive Web sites, the content and navigation structure provided to the user are being continuously adapted to the specific user based on his present and past activities [224]. While this term may refer to Web sites that perform manual adaptation based on data automatically collected for multiple users, in this thesis we use this term solely for Web sites that perform automatic adaptation. The adaptation can affect a single user or all users. In the first case, it is referred to as customization, and results in a number of different versions of the same Web site; the second case is called transformation and results in a single version of Web site automatically evolving over time. In the case of customization, the adaptation may focus on activities during the current session, may keep track of past user activities or implement a long-term user profile. The adaptation may concern the content that is provided to the user, the navigation structure that is proposed, the way content is presented or any combination of the above.

For example, some news portals remove links to previously read stories from their home pages or add links to similar stories, being an instance of navigation adaptation. Some e-commerce Web sites may adapt the list of goods offered at the home page to previously visited items, being an instance of content adaptation. Similarly, some search engines (e.g. Google Personalized Search, currently being the default search mode when user is logged in) use search history for the personalization of search results. [12] Finally, recent research on Web site morphing (i.e. adapting "look and feel" to the cognitive style of user), performed in an experimental BT Group Web site [146] is an example of presentation adaptation.

Depending on the adaptation mechanism used, accessing all data or issuing a query to adaptive data-intensive Web sites vary from relatively simple to infeasible. In the case of transformation-based Web sites the main challenge is related to evolving navigation structure and data location; otherwise it is similar to acquiring information from typical data-intensive Web sites. In customization Web sites, if adaptation affects mostly the organization of data and navigation elements (e.g. order based on relevance), access to required data is as feasible as in the case of basic data-intensive Web sites or Deep Web sites; however, it requires keeping session information and a more flexible approach to accessing specific hyperlinks. By contrast, if the adaptation changes the range of displayed data (e.g. by including only the records and hyperlinks judged most relevant), access to full data may be impossible.

1.2.5 Collaborative Filtering Web Sites

Collaborative filtering Web sites are a special case of adaptive Web sites; in this case information on the profiles and behaviors of other users that are related to a given user (by explicit friendship or trust relation or by pure profile similarity) is taken into consideration [211, 88]. Collaborative filtering techniques are used mostly for navigation adaptation and typically concern a small part of the whole navigation graph. A prominent example of collaborative filtering in data-intensive Web sites included e-commerce Web sites, giving recommendations based on other users' actions ("people who viewed/bought this item also viewed/bought the following items...").[13]

1.2.6 Web Applications

By Web applications we refer to all Web sites that implement some complex logic and are stateful. In our understanding, statefulness means that not only some information about user action is stored within a session (during one visit to the Web site) or between sessions (during consecutive visits), but also that it has an important impact on navigation, displayed data or the behavior of the Web site. For example, a Web site that uses Cookies only to track a user for anonymous market research purposes, does not use the state in any way directly

[12]See: `http://googleblog.blogspot.com/2007/02/personally-speaking.html`, accessed on 28.02.2011.
[13]See for examples: `http://www.amazon.com/` or `http://www.merlin.com.pl`.

visible to the user, and is not a Web application. While Web sites can be modeled in terms of information and navigation dimensions, Web applications include the extra dimension of operations, which are attached to specific information entities (e.g. adding a product to a shopping cart), groups of information entities, including the whole site (e.g. setting currency) or navigation elements (e.g. filtering based on selected links or form values) [38].

Typical solutions able to implement complex logic and to maintain session state are based on server-side Web frameworks (see [269] for an overview and taxonomies) using specific URL-encoded session identifiers or Cookies. In some cases (e.g. in on-line games) some parts of this server state may be shared by a number of users so that the actions of one user have an influence on site presentation to other users. In a number of scenarios, server state is directly connected to real-life items, and on-line activities result in real-life actions; for example, in e-commerce sites a user has an influence on stock by placing an order (affecting other users) and causes real-life consequences (goods being sent to him).

It is clear from the above definition, that Web applications include all adaptive and most personalized Web sites, and exclude basic data-intensive Web sites and Deep Web sites.

1.2.7 GUI-centric Web Sites

The user interfaces of Web sites evolved in parallel to the growing complexity of Web sites' logic. Thus, static HTML files where replaced by simple dynamic HTML files (Web pages that use some client-side dynamics, but do not implement the client-side modification of Document Object Model or asynchronous client-server communication), by moderate-complexity dynamic HTML files (Web pages that often modify DOM at runtime, asynchronously update of some parts of pages and limit full-page navigation to a minimum), and finally by complex dynamic HTML files (complex Web sites using many operations that cannot be modeled in terms of hypertext navigation, such as on-line spreadsheets or Web-based games) [247].

In this thesis, we will call Web sites that use moderate-complexity and complex dynamic HTML to provide complex graphical user interfaces, GUI-centric Web sites. This category is orthogonal to all previously mentioned ones, as all basic data-intensive Web sites, Deep Web sites and Web applications may or may not be GUI-centric. GUI-centric Web sites aim at overcoming some typical human-computer interaction problems present in traditional Web sites, such as disruptive wait time while loading the next Web page, and little interactivity of Web sites (no drag-and-drop support, no animation or dynamic content hiding capabilities, limitation of form-based GUI components, missing some typical Windows applications components such as scroll bars and combo boxes, etc.). Thus, they partially recreate "the seamless user experience of most other desktop applications" [252]. To deal with the imperfections of basic Web technologies two main types of techniques are used. The first group is related to client site dynamism and is responsible for the better interaction of users with a Web page once it is loaded. It is propelled both by the development and maturing of dynamic HTML and general purpose JavaScript libraries, and by the more efficient execution of JavaScript code. The second group is connected to a new approach to client-server communication that becomes asynchronous and requires no reloading of entire pages, sending data to and from server "in the background". While several approaches to client-server communication exist, the most popular are related to sending XML or JSON data over XMLHttpRequest object or IFRAME / hidden frame in a synchronous or asynchronous way.[14]

1.2.8 E-Commerce Sites

Another category, that is not directly related to any of the previously mentioned classes of data-intensive Web sites, consists of e-commerce sites. We cite it here as it accounts for an important fraction of all data-intensive Web sites, and is of special interest to the Web information extraction usage scenarios described in this thesis.

Data-intensive Web sites operating in the area of e-commerce publish several areas of structural or semi-structured data. The first area is related to the conditions of sale of products or services; it typically covers product name (sometimes also product codes and categories), price or prices (e.g. before and after tax or wholesale and retail price), transportation costs, stock levels (the number of items that can be purchased) and order realization time (availability date, the number of days needed for transportation). Another category of information covers product presentation, which mixes highly structured data (e.g. dimensions, quantitative characteristics of products, average user scores) with unstructured descriptions and user comments. Finally, in many areas an important – even if not very abundant – source of information are comparative tests of multiple products (performed by specialized laboratories, magazines or Web sites) typically published both as semi-structured tables and free text descriptions.

The aforementioned areas of data may be presented in multiple ways, including lists of products with basic sales information, pages of individual products, and the view enabling features-based comparison of multiple products.

1.2.9 Social Networking Web Sites

Social networking sites is a new category of data-intensive Web sites that has become popular in the last few years [87, 118, 111, 8]. They contain information about people and relations (trust, friendship) between them. Apart from well-structured information on specific people (hobbies, preferred book authors and movies, professional experience, expertise areas), they provide information on connections between people. While the extraction of information from user profiles at social network sites is similar to extraction from other Web sites, extraction of relations – which are typically at least as interesting as profiles – requires a different Web crawling model and output representation oriented towards graph data.

1.2.10 Generalized Data-Intensive Web Sites

Data-intensive Web sites typically focus on a specific domain and are organized around just a few entities with associated schemas. An interesting exception to this rule are general-

[14]Together, they are often called AJAX; however, this name is imprecise – AJAX (standing for Asynchronous JavaScript and XML) covers a more specific set of technologies: user interface based on dHTML, dynamic user experience using Document Object Model, XML-based encoding of exchanges data, asynchronous communication based on specific XML transport browser objects, and JavaScript as a language connecting all these components [252].

ized data-intensive Web sites, which provide infrastructure for the storage of data about any entities, obeying any schema. While such Web sites are platforms or on-line database management systems, they clearly belong to the universe of data-intensive Web sites. The most prominent example is Google Base[15], but Yahoo! is also offering a solution falling into this category[16]. As such Web sites are highly data-centric, they typically provide data through an API, and require no data extraction. However, we list them here for the completeness of our analysis.

1.2.11 Basic and Complex Data-Intensive Web Sites

Apart from basic data-intensive Web sites, we discussed above eight specific categories of Web sites, which we will refer to as *complex data-intensive Web sites*. We present schematically the relation of these eight groups to basic data-intensive Web site in Figure 1.3. Basic data-intensive Web sites are located at the center of the figure, and for each category of complex Web sites located around the center, we name the key additional challenge it brings in.

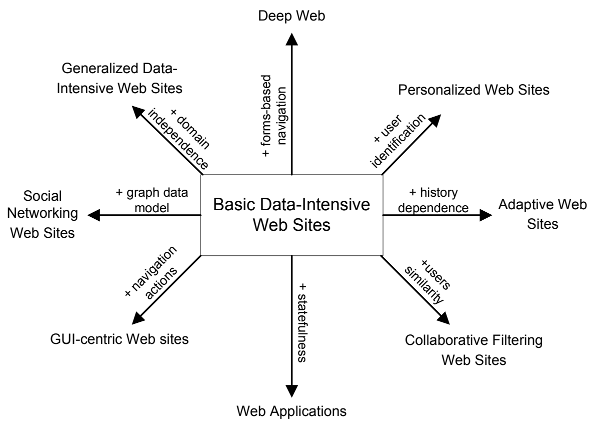


Figure 1.3: Relations Between Different Types of Data-Intensive Web Sites

Deep Web sites bring in the challenge of Web navigation based on HTML forms. Personalized Web sites are challenging because they require user identification. In the case of adaptive Web sites, the content that is served depends on navigation history, and in the case of collaborative filtering it uses similarity between users. Web applications bring in the challenges related to application statefulness. The inherent challenge of GUI-centric Web sites concerns navigation actions that depend on client-side code and user interaction approach. Social networking Web sites are characterized by a graph-based data model. Finally, generalized data-intensive Web sites are domain-independent and therefore hard to model in advance.

Extracting data from complex data-intensive Web sites is the key topic of this thesis. However, our objective is to develop data extraction methods general enough to deal not only with data extraction from all eight categories of complex data-intensive Web sites, but also from basic data-intensive Web sites, which still account for the majority of data-intensive Web sites.

[15]http://base.google.com/.
[16]Databases support, being part of Yahoo! Groups. http://groups.yahoo.com/.

1.3 Web Information in Electronic Commerce

Information access is a critical success factor in any kind of business operation. Although the information extraction methods that are discussed in this thesis are applicable in almost any branch of industry and type of activity, we decided to use electronic commerce as our key application area. There are two reason for this decision. The first reason is related to the electronic form and Web-orientation of e-commerce activities, which allows us to consider more usage scenarios of information extraction than in the case of general problem of "business applicability". The second reason is a growing adaptation of e-commerce solutions in other areas of business activities, and our strong belief that this process will continue, making today's e-commerce an indication of future paradigms in many other industries.

1.3.1 Electronic Commerce Activities

There are a number of definitions of electronic commerce activities. Some of the shortest among them define electronic commerce (e-commerce) as "the automation of commercial transactions using computer and communications technologies" [272], "the buying and selling of information, products, and services via computer networks" [166] and "support for any kind of business transactions over a digital infrastructure" [52]. Another definition states that "electronic commerce refers generally to all forms of transactions relating to commercial activities, including both organizations and individuals, that are based upon the processing and transmission of digitized data, including text, sound and visual images" [238].

Today's Web-based electronic commerce is not entirely a new phenomenon. It evolved from previous technologies (French Minitel network, telephony-based sales) and business models (different types of mail order). Moreover, it uses metaphors from traditional shopping (such as product catalogue or shopping cart). However, many (including Jeff Bezos, the Amazon's founder) consider that the quantitative changes introduced by e-commerce (such as quicker access time and larger offer of goods) are "so profound that they become qualitative" [272], making electronic commerce both a business and cultural revolution.

Electronic Commerce Stakeholders

Today's electronic commerce has a number of classes of stakeholders, varying in terms of their involvement in flows of goods and services, information, and money, in terms of their relation to the production process, their involvement in multiple markets, their buyers (which may be consumers, businesses or government bodies), as well as in terms of value-added they provide. Market participants can be very roughly divided into manufacturers (that create products and provide them to the market), final customers (that somehow consume products), and intermediaries (that operate between these two groups).

The intermediaries may provide very different types of value-added. The most important areas of intermediaries value-added in e-commerce include logistics, assortment, communication, information, support services and value-chain system integration[17]. Different types of

[17]E.g. joining ERP and marketplace software [126] via open Internet platforms [272].

value-added in these areas are listed in Table 1.1. With respect to the mentioned value-added categories, we identified a few typical electronic commerce stakeholders:

- players involved mostly in the physical handling of goods include **manufacturers** (that produce goods and provide their basic specification), **content providers** (that provide digital goods and handle their distribution using computer networks), **importers** (that bring goods from other markets and localize their specifications), **on-line wholesalers and retailers** (that sell large quantities of goods on-line), **on-line shops** (that sell on-line goods to final customers and send them using mail), **click-and-mortar shops** (that offer integrated retailing with on-line order and personal reception) and **social shopping platforms** (which are principally identical to on-line stores apart from their usage of a social networking mechanism to boost sales),
- actors that focus on information mediation but support also the physical transportation of goods include **virtual importers** (that aggregate the offer of multiple foreign suppliers and import goods on demand), **on-line shops using drop-shipping** (that sell goods directly from inventories from a number of suppliers), **virtual franchises** (that sell goods from the inventory of a single supplier) and **offer aggregators** (that aggregate the demand of a number of users, wholesale goods and distribute them to individual users),
- players that implement mechanisms of information exchanges and transaction support include **shop platforms operators** or **e-malls** (that enable individual companies to run their shops and often provide also an aggregated view of the offers of multiple shops), **electronic markets** (that support exchanges between multiple partners from a specific group (horizontal markets) or from a specific industry (vertical markets), **auction services** (a specific kind of electronic market supporting dynamic pricing through different types of price negotiations) and **cybermediaries** (that offer the comparison and sales of products or services of multiple suppliers within a specific industry; widely represented, for instance, in the travel industry, with examples such as Expedia, Travelocity and Orbitz),
- companies that provide purely information-based services include **information brokers** (that gather and analyze market information on demand, for the needs of a specific buyer), **classified publication services** (that enable the exchange of signals regarding willingness to buy or sell specific goods, without providing transaction support), **comparison shopping services** (that enable users to compare prices and other sales conditions of the same goods in different on-line shops), **product search engines** (similar to comparison shopping services, with slightly more focus on search facilities and product description) and **meta-search services** (that perform ad hoc searches in a number of Web sites and integrate search results),
- other transaction-oriented services include different types of **payment services** (that handle credit cards and money transfers), **certification services** (assigning trusted certifates required by secure transaction handling) and **external shop assessment services** (that gather information about the reputation of on-line and off-line businesses).

Area	Value-Added
Logistics [24, 222, 281]	1) bringing (and localizing) goods from other markets; 2) moving goods within one market ("pass-through middleman"); 3) making goods accessible in local points of sales; 4) optimizing transportation and inventory (economies of scale); 5) aggregating buyers' offers; 6) keeping stocks (smoothing away demand and supply fluctuations); 7) "the transportation" of digital goods (e.g. content delivery networks).
Assortment [29, 281, 126, 222]	1) rich product information (including user-generated content); 2) the aggregation of an assortment from multiple sellers; 3) search facilities (lower costs of information acquisition); 4) consulting services; 5) the implementation of better user interfaces; 6) the personalization of product information; 7) the customization of digital or physical goods; 8) making available niche ("long tail") products; 9) combining multiple products (including cross-selling).
Communication [24, 126, 222]	1) the support of sellers and buyers matchmaking; 2) support of negotiations; 3) co-ordination of communication between partners; 4) community building; 5) marketing activities; 6) after-sales service (support).
Information [160, 222, 281, 24]	1) collecting buyer implicit feedback and behavioral patterns; 2) reuse information used in physical value chain;[18] 3) market monitoring; 4) providing domain-specific analytics; 5) providing information for quality assurance;[19] 6) partners' reputation assessment.
Support Services [281, 24, 126]	1) financial services (including payment handling); 2) insurance; 3) accounting and legal support; 4) complexity management (e.g. expert system); 5) decision support systems and analytics; 6) project management; 7) product design.
Value-Chain Systems Integration [126, 272]	1) higher effectiveness of interorganizational business processes; 2) collaborative demand planning; 3) synchronized production planning; 4) joint product development; 5) limiting explosive fluctuation of demand in manufacturers.

Table 1.1: Value-Added of E-commerce Intermediaries

Companies (such as electronic marketplaces) that combine information mediation and transaction support between multiple participants are probably one of the most complex and diverse groups of on-line businesses. They may be sell-side (i.e. focus on marketing, sales and distribution, and "allow selling organizations to interface with a multitude of customers", typically without integration with their IT systems), buy-side (i.e. focus on procurement, typically with integration with buyers' IT systems) and market-type (neutral) [24]. Another classification divides electronic marketplaces into private exchanges (operated by a single company optimizing its sourcing, typically with a long-term relationship and the tight integration of IT systems and supply chain), consortia (closed marketplaces, jointly owned and operated by companies that participate in on-line B2B exchanges), and public e-markets (open markets, operated by third-party providers, providing mostly exchanges of commodities that need little or no customization) [160]. Other characteristics that differentiate on-line marketplaces include type of relationship (ad hoc as in spot buying and selling, or pre-established and long-term), pricing method (fixed or dynamic, based on negotiations or auctions) and type of sold goods ("commodities, MRO-type supplies, direct goods, non-tangible goods and services,

[18]For example, providing end users with location tracking (FedEx) or pension statements (Nordea); previously used only internally.

[19]E.g. based on product inspection, buyers complaints and servicing history.

complex (project) products and services, capital goods") [24]. Finally, on-line businesses may differ with respect to market dynamics and investment specificity (see Figure 1.4).

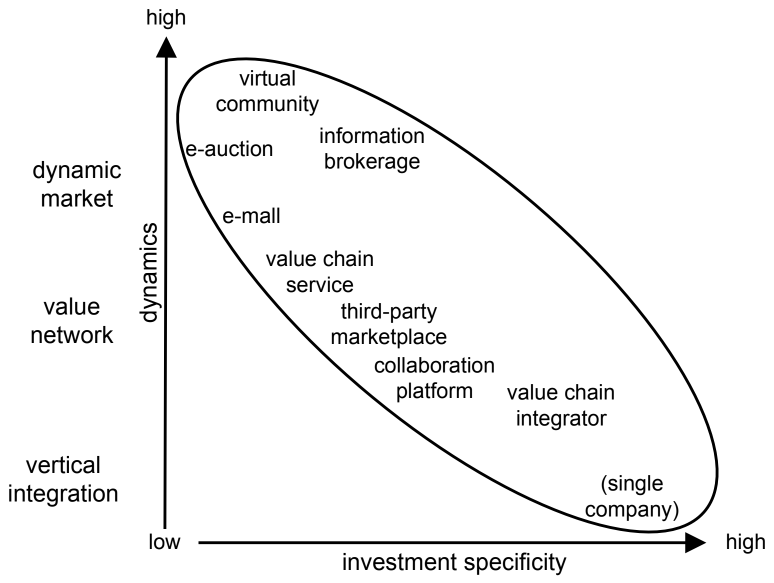


Figure 1.4: Business Models and Value Networks / Dynamic Markets (Source: [263])

1.3.2 The Role of Information in E-commerce

Information has a double role in business and in electronic commerce. Firstly, it can act as "raw material" in the production of goods and services (especially digital ones), becoming part of the final product. Secondly, information is necessary for manual or automated decision making (widely understood, including such activities as strategy formulation and market monitoring for opportunities) [227]. In this thesis we focus on e-commerce applications related to decision-making. Examples of useful information in electronic commerce include:

- products' (technical) specifications and descriptions,
- information on the offers of suppliers,
- information on demand,
- characteristics of Web site visitors' behavior,
- information on customers (individuals and segments),
- assessment of marketing channels' effectiveness,
- information on past transactions,
- information on competitors' offers and positioning,
- user feedback (both in terms of product quality and customer care).

Decision making in contemporary businesses is challenged by a number of factors; many of them are intrinsic for an economy based on knowledge. An extensive list of such factors is presented in Table 1.2. The majority of them are related to two seemingly opposed

Information overload	You have an avalanche of information literally at your finger-tips, but much is conflicting and of uncertain reliability
A galloping rate of change	You must make intelligent decisions about moving targets. What's fact today may be fiction tomorrow
Rising uncertainty	The days of predict-plan-execute are gone. Discontinuities are the norm
Few historical precedents	You must decide correctly within new organizational models (such as virtual organizations) and about new technologies or electronic commerce with little historical experience to guide you
More frequent decisions	Standard operating procedures have been replaced by decisions tailored for individual customers, suppliers, partners, products, and cases
More important decisions	In today's flatter organizations, many are making decisions that have the potential to affect the well-being of the entire firm. These decisions were once made only by those at the top
Conflicting goals	You must deliver short-term, but also experiment and "learn" to prepare for long-term, which is, of course, just the many short-terms yet to come.
More opportunities for miscommunication	Cross-functional and multinational teams are becoming the norm, where function-bound and ethnocentric views of the right answer can quickly derail a decent solution.
Fewer opportunities to correct mistakes	In a fast-paced world, you have less time to correct mistakes and re-establish credibility.
Higher stakes	In our winner-take-all society, fewer people will be big winners. And if you are not one of them, you may well find yourself pushed to the sidelines.

Table 1.2: Why Decision-Making Is Increasingly Challenging (Source: [237])

characteristics of today's economy: imperfect information and information overload.[20]

In contrast with the perfect information assumption in some neoclassical economic theories[21], in real-life scenarios businesses must deal with **imperfect information** for a few reasons. Firstly, some information is not public and not available to all players (a situation referred to as information asymmetry [14]). Secondly, some information may be available in a form requiring time-consuming processing, and subsequently cannot be effectively used to take a decision. For example, a lot of information about stock companies is available by legal obligation. However, its analysis by non-professional traders is time-consuming. Thus, quick decisions using partial information may have better results than belated decisions preceded by a complete analysis. Finally, the information may be incomplete or of incompatible granularity. For example, information about sales in Europe has limited applicability for making regional decisions. Imperfect information's impact on decision making means that market participants compete not only in terms of their primary activities, but also in terms of information access and its analytics [89].

The second challenge, **information overload**, is defined as "the state of an individual (or system) in which not all communication inputs can be processed and utilized, leading to breakdown" [163]. Reinforcement of information overload is due to: the quick development of ICT technologies enabling easier production and transformation of information, the business shift towards information-centric digital goods and services, and the wider availability of more

[20]We use the term "information" here in a broad sense, covering all four levels of the pyramid of wisdom; such understanding is typical for economics.

[21]I.e. the situation when all players have complete knowledge of other players actions at the moment of making decisions; this is the situation assumed for examples in perfectly competitive market models.

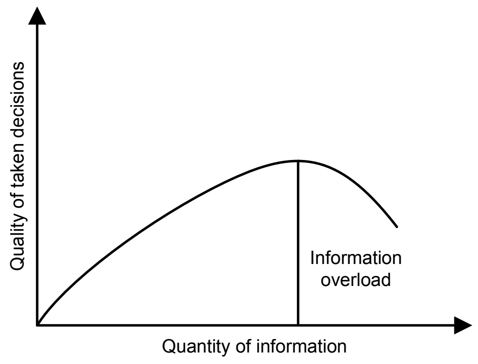


Figure 1.5: Information Overload and Decision Taking

instantaneous communication services and devices [110].

There are two types of information overload: conversational overload (when too many messages are delivered) and information entropy (when messages are not sufficiently structured to be processed or recognized as relevant) [163]. Information structurization efforts, which are discussed in Chapter 2, aim at dealing with both of these problems: they directly limit information entropy, and enable the automated filtering of relevant information only.

Information used in decision-making may come from internal and external sources. Electronic commerce is characterized by a relative abundance of both these categories of information source. Internal information sources include server logs, visitor tracking data, history of transactions and feedback from customers. All these interconnected categories of information "leave a trail of information about consumer demand and tastes, which has a high value in its own right" [281]) and can be analyzed jointly.

The abundance of external information sources has two distinct causes. Firstly, e-commerce forms a complex and dynamic ecosystem with a large number of companies (including foreign market participants) involved in different types of information and product flows. Secondly, a significant part of information from and about other market participants is available in an electronic form (in contrast with many off-line businesses, which provide information about their offer via proprietary software, in a paper-based form, or simply by shelf-based display).

1.3.3 Scenarios of Using Web Information in E-Commerce

The unprecedented accessibility of market information brings the possibilities of large-scale business intelligence and of competition based on analytics [89]. In this section we overview a number of typical scenarios of using external information available on the Web in e-commerce.

For each of these scenarios two basic usage modes are possible. On one hand, information may be gathered continuously in order to use past information in advanced analytical tasks, such as the discovery of behavioral patterns among Web page visitors and customers, sales forecasting or the formulation of company strategy. On the other hand, present information may be accessed ad hoc or repetitively in order to take operational decisions (including the automation of some decisions based on rules or advanced business intelligence [132]) or to discover deviations from forecasts and strategies (in order to introduce corrections).

Suppliers Monitoring

When taking purchase decisions all trade companies need to consider three kinds of information. The first area concerns the estimated demand for products and defines how many goods will be possibly bought from the company. The second area, internal to the company, concerns the current and minimal stock for specific products, and defines how many goods the company needs to buy. Finally, the third area covers assortment, prices, current stock and the delivery date of goods offered by specific suppliers, and directly influences decisions which goods should be bought from specific traders. The use case of suppliers monitoring focuses on the third area of information.

The range and freshness of supplier information used in decision making is typically limited by two factors. The first is the number of possible suppliers of goods, which is very high in large and competitive markets such as the U.S. or U.E. branch markets. The second is a very quick pace of change in the list of market participants (creation and bankruptcy of companies), goods offered (introduction or withdrawal of products), prices, stock levels and possible delivery dates. As a result, even if the criteria defining “best offer” goods remain the same, different suppliers are optimal at different time spans.

Supplier monitoring consists in the continuous acquisition and integration of information about prices, stock levels and delivery dates, in order to choose the best supplier at a specific moment. This scenario covers two main kinds of purchase decisions. The former are typical operational purchase decisions, i.e. purchases needed to keep stock levels above minimal values. The latter are occasion-seeking purchases, i.e. unplanned purchases stimulated by exceptionally good offers.

In typical manual scenarios, suppliers monitoring is time-consuming and costly, resulting in the rare update of information from a limited set of sources. In the case of the full automation of supplier monitoring (thanks to APIs, structured formats of offer file or the application of information extraction), larger coverage (in terms of both the number of sources and monitored items) and more frequent information updates become possible.

Competitors Monitoring

Each company functions within a specific competitive environment. In case of electronic commerce this environment tends to include a significant number of entities of different types (varying from on-line shops to different kinds of infomediaries). Information about competitors’ operations have a direct impact on almost all areas of business activities; it is especially important in the case of pricing and the range of offered goods. Price is one of main factors of competitive strategy; for widely accessible products it typically needs to be similar or lower than that offered by competitors, while for unique or rare products a higher margin of profit is possible. Another aspect of competitive strategy is related to assortment. It can vary from niche strategy (i.e. offering goods that are unique or almost unique) to offering goods or services with value-added that are not accessible to competitors. Both the formulation and monitoring of pricing and assortment strategies require competitors monitoring i.e. the continuous acquisition of information about competitors’ offer.

In this case both the number of Web sites to be monitored and the pace of change are typically higher than in the case of suppliers monitoring. Thus, while it is possible to perform basic competitors monitoring in a manual way, only the automation of information collection enables really quick reaction to competitors' actions.

Foreign Markets Monitoring

Due to progressing globalization and quicker information exchanges, the prices and availability of goods in a specific market are strongly connected with other economies (the strength of this connection and specific key markets depend on the business area). In many cases the monitoring of changes happening in other key markets allows companies to anticipate changes in their markets and to properly react before they happen. Depending on the scenario, observed changes may concern the prices of specific products or groups of products, as well as the assortment of key players in foreign markets. When combined with currency exchange rates, such information may be used to take decisions related to purchases, pricing and the range of offered products.

Distribution Network Monitoring

Manufacturers and importers are primary entities that introduce products to sales networks. However, afterwards they often lose control over where, how and at what prices their goods are sold, and how they are positioned against competing (substitute) and complementary products. At the same time this information would allow them to assess the real extent of the distribution network (and its diversity in individual areas or sales media), to verify if the planned positioning of products is followed, to have some insights into brand recognition and product exposition (again, especially as compared to competitors' products), and even to detect possible fake goods sales.

That is why manufacturers and importers typically are interested in monitoring the distribution network i.e. in continuously acquiring information from multiple points of sale that possibly offer their products. This activity can be partially performed based on internal data (describing sales to specific customers), and data acquired from partner companies (e.g. franchises or co-operating wholesalers). However, these sources have limited a range and cover only some aspects of sales (typically ignoring exposition and information about competitive products). As a result, they need to be supplemented by information collected by the inspection of points of sales. In the case of brick-and-mortar shops this method is costly, so information gathering is performed rarely and is limited in range. It is much easier in the case of on-line shops; however, even in e-commerce manual inspection is feasible only to some extent. It is thanks to automation that large-scale on-line distribution network monitoring becomes possible.

The frequent monitoring of multiple on-line points of sale enables not only the competitive analysis of prices, exposition and stock levels, but also the detection of events that require reaction (e.g. manufacturers' products being gradually replaced by their substitutes, or sales of manufacturers' products at extremely low prices).

Data Migration Between Co-operating Parties

Four aforementioned scenarios consisted in using publicly available (and not copyrighted) information for analyses without the explicit consent of monitored parties (it is possible in the case of suppliers monitoring, but even in this case it is not necessary). Other possibilities are related to agreed-upon information exchanges related to constant or temporary co-operation between companies. In this case, information suppliers may provide an API enabling flexible access to structured information, or provide information in specific machine-processable formats (e.g. in XML or RDF). In cases where no IT systems adaptation is feasible, information suppliers may also agree that information from their Web site is acquired from their Web page; while this approach will use the same technologies as the four previously described scenarios, it can be legally extended to specific copyrighted content or data (e.g. test results, reviews etc.), within the scope of the agreement between companies. Shopping comparison Web sites are a typical example of businesses that may be interested in acquiring information from many partners without making them introduce any adaptation to their Web sites and underlying systems (thus, lowering the entry barriers).

Market Data Collection by Analysts

In all previous scenarios the information flow is related to the operations of specific businesses (such as purchases, sales or specific comparison services). However, there are groups of entities that are interested in the collection of market information without participating in market activities. These entities include specific groups of public bodies (such as EUROSTAT in European Union, the Census Bureau in the U.S. and GUS in Poland), as well as companies focused on market research (such as consulting or marketing companies). Market data or information collection, which can be performed at a single-time or in a continuous way, consists in gathering information about the activities of a relatively large number of market participants and the products or services they offer, in order to enable market diagnosis (e.g. the assessment of price level and its changes, the discovery of actual distribution networks or the analysis of the ranges of goods offered by specific classes of businesses). While multiple methodologies (e.g. surveys, polls, focus groups) and information sources (e.g. interviews, obligatory reporting) exist for market data collection, in the case of e-commerce some part may be acquired automatically from Web sites of e-commerce players.

1.4 Summary

In this chapter we discussed the role of Web information in electronic commerce. We started by introducing definitions of basic terms such as data and information, followed by a discussion of the different levels of data and document structures (Section 1.1). Next, we presented and compared different types of data-intensive Web sites, providing semi-structured information (Section 1.2). Finally, we described preliminaries of electronic commerce with special attention paid to role that information plays in on-line business, and to scenarios of using information acquired from the Web in e-commerce activities (Section 1.3).

Chapter 2

Web Information Extraction

While recent years have seen a rapid growth of multimedia content (including photos, audio, movies, Flash animations and 3D models), with sole Google sites hosting over 240 million videos[1] and with a YouTube number of videos 2006-2008 growth rate of above 1900%[2], the vast majority of Web content consists of different kinds of textual documents (with or without accompanying media, formatting, etc.). They are provided in a number of different formats (HTML, PDF, XML, DOC, etc.) and vary from plain text to semi-structured documents containing data records. In case of HTML documents, that typically contain multiple media, an important area of Web pages is still covered by textual content [184].

This makes different methods of bringing structure and semantics to the Web (including Web information extraction) an active field of both research and commercial activities. In this chapter, we study the technological environment of our research [152], i.e. different approaches to structuring Web content (Section 2.1), the problems of information extraction and Web information extraction (Section 2.2), and the technological challenges of dealing with data-intensive Web sites introduced in Chapter 1.

2.1 Bringing Structure to the Web

In recent years, a lot of effort has been directed towards adding more structure and semantics to Web content in order to make it processible by computers [115]. These efforts can be roughly divided into two groups: the bottom-up approach and the top-down approach.[3]

The **bottom-up** approach advocates the manual introduction of structure and semantics directly by the authors of Web sites or Internet users. Examples of bottom-up activities include moving data to structured or semantic formats (such as XML [55] and XSLT [76],

[1]Source: search for "*" in Google Video gave 241 million results (11.09.2008).

[2]According to a Wall Street Journal article (see: `http://tinyurl.com/youtubeNumbers2006`, accessed on 01.10.2011) in August 2006, YouTube contained 6.1 million videos; in September 2008 the number has grown to 117 million (counted using query "site:youtube.com" in Google Videos Web site, 12.08.2006).

[3]We follow here the division proposed in [159]. However, we apply it more generally to content enrichment with structure and semantics, while in the cited article it concerns just the development of Semantic Web.

RSS [3], CSV, RDF [43], RDFa [12] or OWL [203]), providing data access APIs (e.g. by using SOAP [137], REST [106] or XML-RPC [274] Web services, with SOAP, XML or JSON[4] data representation) and content enrichment with meta-data (examples include Dublin Core[5], microformats[6] or ontologies)[7].

This approach is pulled by the gradual usage of additional structure by search engines, large Web sites and Web browsers[8]. At the same time, the bottom-up approach is pushed by a growing number of content management systems, popular Web applications and on-line publishing Web sites[9] that have microformats and RDFa support.

In spite of the progress of the bottom-up approach, its adoption remains relatively low. While the number of APIs[10] and standard vocabularies (ontologies) grows quickly in relative numbers, they are still relatively obscure technologies adopted mostly in a few niches. There are at least three reasons for this situation.

Firstly, more structured content is still a low-importance factor for customers and major search engines. Secondly, implementation of the bottom-up approach requires costly changes to legacy Web applications and typically involves more effort during content authoring. Finally, in some areas (e.g. in e-commerce) making data too easy to process may be against the interests of the Web site's owner (e.g. because it makes prices too easily comparable between sites, or because it gives too easy access to product list changes to one's competitors)[11].

All of these reasons propel the development of the top-down approach based on the automated processing of Web content. In this approach, better known in the research community as Web mining, the hints present in the content are used altogether with external resources in order to extract information with clear structure and semantics from Web content, without direct co-operation of individual Web sites. The output of automated top-down processing is more "digestible" (e.g. better structured, aggregated, organized or summarized) to users or machines than original content.

The very basic example of such an approach is general purpose search engines. They use some general algorithms to impose on any kind of Web content structure consisting of the internal search engine's index (often using basic linguistic resources [34]), the structured

[4]See: `http://www.json.org/`.

[5]See: `http://dublincore.org/`.

[6]Such as hCard (see: `http://microformats.org/wiki/hcard`) or hCalendar (see: `http://microformats.org/wiki/hCalendar`.)

[7]such as FOAF [57], SIOC [56] or SKOS [209].

[8]Including Yahoo!'s Search Monkey search engine (available at `http://developer.yahoo.com/searchmonkey/` until October 2010), Google's rich snippets using microformats and RDFa, Facebook making use of Open Graph Protocol and FireFox Operator (see: `https://addons.mozilla.org/en-US/firefox/addon/operator/`, accessed on 01.10.2011) plugin exploiting multiple types of annotations.

[9]Including Content Management Systems such us PostNuke and Drupal and Web sites such as Digg, Facebook, Flickr, Salesforce, LinkedIn, Technorati, Twitter, Upcoming.org. See: `http://microformats.org/wiki/implementations` for a more extensive list.

[10]ProgrammableWeb Web site was cataloging about 40 new APIs per month in first quarter of 2008 with dynamics growing to 60 APIs per month in July 2008. They reached the total number of almost 900 APIs in July 2008. See: `http://blog.programmableweb.com/2008/04/02/700-apis-and-counting-now-at-40-per-month/`, accessed on 02.09.2008 and `http://blog.programmableweb.com/2008/08/01/60-new-apis-in-30-days/`, accessed on 02.09.2008.

[11]Indeed, in some domains it is not uncommon to use different levels of HTML obfuscation to make data less and not more machine-processable.

presentation of data results (composed at least of title, content snippet, URL, and modification date) and query relevance function (using many features inherent to the content, to its location on the Web, as well as some general knowledge repositories[12]).

It is to be noted that bottom-up and top-down approaches are interrelated in two important ways. Firstly, a number of Web sites and Web applications that allow for manual annotation of Web sites in an external and generalized manner (examples include del.icio.us, digg, SpinSpotter, dapper and Yahoo! Pipes), clearly combine both approaches. Secondly, the more structure becomes available thanks to bottom-up approaches, the more of it could (and should) be used by top-down approaches.

The bottom-up approach is mostly beyond the scope of this thesis. By contrast, Web information extraction, which is central for this dissertation, combines a number of methods used in the top-down approach to structure and semantify Web content. That is why, before defining Web information extraction and presenting its challenges, we first present different Web mining techniques.

More formally, we define Web mining as the "use of data mining techniques to automatically discover and extract information from Web documents and services" [174]. Four main steps of typical Web mining activities are: resource finding ("retrieving intended Web pages"), information selection and pre-processing ("automatically selecting and pre-processing specific information from retrieved Web resources"), generalization (the automated discovery of "general patterns at individual Web sites as well as across multiple Web sites"), and analysis ("validation and/or interpretation of the mined patterns") [102, 174]. Three main areas of Web mining concern Web content mining, Web structure mining and Web usage mining [174, 199, 53].

2.1.1 Web Content Mining

Web content mining is "the application of data mining techniques to content published on the Internet, usually as HTML (semistructured), plain text (unstructured), or XML (structured) documents" [171]. It focuses on acquiring new or better structured information from a wide variety of Web documents (including multimedia, information contained in digital libraries, enterprise portals and non-Web resources available with WWW technologies [174]). It is the most active field of Web mining research and covers a variety of tasks.

The first of these covers various forms of assigning groups or labels to documents, including the classification (e.g. [216]), categorization (e.g. [96]) or clustering (e.g. [259]) of (hyper)text documents. While many early approaches in this area use only document text, many recent solutions exploit HTML documents structure (e.g. HTML tags [125] or the layout of the referring page [250]).

Another area of Web content mining covers text summarization methods varying from topical segmentation [120, 109], the discovery of keywords and key phrases (see for example [264]), and key blocks of Web pages (see for example [90]), to solutions that combine hierar-

[12]For example, Google used over 100 factors in 2002 (see: http://searchenginewatch.com/showPage.html?page=2164991, accessed on 15.07.2010), and more than 200 in 2008 (see: http://news.cnet.com/8301-1023_3-9999038-93.html, accessed on 15.07.2010).

chical text segmentation and segment labeling in order to provide hierarchical summaries of documents [73].

Few tasks in Web content mining are geared towards acquiring information on classes, relations and specific instances of classes from Web content. Typical tasks in this area include schema or type hierarchy discovery, relations mining, and ontology learning and population (see [275] for an in-depth overview).

With explosive growth in the size of social content on the Web, specific methods of text analysis emerged, including opinion mining (see for example [92] or [286]), the analysis of emotions in user-generated content (see [5] for an extensive overview) or mining knowledge from user tags (see for example [277]). Similarly, the gradual adoption of Semantic Web technologies is followed by the birth of Semantic Web mining [171].

Another large research area is multimedia content mining, covering problems such as line and edge detection, segmentation into scenes or tracks (video & audio), face recognition, image labeling, image segmentation, content-based and concept-based indexing, image/videos/scenes clustering and classification, human motion tracking (video), by-example querying, and video summarization (see [162] for an extensive state-of-the-art review).

Finally, information extraction which is the topic of this thesis, has for years been one of the most important topics in Web content mining. Due to its importance to this thesis, we discuss information extraction it in detail in Section 2.2.1.

2.1.2 Web Structure Mining

Web structure mining focuses on analyzing the topology of the hyperlinks graph between Web pages. Research in this area is mostly inspired by more mature fields using the analysis of large graphs such as social network analysis and bibliometrics.

One of key Web structure mining research areas is in **measuring the importance of Web sites**. A basic assumption of such algorithms is that a hyperlink from page A to page B typically reflects the trust author of page A has in page B.

The first well-known algorithm in this category is Hyperlink-Induced Topic Search (HITS) [168] (with a few further improvements proposed, for example, in [186]). It was based on the recursive, hyperlink-based calculation of the quantitative authority of two types of pages: authorities (high-quality pages relevant to a given query) and their hub pages (pages that link to multiple authorities).

An even more renowned Web site importance calculation algorithm is PageRank [217], which underlies Google's success. It uses the Web graph to iteratively and globally calculate the probability that a given page will be visited by a random surfer, i.e. the user that continuously follows a random outgoing hyperlink of the current page. Multiple PageRank improvements were proposed, including simplified calculation [167], personalization [147] or extended assumptions (e.g. including the "back" button [261]).

Another prominent category of Web structure mining tasks is community discovery based on dense subgraphs of the Web graph. This task consists in finding groups of strongly interconnected Web sites, e.g. organized around vertical portals or connected into webrings. Examples of research into this area include [177], focusing on the discovery of emerging communities in a process called trawling, and a number of papers focused on link spam detection (e.g. [241]).

Other tasks using Web structure mining include Web site complexity measurement and Web page categorization [174]. For example, the text categorization methods proposed in [125] use both the document content and the features of pages linking to it.

2.1.3 Web Usage Mining

Web usage mining "analyzes results of user interactions with a Web server, including Web logs, clickstreams, and database transactions at a Web site or a group of related sites" [171]. It focuses on finding interesting patterns in the sessions and behaviors of individual users and "intelligent crowds". When applied to multiple users, Web usage mining techniques provide general information on user behavior patterns in the Web sites.

At large, Web usage mining uses three sources of information [258]. In the case of server-level collection of data, Web server or Web application logs (containing list of HTTP requests and their details) are used as a data source. While this method tracks all HTTP requests, it requires full access to the server, provides limited information, and in many cases makes hard tracking multiple requests of a single user. In the case of client-level collection of data, a JavaScript library (such as Google Analytics[13]) or browser plugin (such as Google Toolbar[14] and Alexa Toolbar[15]) are used to track user data. This approach can provide more precise data about a user's system (e.g. screen resolution and colors), and behavior (e.g. time spent on a specific page, information on page scrolling or other client-side events) and help session tracking. However, its scope is limited to users with JavaScript on (and no anti-tracking plugins activated), and with tracking plugin installed, respectively.[16] Finally, proxy-level collection of data uses Web proxy servers logs. It gives access to the same information as server-level tracking, but makes it possible to give information about all Web sites that are visited by a specific community of Web proxy users.

It is to be noted that only proxy-level collection and client-level collection with widely used plugins or JavaScript libraries are pure top-down approach activities, while server-side collection belongs more to the bottom-up philosophy.

The patterns acquired by Web usage mining may concern some populations of Internet user or individual users. In the first case, they may be further used in business intelligence (e.g. the assessment of marketing strategy efficiency, the discovery of customer clusters, the analysis of products or documents popularity) [59, 6], improvement of the Web site (e.g. modification of the navigation structure or improvement of the visibility of some elements) [257] or automated recommendations for other users (e.g. different variants of collaborative filtering [211, 88] or the usage-based ranking of products or documents [196]). By contrast, data concerning a single user may be used to build individual profiles enabling automated content and navigation personalization [98, 226].

[13]See: `http://analytics.google.com/`.

[14]See: `http://toolbar.google.com/`.

[15]See: `http://www.alexa.com/site/download`.

[16]For example, Alexa Toolbar's bias towards webmasters and search marketing sites is well-known (see: `http://www.norvig.com/logs-alexa.html`, accessed on 15.09.2011).

2.2 Information Extraction

Information extraction is one of the most important top-down Web content structurization tasks. This section presents of a few approaches to defining information extraction and Web information extraction, followed by a description of key information extraction tools i.e. wrappers and data extraction rules.

2.2.1 Information Extraction and Web Information Extraction

Eikvil [97] defines information extraction in the following way: "The goal of information extraction (IE) is to transform text into a structured format and thereby reducing the information in a document to a tabular structure. Unseen texts are taken as input to produce fixed-format unambiguous data as output. Specified information can then be extracted from different documents with a heterogeneous representation and be summarized and presented in a uniform way." A shorter definition provided in [202] states that information extraction is "filling slots in a database from sub-segments of text". The same authors draw a distinction between information extraction and information extraction from the Web, stating that the latter involves less grammar but more formatting and linking.

In this thesis we will use three separate terms: information extraction, information extraction from texts and Web information extraction. By information extraction (IE), we will mean any instance of acquiring information (as defined in Section 1.1) from any type of unstructured or semi-structured documents.

By the term information extraction from text (IET), we will refer to the acquisition of information from any unstructured textual documents expressed in grammatical natural language. Exemplary tasks of information extraction from text may be "to find management changes reported in the Wall Street Journal or to identify the targets of terrorist attacks reported in the newspapers" [213] or to extract named entities (e.g. people and organizations) from on-line news.

Finally, we will use the term Web information extraction (WIE), meaning the acquisition of information from any semi-structured documents accessible using the HTTP protocol, whenever they reside in intranet or Internet Web sites. The term "screen scraping", widely used especially in non-science technical publications, can be treated as a synonym for Web information extraction; however, we will not use this term in the reminder of this thesis.

It is clear from the provided definitions that information extraction is a hypernym of both information extraction from text and Web information from the Web, and these two terms are in turn disjoint.

This thesis focuses on Web information extraction techniques and applications. Thus, apart from the short comparison of IET and WIE that follows, we will not further elaborate on information extraction from text. For detailed description of IET techniques we refer readers to [136, 213, 202, 265].

While information extraction from texts and Web information extraction has a seemingly similar objective (to structure some less structured documents), these tasks differ in some significant ways. First of all, IET strongly relies on natural language technologies such as corpus-based and symbolic natural language processing (e.g. language modelling, shallow text processing, part-of-speech tagging), while WIE explores mostly the structure and layout of parsed documents. Even if a specific WIE solution uses some natural language techniques (e.g. segmentation or the detection of key phrases) or resources (e.g. a thesaurus), the extent of performed natural language processing is much lower than in extracting information from texts. As a consequence, IET solutions are much more language- and domain-dependent.

Secondly, the majority of documents subject to Web information extraction are created dynamically from structured or semi-structured data repositories (such as databases and XML files); thus, WIE's objective is typically to rediscover some structure that was encoded in a Web page by some information system, using rather stable rules or templates. By contrast, IET is performed on natural language documents that are authored by people (and not information systems); thus, the rules used to encode information in these documents are much less stable and precise. For the same reasons, the hints used in extraction (syntax, clue phrases and the n-gram of words in the case of IET; tags, keywords and layout in the case of WIE) are typically similar in a big number of documents processed by WIE and can vary significantly in the case of documents subject to IET.

Finally, IET is typically performed on individual documents (possibly acquired by information retrieval techniques), while WIE may be also performed on more complex information sources (such as Deep Web databases or Web applications). Indeed, even if the research problem of extracting information from individual Web documents has been subject to a multitude of studies, our focus in this thesis is on extracting information from complex data-intensive Web sites that may require data-driven navigation, the construction of records from data spread into multiple pages, dealing with technological and user interface complexities, handling stateful Web sources, as well as working with graph-based data models.

2.2.2 Wrappers and Extraction Rules

As observed by Muslea [213], "a key component of any IE system is its set of extraction patterns (or extraction rules) that is used to extract from each document the information relevant to a particular extraction task". Extraction rules are declarative expressions following the syntax of a specific language ([213] cites several examples; however, in today's scenarios other popular languages such as XPath or regular expressions can be used), that specify what information should be extracted and/or how it should be extracted.

Extraction rules specify what should be the output of the structurization of unstructured or semi-structured documents. Moreover, they may also provide information on how the documents should be acquired and what transformations are needed to impose the desired structure on extracted information. Apart from systems that perform fully automated extraction, any information extraction tool requires the manual, semi-automatic or fully automated creation of extraction rules.

Extraction rules are declarative in their character, i.e. they are not expressed in a directly executable code. Therefore, a component able to parse their syntax and use them to perform actual information extraction is required. In this thesis we will refer to such a component by the name of extraction rules execution engine (or execution engine). To execute a given set of extraction rules an execution engine may need some external resources such as output schema definition, thesauri or ontologies.

Another central concept in Web information extraction, referring to a component performing information extraction from specific sources, is a wrapper. A wrapper is defined as follows: "A wrapper can be seen as a procedure that is designed for extracting content of a particular information source and delivering the content of interest in a self-describing representation. [...] In the Web environment, its purpose should be to convert information implicitly stored as an HTML document into information explicitly stored as a data-structure for further processing." [97]

In most data extraction systems, wrappers need to be created for each data source. "The problem of generating a wrapper for Web data extraction can be stated as follows. Given a Web page S containing a set of implicit objects, determine a mapping W that populates a data repository R with the objects in S. The mapping W must also be capable of recognizing and extracting data from any other page S' similar to S. We use the term similar in a very empirical sense, meaning pages provided by a same site or Web service, such as pages of a same Web bookstore. In this context, a wrapper is a program that executes the mapping W." [262]

Eikvil [97] states that a wrapper acquires information by performing four steps: a) it accepts queries, b) it fetches relevant pages, c) it extracts information and d) it returns the result.

In this thesis we will understand wrapper as being any software component able to acquire information in a desired form from a specific information source, with all the extraction rules and external resources it requires (if any). For example, the same execution engine with the same external resources and a different set of extraction rules adjusted to three Web sites, will be considered as three separate wrappers.

We will also divide information extraction systems those into using procedural wrappers (i.e. those using wrappers developed manually for each data source directly in some executable or compilable programming language), systems using declarative wrappers (i.e. wrappers consisting of the general execution engine and declarative extraction rules developed for specific data sources), and wrapperless systems (i.e. systems that perform automated data extraction without the need for wrapper generation for each data source).

2.3 Challenges of Information Extraction from Data-Intensive Web Sites

In the previous section we described the nature of Web information extraction and presented some general methods of coping with this task. In this section we review the most important aspects of contemporary Web sites and Web information extraction tasks, which make the development of a general-purpose Web information extraction system a challenging task.

To compile a complete list of challenges of Web information extraction we combine four distinct sources of information. The first of them is a literature review in the area of information extraction. The second source consists in an analysis of users' interaction with different types of complex Web sites. The third source is a review of a variety of different real-life Web sites. Finally, we also analyze a few technical usage scenarios of Web information extraction.

2.3.1 Literature Review

In [180] the author lists the following challenges for wrapper induction systems (extending the list previously proposed by Hsu [155]):

- **missing attributes** (missing or null values, sometimes with an implicit default value),
- **multi-valued attributes** (multiple values assigned to a single attribute e.g. multiple cities served by the same hotel),
- **multiple attribute orderings** (different orders of attributes in different records),
- **disjunctive delimiters** (multiple delimiters for the same attribute),
- **nonexistent delimiters** (two attributes concatenated with no delimiter between them; this situation is typical for attributes in codes and URLs),
- **typographical errors and exceptions**,
- **sequential delimiters** (delimiters composed of multiple elements present, e.g. the sequence of tags embedded in one another),
- **hierarchically organized data** (data not structured into simple lists or tables).

The importance of missing (optional) attributes and permutations of attributes (different attribute orders) is also underlined in [97].

In [219] the challenges related to navigation in complex data-intensive Web sites are discussed. The authors state that "in [their] experience, the most difficult problem involved in commercial web wrapper generation is not parsing (which has been the main issue addressed in literature), but creating the navigation sequences required to access the data." Among the challenges in data access they list dynamic HTML, HTTPS, HTML frames, JavaScript support and issues related to "complex non-standard session maintenance mechanisms based on randomly generated session-ids".

A similar focus is presented by Firat in [107], who argues that, "a comprehensive web client" must handle GET and POST HTTP methods, HTTP responses parsing and understanding (including redirects), Cookies, SSL, certificates and authentication. It should be able to interpret JavaScript as well.

Many authors (e.g. [72, 97]) underline that one of key difficulties in data extraction is related to the diversity of content and navigation structure subject to extraction processes. It means, on the one hand, that data extraction in some cases needs to deal with extremely difficult types of content; and on the other hand, that in other cases it should be able to use hints contained in well-structured data sources.

Another important challenge is related to how the results of search engine queries are split into pages. In [72] three possibilities are listed:

- **One-level one page result** is the situation where all results are listed on a single page.
- **One-level multi-page result** is the situation when results are split into multiple pages by limiting the number of results per single page.

- **Two-level pages** occur when, apart from one or multiple pages containing a list of results, a single "full info" page exists that contains more details for each data record in the results page.

Chang and colleagues in [66] list the following "task difficulties" related to characteristics of specific Web-based data sources:

- the **type of input documents**: structured, semi-structured or unstructured,
- the number of **tuples in single page**: one of multiple,
- the number of **pages** used **to extract single tuple**: data contained in single page or spread into multiple pages,
- the presence of **optional or multi-value attributes**,
- the multi-ordering (**permutations**) of attributes,
- the **variable format** for single attribute,
- the presence of so-called **untokenized attributes** (i.e. multiple attributes merged into single token; it is the case of some codes, such as COMP4016, which is composed of two parts, or of URLs that may also contain partial data not separated from other tokens).

A very technical perspective on challenges in accessing and preserving Web content is presented in [201].[17] They identify the following groups of challenges:

- parsing and interpreting alternate content types such as Flash, PDF, XML, RSS, RDF, MS Office formats, Java applets,
- dealing with different types of selection-based or open-domain (as in keyword search) form fields,
- multiple applications of JavaScript, including dealing with dynamic menus, scripts opening new windows, dynamic time-bound elements generated on client-side and dynamic URL generation,
- the acquisition and parsing of multimedia content (both non-streaming and streaming approaches),
- dealing with authentication and CAPTCHA-like fields,
- non-Western character encodings (charsets),
- dynamic server-side technologies (including content depending on the number of visits),
- embedded proprietary software (e.g. map applications),
- dealing with Cookies (both for user authentication and session coherence validation),
- limitations of robot access (robots.txt files, technical methods of robot detection, per-user content access limits).

Apart from works listing multiple challenges, as described above, multiple researchers have characterized worldwide a number of very specific individual data extraction tasks and challenges. Examples include: schema mapping [278, 149], rewriting user queries onto heterogeneous data sources [141], Deep Web sources discovery [187, 81, 37], dealing with limited query capabilities [221, 225], data extraction from specific binary formats [103], data extrac-

[17]While Web content archiving is a distinct problem from Web information extraction, it shares the same challenges related to Web navigation and GUI interaction.

tion using graphical features [183, 233], fully automated data extraction based on template similarity [284], probing Deep Web sites [135], meta-search [191, 150] and source selection [268, 256].

2.3.2 User Interaction Model

Information extraction from complex Web sites often needs to start with mimicking (at least to some extent) user behaviors in this site – but at a larger scale and speed. From this point of view, understanding user interactions with Web sites is one of key steps in analyzing Web information extraction challenges.

We analyze them in three ways. Two of these ways focus on single interactions of a user with a Web site by using a Web browser application. The first view concerns the forms in which that data are stored and represented during user interactions with a Web site, as well as the transformations between them. The second approach consists of a detailed, technical analysis of how user interaction with a Web site is performed. The third perspective concerns complete user navigation in a Web site that consists of multiple individual interactions.

Data Form Changes During User Interaction With a Web Site

During user interaction with a Web site, data change their form via a sequence of well-distinguished transformations. We summarize these transformations in Table 2.1.

Transformation	Description
Information need to user action	The user performs some Web actions that (s)he finds useful for fulfilling his/her information need.
User action to HTTP request	Client-side Web application logic or Web browser interprets user actions and issues corresponding HTTP requests.
Request to query	Server-side Web application logic interprets received HTTP requests and issues corresponding query to the database.
Query to raw data	Database returns raw data corresponding to issued query to server-side Web application logic.
Raw data to encoded data	Web application encodes received raw data in a form that will be interpretable by client-side Web application logic or Web browser by using some data encoding templates.
Encoded data to user content	Client-side Web application or Web browser integrates received encoded data into content displayed to the end user.
User content to information	User (or some agent) interprets presented content and extracts useful information out of it.

Table 2.1: Information Transformations During User-Web Site Interactions

Technical View of User Interaction

To analyze a single user interaction with a Web site, we developed a model, which is schematically depicted in Figure 2.1. A detailed presentation of its all components and the stages of user interaction with a Web site are presented in Appendix A.

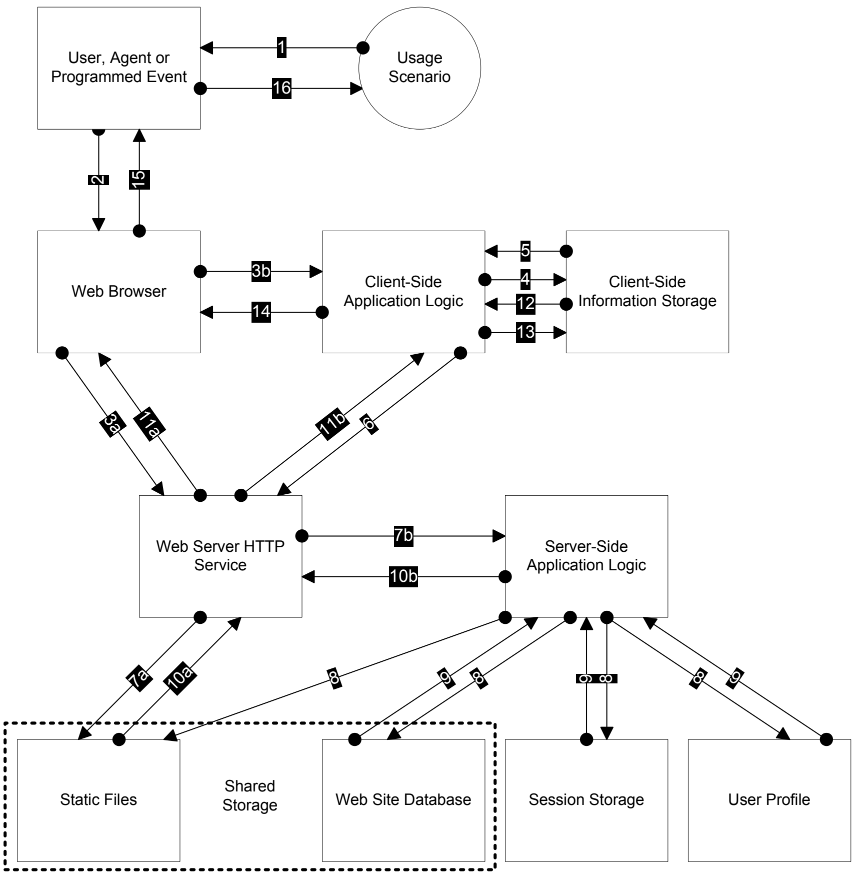


Figure 2.1: Schematic View of User's Interactions With Complex Web Site

There are a few key challenges related to the static structure of the presented user interaction model. The first group of challenges is "cognitive", i.e. related to the fact that a user has no insight into some of the static components and their behavior.

Firstly, we have no access to server-side Web application logic and the database model. We do not even know if the content returned by the server is static or generated. If it is generated, there is no way of knowing the complete code of server-side Web application logic or of verifying if content is generated based on the database, session state or user profile. Possibilities include purely algorithmic applications that use none of them, to complex, data-driven, collaborative-filtering and adaptive Web sites that use all these components.

As a result, the server-side behavior of a Web application can be modeled only based on a sample of pairs of requests and responses. Another related challenge is that normally negative examples (i.e. request-response pairs that are not possible in the Web site) are missing, making some of machine learning algorithms not applicable or hard to apply for Web site modeling.

Moreover, both server-side and client-side logic tend to change over time in an unpredictable way. While in the case of client-side logic the challenge is to react to changing code components that are easily observable (e.g. when JavaScript libraries change), in the case of server-side logic another challenge is to detect that the change actually occurred.

The second group of challenges is related to the fact that the majority of Web sites and Web applications focus on specific purposes. As a result, Web applications may be very rigid with respect to how data are exposed (querying capabilities, limiting the number of records that can be obtained, dispersing attributes into Web pages, the multitude of different querying combinations of navigation alternatives). Adapting to a Web site's rigidity when needed, but also using its flexibility when applicable, is one of the key challenges for the efficient extraction of data responding to a specific query.

Navigation View

The view that was presented above concerns a single user interaction with a Web application. However, in normal settings we are interested in a sequence of such actions forming a navigation session. Such a perspective on user interactions brings a number of new challenges, related both to user actions modeling and the evolution of client-side and server-side session state.

The major challenge of information extraction from Web sites with complex navigation is understanding their navigation patterns. It requires knowing and generalizing possible user actions that exist in Web sites as well as understanding how these actions change the Web browser's content, client-side information storage and all information on user actions stored on the server (all this information together will be further called *navigation state*). More precisely, it requires understanding of what data and what next actions are available at a specific navigation state, as well as the capability to identify the current navigation state at any moment in time (some Web sites behave in a partially random way, or in a way depending on the activities of other users; thus, even repeating the same navigation patterns can lead to a different navigation state, and consequently the current navigation state is challenging and cannot be based only on navigation history).

The next challenge is goal-oriented navigation planning, i.e. understanding which sequence of actions is necessary and best-suited to answer a specific user query. It consists of three specific challenges:

- choosing one of many alternative paths to given data (if the Web site has multiple navigation patterns that could be used to answer a user query),
- limiting navigation to what is really needed to answer a given query, i.e. avoiding issuing these requests (including a request triggered by some programmatic events) that are not necessary to receive the answer to a specific user query,
- dealing with situations where a user query is not compatible with a Web site's navigation patterns.

Finally, navigation poses also a few technical challenges, such as handling different types of authentication, and handling session Cookies and data (i.e. maintaining the session or removing session data as required at a specific situation).

2.3.3 Real-Life Web Sites Analysis

To complete the analysis of information extraction challenges, we investigated a number of real-life Web sites with respect to their data and navigation organization, technologies used and user interaction models. Our analysis has an auxiliary character and does not aim at assuring representativeness. However, we tried to assure maximal technical and informational diversity of Web sites studied.

As candidate Web sites for analysis, we have selected a number of mainstream Polish and English Web sites in different areas of data-intensive Web sites (all candidate Web sites are listed in Table B.1). Our of candidate Web sites, we have chosen a limited number in two steps:

- First, we eliminated all Web sites that provide an API (or flexible export facility) to access the majority of data they contain (i.e. there are better means of querying contained data programmatically than Web information extraction);
- Then, we selected Web sites in such a way that our data set is as diversified as possible with respect to technologies, topics, languages, design patters and user interaction models.

Table B.2 presents a basic analysis of the used technologies of all candidate Web sites that do not provide API or flexible export facilities. Each of these Web sites was analyzed with respect to a number of key technology aspects:

1. the presence of complex user interface elements (is the interface based on non-standard components using JavaScript, Java or Flash?)
2. the presence of complex interactions with a server (does the client application download data in an asynchronous way via AJAX or similar technologies?)
3. the need of navigation actions to receive complete data (in cases where the user query is known, is it still necessary to navigate through multiple pages of the Web site to collect complete data?)
4. the complex character of the data model (is the model of underlying data hierarchical or graph-based?)
5. the complex data presentation (does the Web site use data presentation other than lists or simple tables?)
6. the presence of different data sets in the same Web site (does the Web site contain multiple, distinct data sets covering different areas and having very different data models or schemas?)

Based on the preliminary technical analysis of candidate Web sites described above, ten Web sites varying significantly with respect to their domain and technologies used were selected for further in-depth analysis of technical challenges for information extraction. These Web sites are:

- (2) **GUS**[18] - the Web site of the Polish Statistical Agency provides a variety of data sets covering areas such as macroeconomics, demography, employment, commerce, life conditions, education, local statistics and the natural environment. Data are available

[18]Web site available at: `http://www.stat.gov.pl/`.

in different forms, including downloadable files (PDF or Excel) and various dynamic querying interfaces using different technologies and querying capabilities.

- (5) **gielda.onet.pl** - one of the key Polish information Web sites on the stock exchange market. It provides stock exchange quotes (both current and historical data), technical analysis tools, profiles of companies' actions and other securities, and news related to stock exchanges and companies' quotes. It combines a few technologies, varying from simple HTML (for news, reports, and basic quotes) to Flash and Java (in the case of tools and historical data).
- (22) **123people.com** - an international, multi-lingual people search engine combining information collected from multiple Surface Web and Deep Web sources with results provided by Google in order to offer images, multimedia, text and data resources related to a specific person. It uses specific data presentation and AJAX, and is characterized by a varying structure of result page depending on the scope of information available for a chosen person.
- (25) **Yahoo! Directory**[19] - a classical example and the prototype of most on-line Web site directories. Being a manually maintained resource, it classifies Web sites into a hierarchy of categories (a tree with varying depth for different categories). While it uses relatively simple technologies (pure HTML), its data model poses specific challenges.
- (29) **pogoda.gazeta.pl** - the weather service of one of Poland's top daily newspapers. It provides relatively rich weather information (multiple measures including probabilities; a period of 7 days is covered) for Poland and other European countries. This Web site combines HTML, AJAX and Flash technologies, thus being an interesting case both in terms of server interactions and data presentation.
- (32) **expedia.com** - one of the top world-wide airfare ticket and hotel booking services. It enables an extensive search of flight connections between chosen airports by a number of criteria (such as dates, time vs cost optimization, preferred class or airlines, and the possibility of booking hotels with the flight). The Expedia Web site relies on both HTML and AJAX technologies and makes important use of session-based technologies.
- (34) **filmweb.pl** - the top Polish movies Web site, providing information on a large data set of films, actors and directors, covering basic facts, classification, reviews, users opinions, rankings and awards, trailers, and screen photos. It provides a recommendation mechanisms, both general (based on the similarity of films), and personalized (based on the recommendation of people with similar movies preferences). The Web site contains a significant quantity of inter-connected information and uses sophisticated technical mechanisms, such as AJAX, with a purposeful obfuscation of transmitted data.
- (36) **Telemagazyn**[20] - one of the top Polish TV magazine and Web sites, providing among other things complete information on the programs of close to 200 TV stations. Apart from hours of emissions of specific programs, it provides their brief description and structured specification (cast, director, duration, gender, production year and country)

[19]Web site available at: `http://dir.yahoo.com/`.
[20]Web site available at: `http://www.telemagazyn.pl/`.

as well as navigation between the subsequent episodes of repeating programs. While it uses mostly HTML with the rare addition of AJAX elements (that do not concern key functionalities of the site), it has a specific pivot-table-alike presentation of information.

- (38) **www.eventim.pl** - one of the top Polish and international on-line ticket reservation services, covering all kinds of cultural and sport events. It provides a calendar (and simple search engine) of events, and ticket reservation facilities. The Web site is relatively simple as a source of event information; however, it uses more complex technologies (Java with asynchronous data download) for seat selection and reservation.
- (39) **Facebook** - number one in today's on-line social networking. It provides basic social networking tools (managing the graph of people, pages, interests and groups, and providing a few communication mechanisms), as well as extensive number of in-built and third-party applications, plugins and widgets. Much information stored in Facebook is publicly available to anyone or to friends / liked pages owners, obeying the liberal privacy setting of many users. However, no relevant API, the technologies used by Facebook (including AJAX and in-built limitations of data that can be seen at a given moment) and complex a data model make access to these data challenging.

Web Sites Analysis

Apart from the preliminary technical analysis described above, for the ten selected Web sites we performed an in-depth qualitative study of the following technical aspects:

- **Event model complexity** - we analyzed types of user-triggered and browser-triggered client-side events by using the FireBug plugin for the FireFox Web browser[21].
- **Server interactions complexity** - we studied what HTTP requests are issued automatically (e.g. in an "onload" JavaScript event) and what HTTP requests are performed after a user action, as well as requests that occur repetitively or stay as open connections to the server. We also analyzed if requests are issued in a synchronous or asynchronous way, and what Cookie they set. The analysis was performed with the Fiddler Web Proxy[22] and the FireBug plugin.
- **User interface complexity** - we studied the types of user interface elements in terms of the technologies they use (pure HTML, JavaScript, AJAX, Flash), the user interaction paradigm (based on clicks, entering text, drag and drop, etc.), and technical ways of how these controls interact with client-side and server-side Web application logic (including the way of attaching client-side events and of interacting with the Web server). The task was performed thanks to the FireBug plugin.
- **Data presentation complexity** - we looked for all complex (in terms of data organization and manner of connecting related information) ways of presenting interconnected data within a single page.
- **Statefulness** - for some tasks in a Web site, first we performed them by using the Web site's navigation, then we saved the URL of the last page accessed and tried to access

[21]Available at http://getfirebug.com/.
[22]Available at http://www.fiddlertool.com/

it directly in other Web browser (with the Cookies, cache and history removed).

The detailed review of ten Web sites allowed us to identify a few technical WIE challenges that were not stated in previous literature and did not clearly result from discussed user interaction model. Examples include th presence of complex form controls using JavaScript, different types of pagination, the complexities of multi-frame documents and AJAX communications, as well as the presence of Web actions initiated by client-side Web application logic. Detailed information about the challenges identified in individual Web sites is present in Table C.1, in column "WS".

2.3.4 WIE Application Scenarios

In the previous chapter we reviewed a few business usage scenarios of Web information extraction. By contrast, in this section we list a few different technical applications of WIE systems, i.e. situations where the information extraction tool is part of a larger software solution, and ideally should be adapted to it.

Ad Hoc Querying

The ad hoc querying of individual Web sites is an application scenario of WIE especially important while working with Deep Web sources. In the basic form of this scenario, the only difference as compared with manual interaction with the Web site is that instead of using HTML form, the user is provided with some other user interface (potentially with some query expansion of visual query building facilities), and that extracted data can be manipulated in a much easier way than in the case of a Web site's HTML presentation.

In more complex variants of this scenario, some extra query capabilities can be offered to the user (e.g. querying attributes not present in query form or overcoming the limitations of the number of required attributes or the maximal number of attributes that can be bound in one query). In such cases, specific challenges of rewriting user queries with respect to source query capabilities and of post-filtering obtained data arise.

A specific type of ad hoc queries are entity-centric queries, i.e. queries about individual objects (e.g. a specific person or book), that often require the application of query rewriting, focused crawling and some type of Web content indexing to be performed in an efficient way.

In the case of large Web sites, the execution of ad hoc queries can be time-consuming. In such cases, the specific challenges of providing users with partial results before complete data are downloaded, as well as of choosing the best Web site traversal strategy become apparent.

Integrated Data Querying

Integrated data querying consists in providing users with one query interface (or query language) that (s)he can use to query multiple data sources. While user interaction with integrated data querying is similar to ad hoc querying, it brings multiple specific research challenges.

Firstly, it requires that a generalized user interface is constructed for multiple data-intensive Web sites. While it is simple in the case of keyword-based search interfaces (when integrated querying is often called a meta-search), it may be very challenging in the case of complex Web query interfaces.

Secondly, if data sources are not explicitly specified by users, an algorithm capable of choosing the most relevant ones is necessary. It is challenging because for many sources no explicit machine-processable description of the Web site's content is provided.

Thirdly, the user query typically needs to be rewritten for Web query interfaces with limited query capabilities. While this task is similar to ad hoc querying, in the case of integrated querying it is much more probable that the user query is significantly incompatible with source querying capabilities.

Two additional challenges are related to mapping schemas of data extracted from different Web sources and to data matching, i.e. connecting records from different sources related to the same objects in order to enable the removal of duplicates and the building of more complete data records. Moreover, in some usage scenarios (such as meta-search) ordering the obtained data may also be challenging.

Web Monitoring

Web monitoring consists in the repetitive accessing of Web sources in order to detect changes to the data they contain. While in some simple scenarios users may want to monitor specific URL addresses, in most situations a user's interest is in monitoring the results of specific general or entity-centric queries over single or multiple Web sites. Good implementations of Web monitoring should be able to detect new, modified and deleted data. Web monitoring can just repetitively gather data and store it for future analysis; however, in many applications the definition of rules, triggers or alerts on monitored data is desired.

From a technical perspective, Web monitoring brings in two key challenges. The first is related to estimating data change frequencies in order to limit each Web site's load and to optimally use limited transfer and processing resources. The second is related to dealing with data changing location (in terms of URLs) within Web sites.

Web Data Materialization

Web data materialization refers to a situation where complete data (or all data corresponding to a relatively large query) contained in single or multiple data-intensive Web sites is downloaded and stored in a database or a data warehouse to enable quick and unrestricted access to data.

The key challenge of Web data materialization in the case of complex data-intensive Web sites consists in building a set of queries or a navigation strategy capable of obtaining all data with a relatively low overhead (such as downloading duplicate pages or posing queries with overlapping results).

Web materialization can be performed once or can be repeated continuously to maintain up-to-date data. In the second case, the challenge of calculating revisit frequency becomes applicable also to Web data materialization.

Web Information Restructuring

While all data querying, monitoring and materialization tasks focus on retrieving data corresponding to the information need of users or other information systems, Web information restructuring covers also the problem of transforming obtained data into the most convenient form. For example, data extracted from multiple sources may be transformed into an uniform hierarchical XML structure, be presented as a single multi-level pivot table, or be made available to the user in the form of a hypertext object with multi-faced querying and navigation facilities.

In this scenario, some data transformation rules need to be built upon the structure of the extracted data. While this process can be done *post factum*, after the extraction was done, integrating it into the data extraction process may increase the performance of this task and/or enable the use of partial data before data extraction is finished.

2.3.5 Identified Challenges

Based on an analysis of challenges identified in the literature, a review of the complexities of a typical user interaction with a Web site, a study of a number of real life Web sites and a discussion of typical usage scenarios of Web information extraction, we propose an extensive hierarchy of Web information extraction challenges. This proposed hierarchy contains 336 challenges, grouped into 9 first-level groups, 25 second-level groups and 253 individual topics.

The main groups of the proposed hierarchy are:

- **Group I (Different data models)** covers all complexities of the data model of the underlying data, including the complex structures of individual objects and the complex relations between multiple objects.
- **Group II (Server interaction challenges)** covers all challenges related to communication with Web servers such as HTTP request construction, request issuing and session management.
- **Group III (Server-side Web application logic challenges)** is related to the behavior of server-side Web application logic, including different methods of content generation, server-side Web application logic unpredictability and challenges related to statefulness.
- **Group IV (Client-side Web application logic challenges)** groups challenges related to complex client-side Web application logic, including client-side statefulness.
- **Group V (Extracting information from different types of content)** covers challenges related to the diversity of Web content formats that information extraction needs to deal with.
- **Group VI (Data extraction)** includes all challenges related to actual information extraction from documents, Web pages or composite content.
- **Group VII (Navigation planning)** focuses on the complexities of goal-oriented planning of navigation within a Web site.

- **Group VIII (Navigation execution)** focuses on all aspects of putting a navigation plan into action and adapting it to a situation happening in a Web site.
- **Group IX (Usage scenarios specificities)** covers all other challenges that result from the specificities of individual usage scenarios.

Topics are further grouped into hierarchies of varying depth. Each labeled topic corresponds to an area of data extraction that is challenging and has up to three individual challenges:

- (D)ealing with - being able to work in a specific situation, i.e. to perform data extraction even if given situation exists
- Taking (a)dvantage - being able to profit from the specificities of a given situation, i.e. to make extraction more robust, quicker or easier if given situation exists
- (L)earning - being able to automatically detect that a situation happens in a given site so that special methods of dealing with or taking advantage of it can be applied

The integrated list of challenges, together with corresponding positions from the literature review, the stages of the user interaction model and real-life Web sites are gathered in Table C.1 in Appendix C.

In this thesis, the proposed classification will be used for two different applications. Firstly, it will be applied in Chapter 3 as part of a review of existing Web information extraction solutions, to verify their ability to work with the challenges of the contemporary World Wide Web. Secondly, it will be used in Chapter 5 to demonstrate that the solution proposed in Chapter 4 is significantly more general with respect to identified challenges than previous work.

Apart from these applications, the classification of challenges has also a value of its own as a tool for analyzing the data extraction needs for specific applications. As information extraction needs differ significantly between usage scenarios and types of sources, they also vary between different companies. The proposed classification can be used to identify challenges that are important for specific businesses (i.e. perform requirements analysis) and to help choose the right information extraction tools.

2.4 Summary

In this chapter, we introduced the research areas of Web information extraction from data-intensive Web sites and demonstrated their applications in electronic commerce. To make Web information applicable in the described scenarios, semi-structured documents need into be transformed into more structured data objects. In Section 2.1, we briefly described and contrasted different methods of imposing structure and semantics on Web content, following bottom-up and top-down paradigms. Then, we contrasted Web information extraction with information extraction from text and introduced basic terms of Web information extraction, such as “wrappers” and “extraction rules” (Section 2.2). Finally, we described a hierarchy of challenges that need to be handled by Web information extraction solutions, based on research that combined a literature review, a modeling-based approach, an analysis of a few real-life Web sites and the technical application scenarios of Web information extraction (Section 2.3).

Chapter 3

Comparison of Existing Web Information Extraction Systems

Having defined the problems and usage scenarios of Web information extraction in the previous chapter, in this chapter we provide an in-depth analysis of the existing knowledge base [152]. We start by reviewing existing schemes for comparing information extraction systems, and presenting our own approach comparison scheme in Section 3.1. Next, in Section 3.2 we compare over 40 previous information extraction systems with respect to their ability to deal with the challenges identified in the previous chapter. We analyze them by applying our comparison scheme, look into the flow of ideas in the field of Web information extraction systems and compare the performance of previous solutions. We end the chapter by identifying the key challenges that are still rarely addressed by existing information extraction solutions (Section 3.3).

3.1 Comparison Scheme for WIE Systems

Apart from the previously presented list of Web information extraction challenges, we use systems classification as the second key element of our information systems comparison methodology. We start by reviewing previous classification schemes and then propose our own approach, covering essential aspects of information extraction systems output and used information extraction methods.

3.1.1 Existing Classification Schemes

Classifications proposed in [72] covered three approaches to the problem of wrapping the result pages of search engines. In the *manual coding* approach, general purpose high-level programming languages are used. In the *parsing by tag-marks*, approach a SGML parser usage is hard-coded into the wrapper created in a high-level programming language (e.g.

Python or Java). In *specification by rules*, high-level grammars are used for document parsing and data extraction.

In [155] and [65] wrapper creation methods are classified into four groups. Hand-crafted wrappers created using *general programming languages* (such as Perl, C++ or Java), have few requirements, but result in the difficult development and maintenance of wrappers. Hand-crafted wrappers expressed in *specially designed programming languages or tools* are somehow more easy to develop but still require specific IT knowledge. Systems based on *specific heuristics* (e.g. related to tag patterns or the possible values of attributes) may work well for specific sources or domains, but lack scalability. Systems using *wrapper induction* use supervised machine learning (e.g. inductive learning) based on a sample of labeled examples. While the training set needs to be constructed manually or semi-automatically (using a known list of attribute values), no programming knowledge or experience is required.

In [97] three dimensions for classification of Web data extraction tools are proposed. Firstly, the *knowledge engineering approach* (based on the manual construction of grammars and other extraction resources) is contrasted with the *automatic training approach* (with wrappers automatically learned from training data). Secondly, systems are divided into *single-slot* (unable to extract multiple connected attributes), and *multi-slots* (that are able to link together related information). Finally, systems are classified into commercial and non-commercial (this division is also followed in [176]).

In [213] Muslea divided extraction information systems into tools dealing with three types of tasks: *information extraction from free text* (from grammatical, plain text by using semantic and syntactic information), *information extraction from on-line documents* (from a "mixture of grammatical, telegraphic, and/or ungrammatical text" by using syntactic and semantic information, as well as punctuation and HTML delimiters), and *wrapper induction*.

The most complete classification, proposed in [262], divides data extraction approaches into six categories, and characterizes them with respect to a few other dimensions. The six main categories of data extraction solutions are: specific programming *languages for wrapper development*, *HTML-aware tools* that use HTML documents' structures (e.g. the Document Object Model), *NLP-based tools* using syntactic and semantic information, *wrapper induction tools*, *modeling-based tools* based on the restructuring of Web page structures into desired output data structures, and *ontology-based tools* using domain-specific knowledge resources (e.g. objects, relationships and lexical resources) to enable extraction from any documents.

Other dimensions studied in the same work [262] are: the degree of automation of the creation of extraction rules (manual to fully automatic), support for complex objects (support for nested objects and structural variations), types of pages handled (systems applicable to semistructured data, and to semistructured text), ease of use (mainly the availability of a GUI), the availability of XML output, support for data extraction from non-HTML sources and formats, system resilience (the ability to extract data when minor changes occur in Web site structure), and finally, the system's adaptiveness (its ability to automatically adapt extraction rules to changes happening over time).

In [242] wrappers are classified into: *record-level* extracting elements of a single list of homogeneous records, *page-level* extracting elements of multiple kinds of records, and *site-*

level performing joint crawling of a Web site and data extraction.

In [202] information extraction techniques are divided into: *sliding windows models* based on sequences of tokens of a fixed length, *finite state machines* models and *tree-based models* working with hierarchical data structures. Approaches to information extraction are also split into methods focused on extraction from individual Web sites, and more challenging methods focused on extracting data from multiple sites.

In [180] supervised learning systems are classified into *Finite-state* approaches, with automata symbols corresponding to tokens, HTML tags or both, and *relational learning* approaches using logic programming languages and an abstract representation of a document as a sequence of tokens with their feature attributes (e.g. such as 'uppercase' or 'isHtmlTag').

In [107] information extraction systems are classified with respect to their representation of documents into solution treating *Web pages as a tree*, e.g. based on DOM, and treating *Web pages as data stream*, i.e. sequences of characters or tokens. Systems are also divided with respect to the level of automation of wrapper creation into manual, semi-automatic (i.e. based on user interaction with user interface) and automatic (using supervised machine learning techniques). Finally, a distinction is drawn between declarative wrappers (i.e. such that "there is a clean separation of extraction rules from the computational behavior of the wrapping engine") and non-declarative wrapper (where extraction rules are mixed with statements in a high-level programming language).

In [67] a classification based on system requirements is proposed, dividing systems into those that need programmers, and those that need annotation examples and annotation-free systems (that perform fully automated extraction, as also advocated in [189]).

In [50] a relatively vague classification of Web data management approaches into five groups is proposed. *Web query systems* are systems that give efficient, query-based access to data contained in individual Web sources. *Web information integration systems* focus on giving integrated access to multiple Web-based data sources. Systems focused on *Web data restructuring* are "capable not only of manipulating the structure of documents, but also of providing a mechanism for generating arbitrarily linked sets of documents". Systems that handle *semistructured data* focus on the "querying of data whose structure is unknown or irregular". Finally, *XML query languages* are designed specifically to allow the querying of XML documents.

In [66] two main aspects of Web information extraction systems are used for classification. The dimension of used techniques considers methods used at different stages of information extraction. It includes approaches to tokenization (*word-level* tokenization is the traditional approach, while the *tag-level* approach considers any tag and any text between tags as a token), extraction rules learning (different techniques include *top-down* and *bottom-up* approaches, *pattern mining* and *logic programming*), extraction rules type (rules based on regular *grammars*, *logic* statements or *path expressions*) and used features (these include *syntactic* or *semantic* constraints and *delimiter-based* constraints such as HTML-tags or literal words).

Automation degree is a usage-centric dimension considering the user expertise needed for labeling training examples or constructing extraction rules, the applicability of method to multiple domains, limitations concerning the size or type of input, support for the automated collection of documents for training and extraction, and relational/XML/API output availability (to enable integration with other applications).

In [175] data extraction methods are organized into solutions *based on unique identifiers*, methods *based on DOM tree traversal*, and systems *based on containing text* that focus on documents' text.

3.1.2 Classification Used in this Thesis

In this section we present a classification and comparison scheme that generalizes the work described in the previous sections and extends it with aspects related to recent developments of the World Wide Web. The proposed scheme is used to compare Web information extraction solutions described in the following part of this thesis.

All methods reviewed in this chapter are classified with respect to the criteria defined in the following sections.

Extraction Output Characteristics

Extraction output characteristics cover the characteristics of output schema and used output data format.

Output schema characteristics define if individual attributes are *named* and *typed*, as well as which types of the following complex data structures can be extracted into output schema:

- *lists* – multiple values for a single attribute,
- *records* – multiple structures having the same structure (the same attributes),
- *nested (hierarchical) structures* – records that embed other lists or records,
- *graphs* – non-hierarchical graphical data (e.g. a lattice),
- *content blocks* – values corresponding to unstructured or semi-structured (e.g. HTML) fragments of original documents,
- *documents* – complete documents as extracted values (information retrieval approach).

Output format is related to how output data structure is encoded and presented to the user or other software components. Examples of output format include XML, JSON or RDF; however, this dimension is open-domain.

Extraction Rules Characteristics

Extraction rules characteristics are related to whether a given system uses extraction rules (*rule-based systems*) or not (*ruleless systems*), if extraction rules use *linguistic resources* or *previous data*, as well as how extraction rules are expressed.

We split ways how extraction rules are expressed into *declarative*, *procedural* and mixed. We also classify wrappers into working with *fully specified rules* (rules that are complete and unambiguous) and *partially specified rules* (rules that roughly identify some part of the content on which the actual heuristic extraction algorithm is executed), as well as into *content-based rules* (rules relying on the internal characteristics of data structures being extracted, such as the format of individual attributes and the coherence of data records) and *context-based* *rules* (rules based on information surrounding extracted data, such as lexical, punctuation and tag-based delimiters, DOM tree path or the visual location of data).

Finally, we check if extraction rules in a given system use one of the typical data extraction formalisms: *grammars*, *finite automata*, *regular expressions*, *logic*-based formalisms or *XPath* language.

Extraction Rules Management

The last area of our classification is related to the practical aspects of extraction rules management, such as their creation, maintenance and their ease of adaptation for other domains.

Firstly, we divide systems with respect to wrapper (extraction rules) creation methods into five groups. The first consists of systems with *manual* wrapper creation in some programming language or data extraction formalism. The second group covers systems providing some *GUI* supporting the manual creation of wrappers (including interactive and by-example rule creation approaches). The third one relies on *supervised learning*, i.e. the automated creation of wrappers based on annotated training pages. Systems using *unsupervised learning*, i.e. enabling automated creation with no need of training Web pages, form the fourth group. The final group relies on *automated extraction*, and includes systems that perform extraction from unseen data pages without the need of creating wrappers.

Secondly, we categorize systems into those requiring technical *experts*, active *user* participation, *labeled example pages*, *unlabeled example pages* and *domain data models*.

The third aspect we use to compare systems (called rules management) is related to the capability of *wrapper adaptation* to Web page changes.

Finally, we compare the *effort* that is needed to make each extraction system work in a new domain.

3.2 Comparison of Existing Solutions

The previous section described classical and state-of-the-art Web information extraction systems. In this section, we provide an in-depth analysis of existing solutions. We start by comparing the described systems with respect to their handling of challenges identified in Section 2.3 as well as to their classification with our scheme described in Section 3.1.2. Secondly, we review the performance of these systems reported in previous work. Next, we describe our genetic analysis of existing solutions that covers ideas and authors flow between systems. Finally, we provide a constructive critique of existing work and identify the challenges in this field that still need to be addressed.

3.2.1 Feature-based Analysis

Capability to Deal with Challenges

In the previous chapter we proposed a complete list of challenges of Web information extraction from data-intensive Web sites. We used this list to analyze to what extent existing Web information extraction systems address these challenges.

Our analysis covered all except two systems described in the previous section. We excluded only the WHIRL and Turbo systems that implement very interesting ideas that have an impact on information extraction and integration, but are by no means data extraction or Web navigation systems.

For each atomic challenge, we assigned each system one of three values: challenge not addressed, challenge partially addressed, and significant effort towards addressing the challenge. The distinction between partial solution and significant effort is the following. A system partially addresses a challenge if its authors state that the challenge exists and provide a solution that deals with it in some cases, or if partial support for challenge resolution can be induced from the description of other system's features, even if it is not explicitly stated. Significant effort to address the challenge is the situation where the challenge is properly studied and a solution that works in most cases (but not necessarily in all) is provided.

It is to be noted that we based our analysis solely on available research and technical publications, and assigned a partial or significant support score only if the solution is explicitly described. All cases when no details on some aspect are provided (which is the case of the most technical challenges for a few systems) were treated as if system did not address the specific challenge.

The complete comparison of analyzed systems with respect to their ability to deal with identified challenges is contained in Appendix C. As the complete data are very detailed and hard to analyze, in Table 3.1 we provide an estimate of the percent of challenges handled by each system in nine key areas of our challenges lists: different data models (DM), server interaction (SI), server-side business logic (SS), client-side business logic (CS), different content types (CT), data extraction (DE), navigation planning (NP), navigation plan execution (NE) and usage scenarios specificities (US), as well as an estimate for the total percentage of addressed challenges (Tot). For each area we provide also an average score of all systems.

The percentage is calculated by summing up all addressed challenges in a group (adding half a point for partially addressed challenges) and dividing them by the total number of challenges. While such an approach does not differentiate between more and less important challenges and adopts a very approximated approach to partially addressed challenges, it provides a simple and intuitive way of comparing systems and of identifying groups of challenges that a specific system focuses on.

To increase the readability of the table, we filtered out all percentages lower than 5% (however, they were still used in the calculation of average values). For each system (each row), all scores that account for at least 90% of its score in the group where it performs best are printed in bold. For example, TSIMMIS performs best in the DM group (29%), but its result in the US group (17%) is also displayed in bold. Similarly, for each group of challenges (each column), all scores that are at least 90% of the group's best result have grey background. For example, in the case of the DE group, the highest result is 25%, but all 23% and 24% results also have grey background.

The first obvious observation when looking at the data gathered in Table 3.1, is that the presented scores are very low. Relatively low results in the "total" columns would not be striking, as many systems focus only on specific areas of WIE problems; however, the fact

System	Tot	DM	SI	SS	CS	CT	DE	NP	NE	US
TSIMMIS	7%	**19%**	5%			10%	10%			**17%**
W3QS	9%	**34%**	8%		11%	10%	7%		7%	
IM	4%			**20%**						**21%**
WebLog	3%				**7%**					
WebSQL	1%									
AK	4%	**9%**					7%			
Webfoot	4%	9%				8%	**12%**			
ShopBot	7%		6%		**10%**		**9%**	6%	7%	8%
WIEN	6%	**17%**				8%	**17%**			
Araneus	10%	9%	8%		**19%**	10%	12%	9%	7%	12%
WebOQL	11%	**39%**	6%		9%	6%	11%	10%	12%	
FLORID	5%	**17%**	5%					6%	5%	
Jedi	5%	**11%**				8%	**12%**			
NoDoSE	7%	11%				10%	**22%**			
Stalker	8%	**23%**				6%	**23%**			
SoftMealy	8%	17%				10%	**24%**			
Junglee	3%		**13%**			6%				**12%**
W4F	7%	**21%**	5%				17%			
DEByE	12%	21%	5%		17%	6%	**25%**	7%		
XWrap	5%		8%			6%	**12%**			
WebVCR	7%				**26%**		11%	5%	9%	
Lixto	9%	**27%**			23%	8%	7%	12%		
HiWE	6%		8%		**14%**	8%	7%		5%	**15%**
Omini	3%						**9%**			
IEPAD	5%	11%				6%	**14%**			
EXALG	7%	**39%**					13%			
WL2	4%						**18%**			
RoadRunner	8%	**30%**					18%			
Denodo	15%		29%	13%	**44%**	6%	9%	13%	5%	12%
DeLa	8%	**29%**			6%	6%	15%			
Cameleon	9%		**22%**		17%	8%	13%	8%		
VIPS	3%						**12%**			
DEPTA	6%	6%					**23%**			
IDE	4%	9%				8%	**11%**			
ViNTs	7%	19%					**23%**			
Thresher	6%	11%				**17%**	14%			
DWDI	9%		5%		**20%**		5%	**20%**	**19%**	
SSF	3%					**8%**	**9%**			
Senellart	9%	11%	5%		**16%**		9%	13%	10%	8%
DEQUE	12%	6%	5%	15%	**24%**		11%	14%	7%	19%
Average	**0%**	**12%**	**4%**	**1%**	**7%**	**6%**	**12%**	**4%**	**3%**	**4%**

Table 3.1: Previous Solutions – Overview of Addressed Challenges

that the maximum group-level score of individual system is below 50% may be surprising.

This is due to several factors. First of all, some parts of challenges are related to relatively new phenomena such as AJAX or social networks, and naturally could not have been addressed by early WIE systems. Secondly, some of the challenges are very specific and concern a relatively low number of Web sites. Many of them were not noticed by the research community, or even if noticed, they were considered unimportant. Thirdly, as we stated previously, we based our score on information explicitly given in research papers. Thus, our measurement has a negative bias, due to the fact that some details (mostly technical) are skipped in research publications. Finally, some of the challenges are not *sine qua non* requirements for systems to work, but rather considerations for systems' augmented performance of flexibility. For example, the ability to gather statistics, build indexes or limit overhead while downloading Web content are important from an efficiency perspective, but are not considered as crucial system's features.

When we look into the structure of the discussed table, we notice that there are only a few "general systems" that deal with almost all areas of challenges. These systems (Araneus, WebOQL, Denodo, Senellart et al., DEQUE) tend also to have a relatively high total score. On the other hand, systems that focus solely on one or two areas are also a rarity, apart from pure extraction systems (Ashish/Knoblock, Omini, WL^2, RoadRunner, VIPS, DEPTA and ViNTs) that focus their efforts on data model and data extraction challenges.

These two groups of challenges are indeed the best addressed ones. Dealing with different types of content and providing server interaction facilities is also touched on by a significant number of systems, but with a relatively low percentage of addressed challenges. By contrast, client-side business logic challenges are addressed by fewer systems, but with a higher average implementation level. Results in groups related to navigation show that not only navigation are capabilities inherent to a few systems, but also that they deal only with a small part of navigation-related challenges. Finally, in the case of server-side business logic, only three systems (Information Manifold, Denodo and DEQUE) address at least 5% of challenges, making it clearly an area where a lot of research is needed.

Output Characteristics

Output characteristics are the first dimension of our classification scheme proposed in Section 3.1.2. We applied this comparison scheme to all solutions analyzed in the previous section. The results of this analysis are gathered in Table 3.2. Data marked for each system are: usage of named attributes (column N), usage of typed attributes (column T), support for the extraction of records (column R), lists of values (column L), hierarchically nested data (column H), graphs (column G), content blocks (column B) or complete documents (column D), as well as output format for the extracted data.

The results of our analysis demonstrate that many information extraction systems adopt a very unorthodox approach to the schema of the extracted information. The majority of systems return data without any type assigned (i.e. as strings that need to be interpreted and transformed by external business logic) with a few systems making type assignment optional, and a few of them are even incapable of assigning attribute names.

Most of the analyzed systems are capable of extracting data organized into records (or multislot data, by using naming from early WIE systems). Many systems support the extraction of multi-valued attributes (i.e. lists), and about half of them handle hierarchical structures based on nesting. Only a few systems return unstructured content blocks or complete HTML documents. WebOQL is the only system that explicitly provides graph output (while Thresher's output format is RDF, which is graphical by nature, Thresher uses it only to store hierarchical data); other systems can extract data from Web sites using the graph model (e.g. social networking services), but need to store information in an hierarchical form.

Many authors provide no details on the format of output data. Out of explicitly listed formats, XML seems the most natural choice, but solutions providing the direct integration of output into other applications (as objects or through ODBC/JDBC drivers) are not uncommon.

System	N	T	R	L	H	G	B	D	Output format
TSIMMIS	●	○	●	●	●	○	○	○	OEM objects
W3QS	●	○	●	●	○	○	○	○	table
IM	●	○	●	●	○	○	○	○	?
WebLog	○	○	○	●	○	○	○	○	?
WebSQL	-	-	○	○	○	○	○	●	?
AK	○	○	●	●	●	○	○	○	?
Webfoot	○	○	●	●	○	○	○	○	?
ShopBot	●	○	●	○	○	○	○	○	?
WIEN	●	○	●	●	●	○	○	○	?
Araneus	●	○	●	○	○	○	○	○	ADM
WebOQL	●	○	●	●	●	●	○	○	HTML
FLORID	●	○	●	●	●	○	○	○	?
Jedi	●	●	●	●	○	○	○	○	XML/Java obj.
NoDoSE	●	○	●	●	●	○	○	○	text, OEM
Stalker	○	○	●	●	●	○	○	○	?
SoftMealy	○	○	●	●	○	○	○	○	?
Junglee	●	●	●	○	○	○	○	○	ODBC/JDBC
W4F	●	◐	●	●	●	○	○	○	XML/Java obj.
DEByE	●	○	●	●	●	○	○	○	XML
XWrap	●	◐	●	●	○	○	○	○	XML
WebVCR	-	-	○	○	○	○	○	○	-
Lixto	●	○	●	●	●	○	○	○	XML
HiWE	-	-	○	○	○	○	●	○	HTML
Omini	-	-	○	○	○	○	●	○	HTML
IEPAD	○	○	●	○	○	○	○	○	?
EXALG	○	○	●	●	○	○	○	○	?
WL2	○	○	○	●	○	○	○	○	?
RoadRunner	○	○	●	●	●	○	○	○	?
Denodo	-	-	-	-	-	-	-	●	?
DeLa	●	○	●	●	●	○	○	○	XML
Cameleon	●	●	●	○	○	○	○	○	XML, SQL, ODBC
VIPS	-	-	○	○	○	○	●	○	HTML
DEPTA	○	○	●	○	○	○	○	○	?
IDE	○	○	●	○	○	○	○	○	?
ViNTs	○	○	●	○	○	○	○	○	?
Thresher	●	●	●	○	○	○	○	○	RDF
DWDI	●	○	●	○	○	○	○	○	XML
SSF	○	○	○	○	○	○	●	○	HTML
Senellart	●	●	●	○	○	○	○	○	?
DEQUE	●	○	●	○	○	○	○	○	?

Table 3.2: Previous Solutions – Output Characteristics

Extraction Rules Characteristics

The second dimension of our scheme concerns the characteristics of extraction rules. Information on all systems can be found in Table 3.3, collecting the following data: support for rule-based (column +) or ruleless (column —) extraction; usage of linguistic resources (column l) or previous data (column d); declarative (D), procedural (P) or mixed (M) extraction formalism (column f); ability to work with fully (column F) or partially (column P) specified wrappers; ability to perform context-based (column x) and content-based (column t) extraction; as well as reliance on grammars (column G), finite automata (column A), regular expressions (column R), logic-based formalisms (column L) or XPath language (column X).

Analysis of the table provides the following conclusions. Firstly, while there are a few Web information extraction systems able to perform extraction without wrapper training (ruleless systems), in the majority of cases the separation of the wrapper creation (rules definition) phase and wrapper usage phase is maintained.

Secondly, there are very few WIE systems that use some lexical resources and are capable of using previously extracted data. This demonstrates that there are probably still experiences from fields of information extraction from text (such as the optional or required use of lexical resources) and databases (such as the holistic integration of multiple schemas by using data instances) that can be used in future Web information extraction systems.

The third observation is that declarative or partially declarative extraction rules dominate over procedural ones. In general, even if declarative approaches typically introduce new extraction languages unknown to most programmers, their usage is a good practice as declarative solutions provide better separation of physical extraction from its logical specification, and enable concise specification of otherwise long extraction rules.

Following the distinction between the rule creation and rule usage phase, most of the analyzed systems assume that extraction rules should be completely specified during the first of these phases, and should unambiguously identify information during usage phase. Only a few systems allow incomplete rules (we may think of them as of hints related to data location in Web pages), and assume that during the execution phase some extra extraction rules tuning may be needed. While this direction is unpopular, it could provide better robustness when changes in source occur. Thus, it is one of the direction to explore for future WIE systems.

The similar domination of one approach can be observed in the case of the distinction between context-based rules and content-based rules. Today, most systems rely only on context (i.e. where the information is located?) without mining rules related to content (i.e. what are the internal characteristics of information to be extracted?). While the complete ignoring of context information seems to be unrealistic, incorporating content information (e.g. based on lexical resources, unique format of fields, words distribution statistics, fields co-occurrences in previous extracted data) may be an interesting research direction. Content-based extraction may be also a promising approach to combining Web information extraction with information integration from multiple Web sites.

The final observation is related to information extraction and modeling approaches applied. While a few relatively early WIE systems used grammars or automata, today mostly regular expressions, XPath and some logic-based approaches are used.

System	+	—	l	d	f	F	P	x	t	G	A	R	L	X
TSIMMIS	●	○	○	○	P	●	○	●	○	○	○	○	○	○
W3QS	●	○	○	●	D	●	○	●	●	○	○	○	○	○
IM	●	○	○	○	D	●	○	?	?	?	?	?	?	?
WebLog	●	○	○	○	D	●	○	●	○	○	○	○	○	○
WebSQL	●	○	○	○	D	●	○	○	●	○	○	○	○	○
AK	●	○	○	○	D	●	○	●	○	●	○	○	○	○
Webfoot	●	○	●[1]	○	D	●	○	●	○	●	○	○	○	○
ShopBot	●	○	?	○	D	●	○	●	○	○	○	○	○	○
WIEN	●	○	○	○	D	●	○	●	○	○	○	○	○	○
Araneus	●	○	○	○	M	●	○	●	○	○	○	○	○	○
WebOQL	●	○	○	○	D	●	○	●	○	○	○	○	○	○
FLORID	●	○	○	○	D	●	○	●	○	○	○	○	○	○
Jedi	●	○	○	○	M	●	○	○	●	○	○	○	○	○
NoDoSE	●	○	○	○	D	●	○	●	○	○	○	○	○	○
Stalker	●	○	○	○	D	●	○	●	○	○	●	○	○	○
SoftMealy	●	○	○	○	D	●	○	●	●	○	●	○	○	○
Junglee	●	○	?	?	?	●	○	?	?	?	?	?	?	?
W4F	●	○	○	○	D	●	●	●	●	○	○	●	○	○
DEByE	●	○	○	○	D	●	●	●	○	○	○	○	○	○
XWrap	●	○	○	○	D	●	○	●	○	○	○	○	○	○
WebVCR	●	○	○	○	D	○	●	●	●	○	○	○	○	○
Lixto	●	○	●[2]	○	D	●	○	●	●	○	○	○	●	●
HiWE	○	●	●[3]	●	D	-	-	●	○	○	○	○	○	○
Omini	○	●	○	○	-	-	-	●	○	-	-	-	-	-
IEPAD	●	●	○	○	D	●	○	●	○	○	○	○	○	○
EXALG	●	○	○	○	D	●	○	●	○	○	○	○	○	○
WL2	●	○	○	○	D	●	○	●	○	○	○	●	○	●
RoadRunner	●	○	○	○	D	●	○	●	○	○	○	●	○	○
Denodo	●	○	○	○	M	●	○	●	○	○	○	○	○	○
DeLa	●	○	●[4]	○	D	●	○	●	○	○	○	●	○	○
Cameleon	●	○	○	○	D	●	○	●	○	○	○	●	○	○
VIPS	○	●	○	○	-	-	-	●	●	-	-	-	-	-
DEPTA	○	●	○	○	-	-	-	●	○	-	-	-	-	-
IDE	●	○	○	○	D	○	●	●	○	○	○	○	○	○
ViNTs	●	○	○	○	D	●	○	●	○	○	○	○	○	○
Thresher	●	○	○	○	D	●	○	●	●	○	○	○	○	○
DWDI	●	○	○	○	D	●	○	●	○	○	○	○	○	●
SSF	○	●	○	○	-	-	-	○	●	-	-	-	-	-
Senellart	●	○	●[5]	●	D	○	●	●	○	○	○	○	○	○
DEQUE	●	○	○	○	D	●	○	●	○	○	○	○	○	○

Table 3.3: Previous Solutions – Rules Characteristics

However, it is worth noting that almost a half of the analyzed systems use their own

[1]Lexicon, POS tagger
[2]Some semantic classes
[3]Labels dictionary
[4]Dictionary
[5]Automatically built gazetteer

information extraction approaches. In some cases it seems inadequate, as proposed approaches could be easily transformed into one of the classical ones, for which exist a large body of research both related to learning and using phases. However, some systems' usage of their own extraction formalisms is necessary, as they are based on information not incorporated in classical formalisms (such as visual modeling).

What is also surprising, especially if we compare Web information extraction with neighboring fields, is the low usage of probabilistic frameworks such as Hidden Markov Models, maximum entropy approaches or conditional random fields. Using such frameworks to combine different information extraction approaches (probably expressed, when possible, as regular expressions or XPath queries) seems an interesting research approach.

Extraction Rules Management

The final dimension of our comparison scheme applied to all systems is related to support for extraction rules management. The results are gathered in Table 3.4. Firstly, we analyzed used wrapper construction techniques (column W) with possible values being manual (M), GUI-based/interactive (G), supervised learning (L), unsupervised learning (U) and automated extraction (A). Secondly, we checked if systems require expert (column E), user (column U), labeled pages (column L), unlabeled pages (column P) and a domain model (column D) to create wrappers. Finally, we looked for the existence of long-term adaptation mechanisms (column A), and described the effort to switch to other domain (column S) with possible values of low (L), medium (M) and high (H).

Our first observation is that the approach to the creation of wrappers is probably the most diversified dimension of our comparison scheme. While in general we observe a trend to move from manual or GUI-based wrapper creation to supervised learning, unsupervised learning and automated extraction, in recent years we have seen a number of systems that focus on problems other than wrapper creation (including Denodo, DWDI and Cameleon).

What varies over time are the extraction system requirements for wrapper learning. While there are still contemporary systems that require an IT expert, it is rather a typical trait of early WIE solutions. Similarly, supervised learning has become unpopular recently. The paradigm based on user interaction has equally been popular during the history of this field; however, the most recent systems tend to require much less specific user activities than the early solutions. The most visible tendency in today's WIE systems is towards working with unlabeled Web pages and possibly using (at least to some extent) the existing domain model.

Our next observation is that the topic of wrapper maintenance remains largely unexplored. While multiple systems provide some support for long-term wrapper adaptation, only Denodo, NoDoSE and DEPTA approaches are real continuous adaptation solutions. Other solutions are based on in-built robustness (e.g. WebVCR) but have no mechanism of adaptation if it fails, or apply the idea of re-learning wrappers from scratch (IEPAD, Senellart et al.) in the case of source modification. Thus, long-term adaptation remains one of the open topics for future WIE systems.

System	W	E	U	L	P	D	A	S
TSIMMIS	M	●	○	○	○	○	○	H
W3QS	M	●	○	○	○	○	○	H
IM	M	●	○	○	○	○	○	M
WebLog	M	●	○	○	○	○	○	H
WebSQL	M	●	○	○	○	○	○	H
AK	G	○	●	○	○	○	○	L
Webfoot	L	○	○	●	○	○	○	H
ShopBot	G	○	○	○	●	○	○	M
WIEN	L	○	○	●	○	○	○	L
Araneus	M	●	○	○	○	○	○	H
WebOQL	M	●	○	○	○	○	○	H
FLORID	M	●	○	○	○	○	○	H
Jedi	M	●	○	○	○	○	○	H
NoDoSE	G	○	●	○	○	○	●	H
Stalker	L	○	○	●	○	○	○	M
SoftMealy	L	○	○	●	○	○	○	M
Junglee	M	●	○	○	○	○	?	?
W4F	G	●	○	○	○	○	○	M
DEByE	G	○	●	○	○	○	○	L
XWrap	G	○	●	○	○	○	○	M
WebVCR	G	○	●	○	○	○	●[6]	L
Lixto	G	○	●	○	○	○	○	L
HiWE	M	●	○	○	○	○	○	M
Omini	A	○	○	○	○	○	○	L
IEPAD	A	○	○	○	○	○	●[7]	L
EXALG	U	○	○	○	●	●	○	L
WL2	G	○	●	○	○	○	○	L
RoadRunner	U	○	○	○	●	●	○	L
Denodo	M	●	○	○	○	○	●[8]	H
DeLa	U	○	○	○	●	○	○	L
Cameleon	G	●	○	○	○	○	○	H
VIPS	A	○	○	○	○	○	○	L
DEPTA	A	○	○	○	●	●	●[9]	L
IDE	G	○	●	○	●	○	○	M
ViNTs	A	○	○	○	○	○	●[10]	L
Thresher	G	○	●	○	○	○	●[11]	M
DWDI	M	●	○	○	○	○	○	H
SSF	A	○	○	○	○	○	●[12]	L
Senellart	A	○	○	○	○	●	●[13]	H
DEQUE	A	○	○	●	○	○	○	M

Table 3.4: Previous Solutions – Rules Management

[6]some in-build robustness
[7]re-learning
[8]automated, based on previous instances
[9]automatic adaptation
[10]re-learning
[11]adding objects to wrappers
[12]re-learning
[13]re-learning

3.2.2 Genetic Analysis

In this section we look at the development of the Web information extraction field as a whole by analyzing the flow of ideas between individual systems. To illustrate the flow of ideas in the field of Web information extraction we analyzed for each system the systems that the authors compare their work to. We assumed that ideas from previous systems were naturally adopted in the next ones, and that the disadvantages of existing solutions defined challenges for new research. Thus, we treated each explicit comparison to other systems as an instance of idea flow.

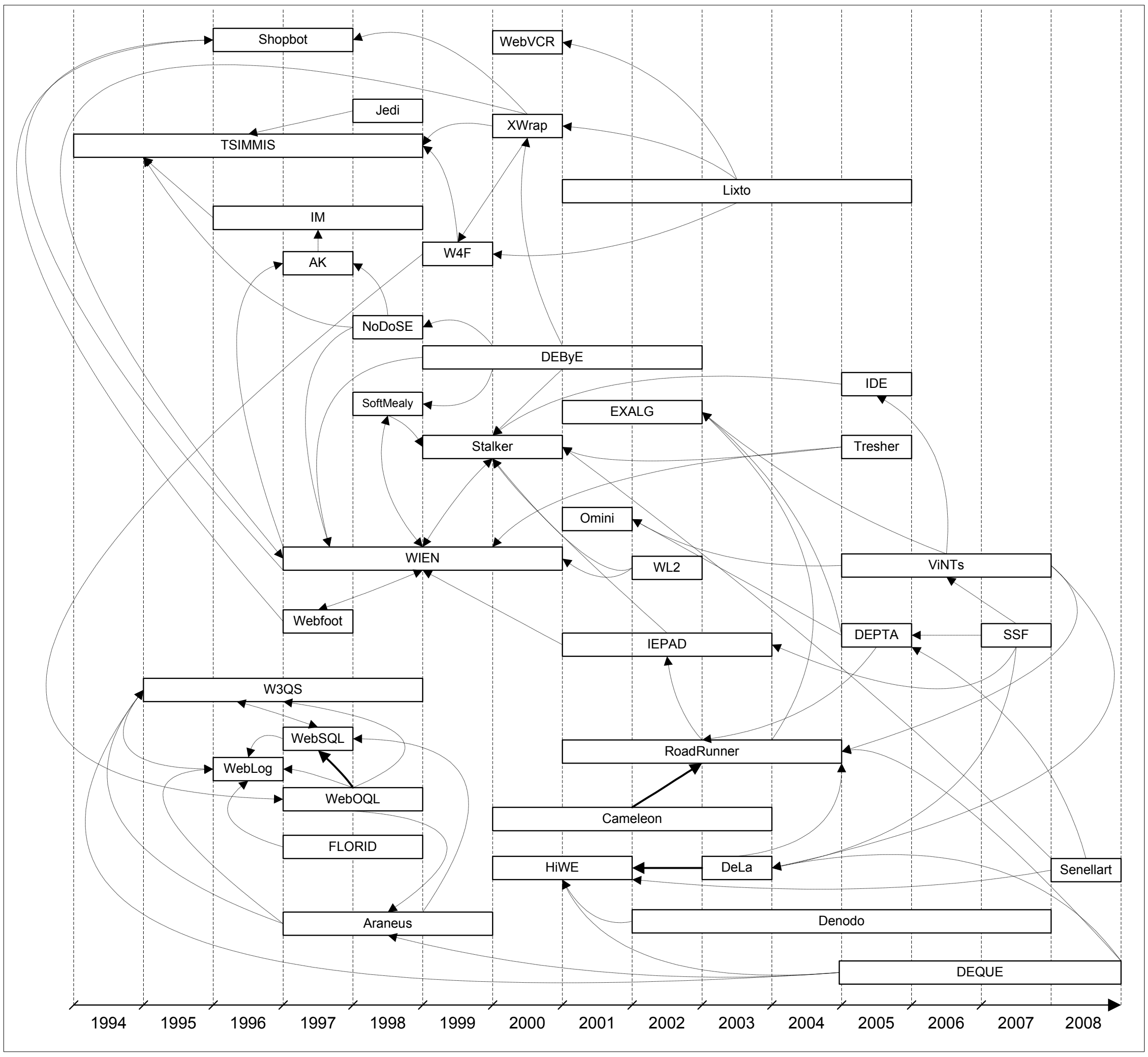


Figure 3.1: Idea Flow Between Existing Systems

The between-systems comparisons are illustrated in Figure 3.1 by arrows drawn from the described system to the systems the authors compare themselves to. In some cases, the arrows are bidirectional, meaning that systems compare themselves to each other. Moreover, we introduce also larger solid (not dotted) arrows corresponding to situations when one system is built upon another (for example WebOQL explicitly reuses some parts of WebSQL).

It is to be noted that our approach differs from purely bibliographic graph-based analyses. If used in flow study, all citations of the resulting graph would be much denser, yet less meaningful.

The first conclusion that can be drawn from the graph is related to its well-connected character. Apart from a separate cluster of Web query languages and related logic-based approaches (gathered in the left bottom corner of the graph), which are mostly disconnected, we are unable to distinguish any autonomous "schools of thought". Thus, the development of the field can be seen as a continuous "memetic crossover" between different existing solutions, characterized by evolutionary improvement and the combination of previous ideas rather than revolutionary changes.

At the same time, the graph helps us to identify the systems that had the biggest impact on later research (i.e. they have the highest number of outgoing arrows). They include WIEN, Stalker, TSIMMIS, RoadRunner, HiWE and WebLog.

As we observed in the previous chapter, information extraction challenges are numerous and very diverse. Thus, while developed systems reused many ideas of previous ones, a growing specialization can be observed over time. As a result, most recent systems (with the exception of Sellenart et al.'s approach) focused on specific information extraction aspects, while the most general, multi-purpose systems (TSIMMIS, Araneus, W4F and Lixto) were developed in the 1990s and early 2000s.

3.2.3 Comparison of Performance

The comparison of the performance of different historical WIE systems is a difficult task for a few reasons. Firstly, for some WIE systems (including WebLog, WebSQL, Araneus, FLORID, NoDoSE, Junglee, W4F, WebVCR and Cameleon) no evaluation was ever described, and for a few (including TSIMMIS and W3QS) only a qualitative evaluation was provided. Secondly, different systems adopt different approaches to both information extraction and performance measurement, thus the results are not related to the same concepts. Thirdly, different systems were evaluated on different example pages, and the only shared data set for the Web information extraction task (RISE-Repository[14]) contains partially unlabeled pages that were gathered over ten years ago and do not reflect the challenges of today's Web. Finally, the implementation of almost none of the works described in this chapter are publicly available and the description of most systems is not detailed enough to enable their re-implementation. As a result, even if we constructed a data set corresponding to all the challenges identified in this thesis, it would not be possible to check the performance of different systems on it.

Due to all these reasons, in this section we focus more on presenting different approaches and aspects of measuring the performance of WIE system rather than on numbers.

[14]Exemplary documents used to be available http://www.isi.edu/integration/RISE/index.html. However, the page was last updated in 2004 and most of links to resources are now broken.

The most popular measures are based on the intuitive notion of the percent of expected successful items that were extracted. This measure is calculated at different levels: the percent of successfully wrapped sources/pages (Stalker, Lixto), the percent of extracted record blocks (DEPTA) or structured records (SoftMealy, DEByE), the percent of extracted lists (Stalker) or attributes values (EXALG, RoadRunner, DEPTA), the percent of correctly labeled forms (HiWE, DEQUE), the percent of well-identified separators (Omini) and the percent of successful (giving non-empty result) submission of queries to Web forms (DEQUE). In some cases (including Stalker, DEByE, EXALG, RoadRunner), apart from success and failure percentage, the percent of partially successful items is also calculated (with definitions of partial success varying from sources with some successfully extracted record in the case of Stalker, to multiple attributes mistakenly extracted as in the case of RoadRunner).

System	Dataset	Measure	Min	Avg	Max
Webfoot	3 sources	% of correct segments (record blocks)	95.8%	98.0%	100.0%
		% lump error (concatenated attribute values)	0.0%	0.9%	1.7%
		% split error (wrongly split attribute values)	0.0%	1.1%	3.3%
WIEN	30 Web sites	% of sites that can be wrapped	–	70.0%	–
Stalker	WIEN dataset	% of sites that can be wrapped with perfect wrappers	–	66.6%	–
		% of sites that can be wrapped with perfect or imperfect wrappers	–	93.3%	–
SoftMealy	one source with multiple pages	% of extracted records	–	87.0%	–
DEByE	23 sets of examples	% of completely received records	40.0%	87.5%	100.0%
		% of incompletely received records	0.0%	12.4%	60.0%
Lixto	12 Web sites	% of correctly extracted attributes (wrapper trained on one example page)	80.0%	95.8%	100.0%
EXALG	45 Web sites	% of completely extracted records	14.3%	80.0%	100.0%
		% of incompletely extracted records	0.0%	20.0%	85.7%

Table 3.5: Previous Solutions – Performance of Chosen Solutions (Percent-Based Measures)

Percent-based measurements for a few previous systems are collected in Table 3.5. Although the results were obtained for different WIE tasks and different datasets and are not comparable, a conclusion that can be drawn from them is that the performance of pure WIE systems is high and approaches 100% effectiveness.

At the same time, the only performance measurement, that is comparable between several systems and is used as a success measure is the percentage of properly extracted records. In this case, the results (gathered in Table 3.6) are comparable because all experiments used the same three data sources (originally included in the WIEN dataset). However, due to a low number of sources, their limited scope of represented challenges and the very high quantitative results of all tested systems, even this comparable study does not simplify the contrasting of multiple systems' performance.

System	OKRA		BigBook		IAF	
	Ex.	Retrieved	Ex.	Retrieved	Ex.	Retrieved
DEByE	2	100%	1	99%	3	99%
WIEN	46	100%	274	100%	–	–
Stalker	1	97%	8	97%	10	85–100%
SoftMealy	1	100%	6	100%	1	99%
IEPAD	–	100%	–	100%	–	100%
EXALG	–	100%	–	100%	–	14% / 86%[15]

Table 3.6: Recall of Some Systems on Three RISE Sources (Based on [181, 65])

While percent-based measures are simple and intuitive, the most visible problem with them is that they do not account for false positives (i.e. items that are extracted, but should not be). That is why authors of a few other WIE systems have chosen rather to use a well-known pair of measures: precision and recall. Precision is the fraction of all obtained items that are the expected ones, while recall is the fraction of all expected items that were successfully obtained. Thus precision lowers with each false positive (i.e. a value that is extracted and should not be), and recall decreases with each false negative (i.e. a value that should be extracted and is not).

System	Datset	Measure base	Avg Prec	Avg Recall
Webfoot	3 sources * 6 configurations	extracted attributes	91.8%	71.6%
Omini	50 Web sites	well-identified separators	100.0%	96.0%
IEPAD	14 Web sites	extracted structured records	100.0%	100.0%
Denodo	44 Web sites in 2 domains	labels assigned to forms	100%	98%
	175 attributes in 44 Web sites in 2 domains	labels assigned to attributes	98%	95%
	521 form fields in 44 Web sites in 2 domains	labels assigned to form fields	82%	93%
DEPTA	72 test pages	record blocks extraction	99.82%	99.27%
		fields separation	99.68%	98.18%
IDE	24 Web sites	extracted objects	99.9%	99.9%
DWDI	7 Web sites	pages acquired	96.0%	100.0%
		records extracted	83.0%	100.0%

Table 3.7: Previous Solutions – Performance of Chosen Solutions (Precision and Recall)

Similarly, as with the percents discussed above (which are technically recall measures), the precision-recall pairs may be calculated for different tasks based on: the number of accessed documents (DWDI), extracted record blocks (DeLa, ViNTs), extracted structured records, extracted attributes (Webfoot, DeLa, ViNTs), identified separators (Omini), values assigned to attributes (Webfoot, Denodo, Senellart et al.), labels assigned to extracted values (DeLa) or form fields (Denodo, Senellart et al.) and forms properly connected to their domain (Denodo).

[15]Percentages of completely and incompletely extracted records.

Precision and recall measurements for multiple systems are collected in Table 3.7. Similarly, as with the percent-based calculations presented in Table 3.5, the individual recall and precision values are only partially comparable with other results. However, the values reported in the literature support the conclusion that tasks of Web Information Extraction are well handled for relatively simple Web sites that were used in almost all previous experiments.

Another aspect of quantitative evaluation that was often used in the case of WIE systems is the time of wrapper generation and/or execution. Unfortunately, such measures, applied among others by DEByE, XWrap, Information Manifold and Ashish/Knoblock, depend strongly on computer configuration and Internet connection performance and are completely incomparable for different systems. For that reason we do not quote any of the reported results. Comparison of the computational complexity of algorithms (presented by the authors of WebOQL and Jedi), is less precise but more useful. However, we skip also these results, as in most scenarios the time of wrapper learning or execution is just a small part of the time spent on the download of Web pages containing data to be extracted.

An important metric for wrapper learning systems is the minimal number of examples needed for building a high-precision-and-recall wrapper. Such a measure (and related experimental procedure) was used, for example, by WIEN, Stalker, Lixto, WL^2 and Thresher. In the case of WIEN, the notion of the complexity of the training example is also considered.

System	Dataset	Measure	Min	Avg	Max
AK	number of needed manual corrections to extraction rules	14 sources: 7 sources with multiple instance, 7 with single instance	0	2.2	6
WIEN	30 Web sites	number of needed example pages	2.0	2.7	9.0
Stalker	18 perfectly wrapped sources from WIEN dataset	number of needed example pages	1.0	1.5	10.0
Lixto	12 Web sites	number of examples needed to train wrapper with the 100% efficiency	1.0	1.8	3.0

Table 3.8: Previous Solutions – Performance of Chosen Solutions (Other Measures)

A few other specific performance measures used in previous work include the number of manual post-corrections needed to obtain complete and well-structured data (Ashish/Knoblock), the average tree distance of extracted and expected data (SSF), the number of enumerated and generated query plans (Information Manifold), and the utility of the extracted data in search tasks (ShopBot). We gathered a few examples of other measurements reported in the literature in Table 3.8. The main conclusion that can be drawn from these data is that the more recent systems require less training data than less recent ones.

3.3 Challenges to Address

Web information extraction is a mature research field, with a lot of work published from the early 1990s to the first decade of the 21st century. The classical problems of extracting data from text and learning wrappers for basic template-based HTML documents were mostly resolved with only minor contributions happening over time. Modern WIE systems are in general capable of learning wrappers, based on a limited number of positive examples, relatively simple user interaction or the comparison of structures of multiple documents, that are capable of handling complex hierarchical data structures.

However, the rapid development of Web technologies, the growth of the complexity of Web sources and the evolution of on-line user interaction paradigms continue to bring new challenges that remain completely or partially unaddressed. That is why, out of a total of 336 challenges identified in Section 2.3.5, 141 are addressed by none of the analyzed systems.

3.3.1 Information Extraction from Web Documents

Even if state-of-the-art systems are generally capable of learning and using wrappers for semi-structured Web documents, multiple challenges are still to be addressed by future generations of Web information extraction systems.

Firstly, virtually all of today's WIE systems can work only with individual HTML documents, ignoring other popular formats (PDF, MS Office formats, graphical files), composite Web content (using multiple frames to present single logical page out of multiple HTML documents), interactive documents (e.g. using dynamic HTML/AJAX components or embedded Flash) and the possibilities of data extraction both from encoded content (e.g. XML or JSON content in the case of many AJAX-like solutions) and user content (i.e. information interpreted by Web browsers and client-side Web application logic).

Secondly, while different systems use different document models and different features ("sources of evidence") of extracted content, none of them is capable of combining purely textual features (sequences of characters or tokens, punctuation), semantic features (the presence of words or patterns belonging or similar to specific semantic classes), DOM-based features (HTML elements sequences or DOM tree paths) and visual data (graphical characteristics of HTML elements, relative positioning etc.). However, due to the enormous diversity of modern Web sites, for some of them only specific features (or combinations of features) can be used to identify information about a user's interest in a way that is both unambiguous and robust to Web site changes over time.

As a result, some complex Web sites can be wrapped only if multiple features are combined, ideally by the use of an extendible, modular model and software architecture. While such modular approach was used in some early WIE systems, it was almost forgotten in more recent solutions.

Thirdly, there are still data structures (such as hierarchies of unknown depth) that are not handled by state-of-the-art systems. Moreover, the focus of today's systems is on extracting individual data objects rather than complete graphs of Web objects (such as data stored in social networking Web sites).

Another issued is that, as we stated in Section 3.2.1, contemporary systems rely mostly on context-based rules, while in some scenarios combining multi-feature context and content information may be necessary, for example, to discriminate between structurally similar but semantically distinct data objects. Content-based approaches may be also beneficial for fully automated wrapper learning when a domain model exists, as they can significantly reduce learning search space.

One thing that could be beneficial to content-based extraction is the system's ability to use already existing complete or incomplete data sources, such as: domain model (with or without data instances), data items extracted from other Web site(s), or even data extracted previously from the source being wrapped (in the case of re-learning or adapting extraction rules). It fits well into the vision of holistic data extraction (somehow initiated by the Turbo system), in which the existing data sources and the similarity of pages in new sources are combined to learn wrappers for multiple sources one after another.

Apart from the ability to use existing data, such an approach brings in the problem of choosing the right order of wrapper learning (starting with sources that are the simplest to wrap). As existing data sources may have different schemas and vocabularies, another challenge of holistic WIE systems should be the capability to detect ambiguities that need to be resolved by users. Formulating these ambiguities for end users as questions related to domain rather than to underlying technical patterns (regular expressions, DOM structures etc.) is a further challenge for future WIE systems.

Another problem that is widely ignored in the existing research is that once a wrapper is trained, it needs to be maintained. The majority of existing approaches assume that wrapper failures are easy to detect, and treat wrapper re-learning as the most natural way of adapting to changes occurring in Web sites.

We argue that both the continuous self-diagnosis of wrapper systems and their ability to adapt to changes are still open research problems. While some wrapper failures may be very visible (e.g. when we stop to receive data from a source), some cases may be much less evident. After a Web site is modified we may still receive some data, while other data may be missing (for example, because some featured search results were formatted in a different way). On the other hand, the modification of a Web site may be gradual, and only after a few modifications the wrapper may fail. However, if wrapper adaptations followed the minor adaptations of Web sites, it would be often possible to completely avoid the failure. Thus, we believe that more intelligent approaches to wrapper failures (probably based on combining and comparing rules based on different features) are needed, followed by the continuous adaptation of existing wrappers (by applying all features and data sources used during wrapper learning, including previously extracted data instances).

The conceptual challenge that naturally follows continuous wrapper adaptation is related to relaxing the strict separation of the training and usage phases of information extraction, and the associated approach of fully-specified wrappers. If we assume that wrappers should continuously evolve as we gather more data from the Web site (e.g. after we find some rare structural exceptions present in the data model) or as the Web site's structure is modified, they can no longer be considered as fixed and unambiguous extraction rules. As a result, we believe that the key challenge for WIE tasks will be to construct systems capable of using multiple features and information sources (including existing databases, other wrappers and interaction with non-expert users) and of treating each performed data extraction as an occasion to fine-tune or adapt extraction rules to changing Web sites, thus blurring the distinction between wrapper learning and usage.

3.3.2 Extracting Information from Complex Web Sites

So far we have analyzed the open problems related to extracting information from Web documents. However, as we discussed in Section 1.2, nowadays the Web contains much complex data-intensive Web sites that require tasks far beyond extracting data from individual documents. These challenges can be split into three key groups: modeling Web site navigation, the creation of a navigation plan for specific tasks or queries, and navigation plan execution.

Web site modeling challenges are related to understanding and representing in an abstract way aspects of Web site behavior that are important for Web information extraction. In the case of a few previous WIE systems that used any Web site modeling, in general the only modeled aspects are forms querying capabilities and types of pages (with respect to their templates). Only DWDI, Denodo (to some extent) and some early systems, such as Araneus, model in a general way navigation actions other than form filling.

The first modeling challenges that remains unaddressed is detecting and taking advantage of query capabilities exposed by the server-side Web application but not reflected in the user interface (e.g. the capability to query more attributes at the time than is suggested by used search form). In the case of some queries, such capabilities, if they exist, may significantly lower the overhead both in terms of the data records that need to be filtered out after extraction, and in terms of used transfer.

The next challenges are related to detecting and modeling the server limits of a number of allowed queries (global, per IP or per user), as well as the limits of a number of results returned per query.

Another group of challenges is related to modeling actions that can be performed in a specific Web site. While the modeling of HTTP requests was somehow resolved in a few systems, none of them have a complete approach to modeling complex user-interface actions (based on mouse, keyboard and window events), controls with "suggestion" functionality (that can significantly simplify the task of filling in open-domain fields) and actions connected to the flow of time (e.g. automatically performed each 15 minutes).

Another related challenge that was addressed by none of the state-of-the-art solutions, is to model a Web site's statefulness. It consists in understanding and modeling the impact of each action and auxiliary file loaded with HTML documents (such as images, style files or JavaScript files) on client-side and server-side session state (including modifying client-side and server-side session storage, setting Cookies, changing the behavior of the client application and loading new data with a specific presentation template).

Finally, modeling of standard Web site's navigation facilities, such as pagination, dynamic filters and data ordering, may be beneficial for navigation planning and actual navigation overhead.

Based on the information contained in Web sites and on the definition of a specific task (such as answering a specific user query or performing a complete crawl of a Web site), the next challenge is to define a query plan to execute this task.

It consists in defining navigation plan (i.e. what actions or classes of actions should be performed in which order?), necessary HTTP requests rules (i.e. which HTTP requests associated with individual navigation actions need and need not to be performed?), data extraction rules (what extraction rules should be applied in a specific client-side navigation state and how should be their output interpreted?) and post-extraction data filtering operations (i.e. rules for discarding some part of data that, due to a Web site's limited capabilities, were extracted but do not correspond to the query). Moreover, in the case of some stateful Web sites, it may be also required to define when an user session should be reset (e.g by removing the client cache and all Cookies).

Navigation planning is challenging, as it should take into considertation the Web site's navigation structure, query capabilities, limitations of a number of results, session evolution (adaptive or personalized character), as well as all other auxiliary information, such as the ordering of data in a specific navigation state. At the same time, out of multiple query plans the ones with lower cost (for example, lowest overhead transfer) should be selected.

The final group of challenges is related to performing navigation and information extraction according to the defined plan. Firstly, it includes multiple technical issues of HTTP server-interactions, such as dealing with redirects, client-side cache control, Cookies, HTTPS, HTTP error codes, and data compression.

Secondly, it concerns performing all Web actions, such as issuing HTTP requests or interacting with user-interface controls (including dynamic HTML and Flash or Java components) according to the navigation plan. In complex Web sites it requires handling synchronous and asynchronous requests as well as permanent connections in an aligned way, and managing the auxiliary files that are necessary for session maintenance and evolution (e.g. loading and executing necessary JavaScript files). If needed, as in the case of CAPTCHA controls, the system needs to incorporate user interaction into query execution.

Another challenge is to interpret the current navigation state and to apply, in a correct moment (e.g. only after all data were loaded), related data extraction rules. While in the majority of cases the interpretation of the current navigation state is trivial (i.e. it results from the URL address or from the sequence of followed actions), it is technically challenging in the case of multiple asynchronous actions that modify the browser state without replacing the previous document. In such cases, the challenge is to detect that data can be already extracted. Moreover, in some cases, it may be challenging to detect where the new data are located in the document (e.g. they may be mixed up with previous, already extracted data).

Apart from asynchronous, AJAX-like requests, the second situation when interpreting the current client-side navigation state may be problematic is when a Web site's behavior is unpredictable (e.g. in some Web sites, the user is directed to the list of results if there are more than one search result, but if there is only one result, the user is taken directly to the corresponding page), or when some technical problem occurs. Adapting to such unexpected situations is one of the key challenges of navigation in complex data-intensive Web sites.

The final challenge related to navigation plan execution concerns all data filtering activities done after information extraction is performed. It covers not only applying all post-extraction filters according to the navigation plan, but also dealing with expected and unexpected data duplication.

3.3.3 Large-Scale Information Extraction

Apart from the challenges discussed above, which concern almost all types of WIE activities in the contemporary Internet, there are a few challenges that need to be addressed in the case of large-scale Web information extraction, i.e. the extraction of millions of data objects from hundreds of Web data sources.

The first question in the case of large-scale information extraction is related to choosing the right sources for specific queries or tasks. This problem was very well studied in the area of databases, as well as for on-line text sources with keyword-based search facilities. However, advanced probing and statistics building features for Deep Web data sources and other complex data-intensive Web sites is still an area where innovative solutions may be proposed.

The second challenge of large-scale WIE systems, addressed so far only by the Denodo Platform, is related to performing information extraction in a distributed way. While it may seem relatively simple, to be well implemented it requires the modeling of which actions should be performed by the same agent from the same IP address, and which actions can, or even should, be performed by multiple different agents.

The third challenges related to large-scale WIE is related to mimicking as well as possible the normal user activities. On the one hand, it guarantees avoiding being detected as a bot (and its consequences, such as blocked access to content or cloaking, i.e. being served other content than normal users). On the other hand, it allows to interaction with the Web site in conditions as similar to "natural" as possible, thus minimizing the risk of errors and unpredictable behavior.

The final large-scale extraction challenge is related to the fact that complete extraction from some large Web sites may last multiple hours or days or be even infeasible. As a result, important, yet largely unaddressed, challenges are related to being capable to receive partial data, as well as to prioritize extracted data in a way that maximizes partial data utility.

3.4 Summary

In this chapter we reviewed the state-of-the-art solutions to the Web information extraction problem. We started by discussing different WIE system classifications from the previous research outlined in Section 3.1.1, and by defining our own WIE system comparison scheme in Section 3.1.2. Then we compared the state-of-the-art systems with respect to their features in Section 3.2.1, and performed their genetic analysis in Section 3.2.2. The performances of existing solutions were compared in Section 3.2.3. Finally, in Section 3.3, we discussed the challenges in the field of Web information extraction that are still to be addressed by future systems.

Chapter 4

Proposed Web Information Extraction Method

In this chapter we present our models and methods [152] of Web information extraction, which aim to overcome most of the challenges identified in Chapter 2. We start by discussing the objectives of our solution in Section 4.1. The two following sections introduce the basic components of our data extraction model (Section 4.2) and its essential concept, called the data extraction graph (Section 4.3). In Section 4.4 we provide algorithms that use the data extraction graph to perform data extraction. Next, in Section 4.5 we present adaptations of our approach enabling data extraction from stateful Web sites. Finally, in Section 4.6 we introduce algorithms that enable efficient query answering based on data extraction graphs.

4.1 Objectives of Proposed Solution

There are two key objectives of the solution that we present in this chapter. The first of them is to provide an extensible, configurable, and flexible data model for representing both the navigation and data extraction capabilities of complex data-intensive Web sites. The model should be able to concisely represent key aspects of data-intensive Web sites, including complex user interfaces using dynamic client technologies, AJAX-like requests and Web applications' statefulness on both client and server side.

The second objective is to provide a set of algorithms that enable data extraction and query issuing using the proposed model. The algorithms should be general (i.e. capable of working with very diverse Web site models), and extensible (i.e. making it possible to plug-in external logic in all key decision points).

4.1.1 Focus of Our Solution

As we observed in Chapter 2, there are three essential groups of WIE challenges: "dealing with" specific situations (i.e. providing extraction formalisms and mechanisms flexible enough to resolve specific situations), “taking advantage of” specific situations (i.e. making data extraction formalisms capable of optionally using some additional information when it is available), and “learning” (i.e. building algorithms capable of the creation and maintenance of extraction rules adapted to specific situations, which is at least partially automated).

Out of these groups of challenges, we decided to focus on the first two, leaving issues related to extraction rules learning to our future work. This decision is due to the fact that for many of the challenges that are central to this thesis, resolving related learning issues is impossible because necessary abstractions have never been proposed.

While developing our Web site modeling approach and all algorithms operating on the proposed model, we adopt two important assumptions. The first is that we should aim at limiting the needed transfer and number of HTTP requests rather than the processing power used by our solution. We adopt this assumption for a few reasons. Firstly, in modern businesses, transfer tends to be more expensive than processing power. Secondly, as we adopt the “politeness policy”[1] and the paradigm of closely mimicking user behavior, the number of HTTP requests that can be issued in a unit of time becomes very limited. As a result, acquiring content is much more time-consuming than its processing. Finally, any decrease in a number of requests makes it less probable that we will exceed Web site’s per IP, per user or per session limitations of a number of HTTP requests or bandwidth available in a period of time.

The second assumption is that both model and algorithms should be as general as possible, even if more efficient algorithms could be developed for some special cases of general problems. This is in line with our objective of providing a single very flexible solution to the problems of information extraction from a variety of complex data-intensive Web sites, rather than providing high-performance solutions applicable to specific groups of Web sites.

It is also to be noted that while we provide a partial implementation of our model (described in the next chapter), its development is not an objective of its own. It should be treated only as a proof-of-concept demonstration of validity of our algorithms. At the same time it gives an insight into the types of technical problems encountered while implementing information extraction from complex data-intensive Web sites.

4.1.2 Detailed Objectives

While designing our model and algorithms, we took into consideration many of the challenges identified in Chapter 2 and listed in Table C.1. We tried to focus specifically on challenges that were addressed by none or almost none of the existing Web information extraction systems, i.e. the challenges discussed in Section 3.3.

We present the list of detailed objectives of our research in Table 4.1. While many of these objectives refer to specific aspects of data extraction from complex data-intensive Web sites, they correspond to virtually all groups of challenges identified in Chapter 2. Therefore, apart from its description, for each objective we provide a reference to one or multiple challenges from our classification described in Table C.1.

[1]We borrow this term, denoting the limitation of parallel requests and of requests frequency, from the search engines industry [63].

Objective (challenge to address)	Corresponding chall. from Table C.1
1. Adopting a flexible approach to choosing the most valuable data during extraction (utility maximization)	VII.3.l – t
2. Combining multiple extraction languages using different features	V.1.a – b, V.1.e, VI.1.a – b, VI.1.r
3. Combining content-based and context-based extraction rules	VI.1.j
4. Connecting data from multiple pages into records	VI.2.d, VI.3.f
5. Dealing with AJAX-like requests	II.2.a – c, IV.4.e, V.1.h, V.1.j
6. Dealing with data duplicates	VIII.2.n
7. Dealing with different file formats	V.1.c – e
8. Dealing with non-deterministic navigation patterns	VIII.1.d, VIII.2.e
9. Dealing with technical issues of HTTP(S) (Cookies, redirects, encryption, compression, error codes, certificates)	II.1.a – i, IV.3.h
10. Dealing with timer-triggered Web actions	IV.1.j, VII.1.e
11. Dealing with Web site's statefulness	II.1.k – m, VII.1.f – h, VII.1.k – l, VIII.1.g
12. Enabling multiple alternative rules for augmented robustness and better maintenance	VI.4.m
13. Enabling user interaction during data extraction (e.g. in order to handle CAPTCHAs)	IV.1.c – e
14. Extracting data from complex data presentation structures	VI.3.a – e
15. Extracting hierarchies of unknown depth	I.1.j
16. Implementing advanced, state-aware query planning with multiple optimization steps for both navigation and extraction	II.2.f – g, III.1.a – b, III.1.e – h, III.1.m, VII.2.a – d, VII.3.a – h
17. Limiting number and frequency of requests	II.2.d – e, II.3.b
18. Supporting data extraction of graph data	I.2.c – f
19. Supporting distributed data extraction	II.2.j – l, II.3.f
20. Taking advantage of typical constructs of Web sites: pagination, in-built ordering and dynamic filters	VII.3.s, VIII.2.a
21. Using additional server query capabilities	III.1.i – l, IV.4.d – f, VII.1.c, VII.1.i
22. Using both encoded and user content for data extraction	V.2.a – c, V.2.e – h
23. Using incompletely specified rules	
24. Working with composite (multi-frame, using JavaScript and CSS) Web content	IV.3.a – h, V.1.f – j
25. Working with interactive content (Web actions based on interaction with user interface)	II.3.e, IV.1.a – d, IV.1.f – i, IV.3.h, IV.4.a – f, V.1.i – j, VII.1.d – e

Table 4.1: Detailed Objectives of Proposed Solution

While developing our model, we took into consideration all objectives listed in the above table. We assumed that it should directly address the majority of them, and that for the remaining ones initial ideas of future model extensions should be presented. While all of the listed objectives are important, eight challenges (marked in bold font) mostly unaddressed by previous solutions have the most significant impact on our model and algorithms.

Enabling the combination of multiple extraction languages (objective 2) is the basic requirement to enable data extraction of different granularities and using different features.

Objectives 5 (dealing with AJAX requests), 22 (using for data extraction both encoded and user content), 24 (implementing a document model rich enough to capture the complexities of content in different types of Web applications) and 25 (working with user interaction models) correspond to the key technological challenges of complex Web applications.

Dealing with a Web site's statefulness (objective 11) is the most complex modeling issue out of all the identified objectives. It is composed of several subproblems, such as: how should the state be understood and represented? How should its evolution be described? How should the current state be controlled (maintained, reset, recreated) with minimal transfer overhead? It is to be noted that none of these questions were answered by previous research.

Challenge 15 concerns the extraction of hierarchies of unknown depth, which is a rare example of data structure not handled by previous solutions.

Finally, the ability to model a few common data-oriented features of data-intensive Web sites (challenge 20) brings in a few possibilities of optimizations for some types of user query.

4.2 Basic Components of Proposed Model

In this section we present the basic static components that form our data extraction and Web site navigation model. This section covers seven specific topics: navigation states and Web actions, Web action templates, data extraction rules, extraction rules composition, extraction rule templates, data records and attribute sets. For each of them, we present both an intuitive introduction illustrated by examples, and formalized definitions.

4.2.1 Navigation State and Web Actions

Intuition

As discussed in Section 2.3.2, a user interacts with a Web site by using its user interface elements, presented in a Web browser. The sequence of such logically connected interactions in a single Web site, further referred to as *Web actions*, will be called a *navigation session*.

At any moment of a navigation session, interconnected information from one Web site is presented to the user in a single Web browser's tab or window. Even if this information forms a logically coherent *Web document*, it is typically composed of multiple relatively independent elements, such as the HTML code of one or multiple HTML frames, image and multimedia files, and client-side code libraries.

In complex data-intensive Web sites, during a navigation session, the Web document visible to the user is not the only information that continuously evolves. It leads us to the essential notion of *navigation state*, i.e. all information associated by server (*server-side navigation state*) and Web browser (*client-side navigation state*) with user navigation at a specific moment. Navigation state can consist, for example, of the current Web document, session-level Cookies and the contents of the shopping cart stored on the HTTP server.

All user actions that change the navigation state will be called Web actions. For example, clicking on a hyperlink or submitting a form in a frameless Web site is a Web action as it changes the contents of the Web browser. Similarly, adding a product to the shopping cart is a Web action as it changes the server-side information associated with the user. Other examples of Web actions include interacting with dynamic Web page elements (buttons, AJAX components, data presentation controls) or using history-based Web browser features (e.g. coming back to previously visited pages).

The observable change in navigation state (i.e. the changes that are visible in the Web browser) can be treated as a Web action's output and further used for information extraction.

We illustrate the notions of navigation state and Web actions by using two examples. The first of them demonstrates how Web actions modify client-side navigation state.

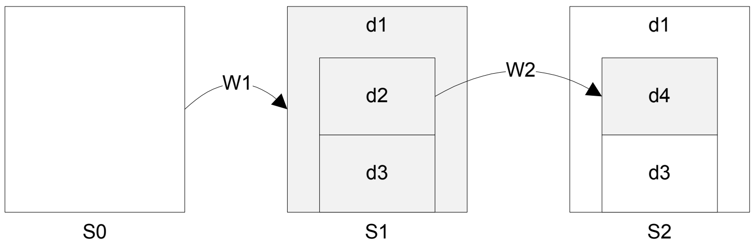


Figure 4.1: Example of Client-Side State Evolution

Example 4.2.1 Changes of Client-Side Navigation State

Let's assume that we start our navigation session with an empty Web browser window. As at this moment the Web browser does not associate any information with user navigation, this situation can be interpreted as an empty client-side navigation state $S0$.

We start the navigation by loading into the Web browser an HTML document containing a frameset with two frames. Complete loading of all three documents (frameset document $d1$ and two frames documents $d2$ and $d3$) forms the first Web action $W1$. Its execution changes the client-side navigation state from empty ($S0$) to containing all three loaded documents ($S1$). As we started our navigation in an empty navigation state $S0$, complete navigation state $S1$ can be interpreted as an output of Web action $W1$.

As the second step, we perform Web action $W2$ that consists in following a link (without a specified `target` attribute) in the first frame. As a result, the content of first frame is replaced by a new document $d4$, and a new navigation state $S2$ composed of $d1$, $d4$ and $d3$ is created. At this situation the part of $S2$ that corresponds to newly-loaded document $d4$ and the modification of the $d1$ *src* attribute of the first frame can be interpreted as a direct output of Web action $W2$.

This example is schematically depicted at Figure 4.1.

The second example focuses on server-side navigation state evolution.

Example 4.2.2 Changes of Server-Side Navigation State

Let's assume that in a Web site we are interested in two distinct aspects of server-side state: the authentication state (i.e. login of authenticated user) and the contents of the shopping cart. Let's also assume that navigation starts at Page 1 with an empty shopping cart, before the user has logged in. Even if server-side Web application logic has already associated some information with the current navigation session (e.g. some tracking information), it does not concern aspects of our interest. Therefore, we will treat the server-side part of navigation state $S0$ as empty.

Starting at $S0$, we perform three consecutive Web actions. $W1$ corresponds to logging in and loads into Web browser a completely new document. $W2$ is a background AJAX Web action and

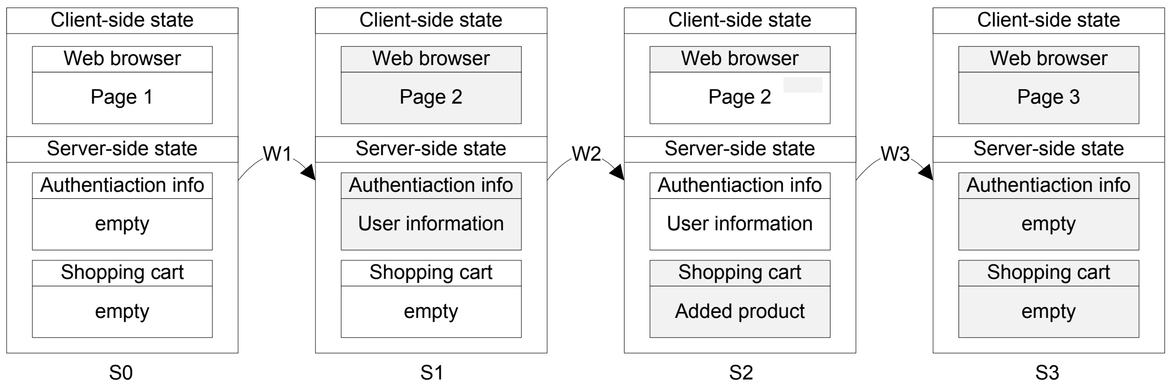


Figure 4.2: Example of Server-Side State Evolution

consists in adding an item to the shopping cart and modifying information on a number of items in the current Web browser document. Finally, $W3$ corresponds to logging out and removing all shopping cart information from the server.

The impact of these Web actions on both the client-side and server-side navigation state is demonstrated in Figure 4.2. $W1$ modifies authentication information at the server-side navigation state and completely replaces the current client-side navigation state. Its direct output consists of the content of the newly-loaded Page 2. $W2$ modifies part of the server-side navigation state that corresponds to the shopping cart. At the same time it introduces only a minor change in the content of Page 2 (symbolized by a small grayed box within the page), and it is this change that can be treated as direct output of $W2$ Web action. Finally, $W3$ completely changes both the client-side and server-side navigation state (by loading a new document into the Web browser and removing all authentication and shopping cart information from the server), returning the content of Page 3 as the direct output.

It is to be noted that while a Web action's direct output (i.e. the part of the current Web document that was changed by the Web action) is often the most important representation of new information, sometimes we are interested in a complete Web document or raw data that was transfered from the server (i.e. the encoded content). Thus, in our model we adapt a multi-valued representation of Web action's output that will be discussed in Section 5.2.1.

Formalization

In this section we formally define the terms informally introduced above. We start by presenting our understanding of the term "Web document".

Definition 4.2.1 Web Document
The set of all logically-connected files that are jointly displayed in a single Web browser window or tab, together with all files that have an impact on the dynamic behavior of displayed content, form a *Web document*. The top-level HTML file of a Web document will be referred to as a *Web document's main file*.

The Web document is the most visible part of the navigation state. However, navigation state consists also of other information stored both on client and server side, as described by the following definitions.

Definition 4.2.2 Client-Side Navigation State
The complete information associated by the Web browser with a Web site at a specific moment of a user's navigation session will be further referred to as the user's *client-side navigation state*.
The (possibly infinite) set of all client-side navigation states existing in a Web site x will be further denoted by C_x.

Definition 4.2.3 Server-Side Navigation State
All information associated with a specific navigation session and available to the server-side Web application logic at a specific moment of navigation session will be referred to as the *server-side navigation state*.
The (possibly infinite) set of all server-side navigation states existing in a Web site x will be further denoted by S_x.

The complete server-side state is not observable by the user. However, typical examples of the server-side navigation state, such as the content of the shopping cart, authenticated user, specific query issued by the user or the list of recently visited products are intuitively understood. For most types of server-side navigation state, an intuitive assumption that they are stored and play important role in a Web sites behavior can be made. Consequently, client-side and server-side navigation states are tightly logically connected and should be considered jointly. This connection is captured by the definition of navigation state:

Definition 4.2.4 Navigation State
Let S_x be the set of all possible server-side navigation states of a given Web site x and C_x be the set of all its possible client-side navigation states. Then, a *set of all navigation states N_x of site x* is as a subset of the Cartesian product of S_x and C_x (i.e. $N_x \subset S_x \times C_x$), such that any pair of client-side and server-side states in N_x is possible in Web site x. Each navigation state $n \in N_x$ can be represented as an orderer pair $n = (s, c), s \in S_x, c \in C_x$.

Apart from the distinction between client-side and server-side navigation state, the navigation state can be also often divided into some logically distinct parts. For example, in many on-line shops we may distinguish components related to product information visible in the Web browser from the content of a customer's shopping cart, and from the customer's authenticated session. We will use such a logical decomposition of navigation state further while discussing the state-aware version of our data extraction algorithms in Section 4.5.

As we discussed above, client-side or server-side navigation state can be modified by executing Web actions. What is most visible to users during their navigation actions, is the changing current Web document. However, it is often accompanied by the evolution of the invisible part of the client-side and server-side navigation state.

Therefore, we use the complete navigation state to define a Web action.

Definition 4.2.5 Web Action
Any function $w : N_1 \mapsto N_2 \times WO_x$, where $N_1, N_2 \subset N_x$, and $WO_x \subset \mathbb{R}$ corresponds to the set of all possible Web action outputs in Web site x, is called the *Web action of Web site x.*
The set of all Web actions of Web site x will be denoted as $W_x = \{w | w : N_1 \mapsto N_2 \wedge N_1, N_2 \subset N_x\}$.

For simplicity, we assume that all Web actions of interest to us are represented in a textual form, called *Web action representation.* For now, we also do not provide a formal definition of a set of data records $\mathbb{R}$, which will be introduced in Section 4.2.6.

The above definition focuses on the descriptive character of Web actions, i.e. how they relate two navigation states. However, Web actions are also executable, making possible navigation state transitions.

Definition 4.2.6 Navigation State Transition
The execution of Web action w on a navigation state $n_1 \in N_1$ corresponds to the *navigation state transition* from navigation state n_1 (*source navigation state*) to navigation state $n_2 = w(n_1)$ (*destination navigation state*). This will be further written as $n_1 \xrightarrow{w} n_2$, and the output of execution of w in n_1 will be denoted by $o = n_1 \xrightarrow{w} n_2$.
As $n_1 = (s_1, c_1)$ and $n_2 = (s_2, c_2)$, the notation $(s_1, c_1) \xrightarrow{w} (s_2, c_2)$ can be also applied.

It is to be noted that apart from the user-initiated actions that we discussed before, navigation state can be also modified by application-triggered events (e.g. the periodic download of new information). We consider such events as a special kind of Web action called *system-triggered Web action.* Dealing with system-triggered Web actions is out of scope of this thesis, and will be included in our future research (as discussed in Section 6.5).

4.2.2 Web Action Templates

Intuition

While many simple Web sites handle only static, click-based navigation with a finite number of possible Web actions, for many complex or application-alike Web sites, W_x is an infinite set. At the same time, in a typical data-intensive Web site there are multiple Web actions that are very similar both in the way they are defined (e.g. the URL that a link points to, or form element that a submit action is performed on) and the way they affect the navigation state. For example, in many electronic shops all links to product detail pages differ only by some product identifier, and all product detail pages loaded when links are followed are very similar with respect to their structure and the data they contain.

Intuitively, such similar Web actions can be generalized. We will call such generalizations *Web action templates* (WATs), and the parts of the Web action definition that differ for individual Web actions will be generalized into named input parameters (called *input slots*). A Web action template may have any number of independent, logically distinct input slots (including zero slots in the case of *trivial WAT*).

Below we present a basic example of WAT that corresponds to the generalization of a few URL-based Web actions.

Example 4.2.3 Generalization of Similar URLs

Let's assume that a Web site accepts, among others, the following URL addresses:

- `http://example.com/cat,books,list.html` - list of offered books
- `http://example.com/cat,books,pivot.html` - pivot table view of offered books
- `http://example.com/cat,cds,list.html` - list of offered CDs

These URLs can be generalized into WATs in at least two reasonable ways:

- two WATs `http://example.com/cat,{`*`category`*`},list.html` and `http://example.com/cat,{`*`category`*`},pivot.html` with a single input slot *category*, corresponding to the list and pivot view of items in a specific category respectively, and
- single WAT `http://example.com/cat,{`*`category`*`},{`*`view`*`}.html` with two input slots *category* and *view*.

The choice of best generalization will depend e.g. on similarity of outputs of all Web actions belonging to a WAT with respect to the information they contain and its presentation.

WATs not only generalize Web actions, but may be also used to create ones. For example, the provision of a pair of values (dvds, pivot) as input to the second exemplary WAT would generated a Web action `http://example.com/cat,dvds,pivot.html`.

Similar generalizations can be performed for more complex Web actions, as demonstrated by the following example.

Example 4.2.4 Generalization of Similar User Interface Web Actions

Let's assume that navigation in a Web site is based on HTML `button` elements with an associated `onclick` JavaScript code and unique identifiers (invisible to the user). Let's also assume that the buttons have unique texts.

In this setting at least two straightforward generalizations of "click a button" Web actions exist: one based on clicking a button identified by unique identifier (with identifier value being input slot), and the second one based on clicking a button that has a specific caption (being an input slot).

Again, choosing the right generalization will depend on the details of a Web site's behavior. In many cases using identifiers would be the better choice, as the assumption of unique texts rarely holds true in real-life Web sites.

Formalization

Web action templates can be defined formally as follows:

Definition 4.2.7 Web Action Template
Any function $wat : A_1 \times ... \times A_n \mapsto W_t, W_t \subset W_x$, where $A_1, ..., A_n \subset \mathbb{S}$ and $\mathbb{S}$ is a set of all possible text (string) values, is called a *Web action template* for Web site x with n input slots.

4.2.3 Data Extraction Rules

Intuition

The data-intensive Web sites that are our essential interest in this thesis typically contain at least partially structured information. When provided with a specific Web document, users can intuitively identify the structure of the information that it contains. Moreover, similar information structures tend to repeat across multiple Web documents obtained through similar sequences of Web actions. Data extraction rules are formalized generalizations of such similar structures, enabling the acquisition of data from the output of a Web action.

Data extraction languages (or methods) used to define data extraction rules differ not only in terms of the syntax they use, but also with respect to features they use and type of input they accept. As discussed in Chapter 2, features may be lexical, separator-based, semantic, template-oriented, structural or even visual. Extraction rules may be also only partially-specified, with final extraction performed with the aid of some heuristics.

Despite all these differences, all extraction rules share a few properties. All of them are specified in some extraction language. All of them can be represented in textual form. Finally, all of them require some content that the extraction rule will be executed on as input.

In our model, the output of an extraction rule consists of a set of records that have shared named attributes (called output slots). To assure flexibility, both records with a single attribute and records with automatically generated attribute names can be used. It is also possible that for some records some attributes are empty.

Our model makes it possible to use any extraction language with return values that can be represented as a set of records. While our implementation discussed in this thesis is mostly based on two languages (XPath and regular expressions), other existing solutions (including solutions for information extraction from text and partially specified rules) can be easily added.

Example 4.2.5 Regular Expression Extraction Rule

Let's suppose that we want to apply the following regular expression `/(\w2,)\s*(\d*?)\s+([\d\w]+)/ism` to Web page content. It searches the content of the Web page for all occurrences of patters composed of a required sequence of two or more letters, followed by zero or more space characters, an optional sequence of digits, one or more space characters and a sequence of alphanumeric characters.

In our approach, this expression will return zero or more records with four auto-named output slots: Subpattern0 (corresponding to complete matched text), Subpattern1 (corresponding to the word), Subpattern2 (corresponding to the optional number) and Subpattern3 (corresponding to a sequence of alphanumeric characters).

For example, applying this regular expression on the following content:
`<li>POP 69 8V</li><li>URBAN HDI</li><li>Abarth 135 TJet</li>`
would generate the following data records:

- Subpattern0 = ”POP 69 8V”; Subpattern1 = ”POP”; Subpattern2 = ”69”; Subpattern3 = ”8V”
- Subpattern0 = ”URBAN HDI”; Subpattern1 = ”URBAN”; Subpattern2 = ””; Subpattern3 = ”HDI”
- Subpattern0 = ”Abarth 135 TJet”; Subpattern1 = ”Abarth”; Subpattern2 = ”135”; Subpattern3 = ”TJet”

As we can see in this example, even if all of records have the same output slots, one of them (Subpattern2) is optional and can be empty for some records.

As demonstrated above, in the case of regular expressions multiple attributes correspond to different fragments of extracted text. It is the most intuitive situation. However, in some other languages different attributes will rather correspond to different representations of the same information, as we will show in an example of the XPath extraction rule.

Example 4.2.6 XPath Extraction Rule

Let’s assume that we want to extract all `<a>` elements from an HTML document by applying the following XPath expression: `//a`. Similarly as in the case of the regular expression extraction rule, multiple records can be returned as a result. However, their output slots will be created in a different way.

The first group of output slots, that will be returned is similar to the automatically generated “Subpattern$X$” slots and corresponds to the text of tag attributes (if it has any of them). For example, if our XPath expression matched the following HTML elements: `<a href="#" title="Do nothing">Link 1</a>` and `<a name="t1"/>`, there will be three automatically created output slots: `@href`, `@title` and `@name`. Similarly as with Subpattern2 in Example 4.2.5, some of them will contain empty values.

The second group of output slots will correspond to different representations of a selected node. Output slots of this type include `innerText` (the content of the element with all the tags stripped), `innerHTML` (the HTML representation of nodes embedded in the chosen one) and `element` (an object representation that can be used, for example, by other XPath rules).

The detailed documentation of all the implemented automated output slots of XPath rules can be found in Section 5.2.2.

To increase the flexibility of information extraction, our model also supports output slots (called *constructor input slots*) that are *constructed* by the concatenation of any automated output slots (such as slots corresponding to individual attributes or contained text in the case of XPath and all subpatterns in the case of regular expressions) and some constant text. This allows to reconstruct and decrease granularity of data extraction output or to add some fixed content. For example, if the year, month and day of some events are extracted as separate subpatterns, they can be combined and returned as a single output slot corresponding to complete date. In other situations we may use output slot construction to surround an extracted value by opening and closing tags, thus enabling the transformation of the extracted data into an XML file.

Formalization

Before formally defining extraction rule, we make a key observation that the format of input content required by different data extraction languages may significantly differ. For example, while the regular expression extraction rules may be performed on any object that can be represented as a string, XPath extraction rules require DOM elements as input.

In our model, we assume that any implemented extraction rule language is capable of performing extraction on a record corresponding to a Web action output (if needed, each language interpreter may perform some data conversion internally). However, we also assume that each language may accept also other formats of input values. The set of all additional input objects accepted by extraction language $\mathcal{L}$ will be denoted as $I_{\mathcal{L}}$. For example, possible additional inputs of regular expression rules include all string values, thus $\mathbb{S} \subset I_{\mathcal{L}_{regex}}$.

Based on the above statements on extraction rules input and outputs, we are able to provide a formal definition of the extraction rule expressed in language $\mathcal{L}$.

Definition 4.2.8 Data Extraction Rule
The data extraction rule defined for a Web site x expressed in language $\mathcal{L}$ can be defined as a function $er : WO_x \cup I_{\mathcal{L}} \mapsto 2^{\mathbb{R}}$ where $2^{\mathbb{R}}$ is a power set of all possible data records.
The set of all extraction rules possible in a Web site x will be denoted by E_x.

For the moment we keep using the intuitive understanding of data records. The formalized definition will be introduced in Section 4.2.6.

4.2.4 Extraction Rules Composition

As extraction rules can be performed not only on client-side navigation states but also on other types of values ($I_{\mathcal{L}}$), it is possible to plug into an extraction rule the output of some other previously executed extraction rule. This property of extraction rules makes them composable.

Example 4.2.7 Extraction Rules Composition

Let's assume that a Web page contains two types of information that we plan to extract: the list of all products, and the ranking of ten best-selling items. Let's also assume that both the product lists and "top ten" ranking are contained in a `<div>` element with attribute `id="content"`.

Then, data extraction can be performed by three extraction rules. The first of them er_1 (e.g. XPath expression `//div[@id='content']`) can be used to extract the content `<div>` element from the Web document. Two others rules (er_2 and er_3) may accept as input the value returned by er_1 and extract all the products and "top ten" items respectively.

In this example, two extraction rule compositions were performed: (er_1, er_2) and (er_1, er_3).

While composing two rules we need to specify which output slot of records returned by the first rule should be provided as input to the second rule. If er_2 and er_3 in the example were regular expression rules, we would use the `innerText` or `innerHTML` output slot (which are both strings). However, if both rules were XPath expressions, we would use the `element` value, which is a DOM element.

It is important to note that two extraction rules should be composed only if all possible values of the chosen output slot of the first rule er_1 are legal inputs of the second extraction rule er_2, i.e. they belong to the $I_{\mathcal{L}}$ of its language $\mathcal{L}$.

There are three key scenarios where the composition of extraction rules is an important mechanism of data extraction, namely:

- combining extraction rules in different languages – for example, the value identified using XPath extraction rules is further split into individual attributes by using regular expressions;
- decomposing complex extraction rules – for example, replacing one complex, hard to understand extraction rule with several simpler rules chained through composition;
- sharing a common part of multiple extraction rules - if multiple extraction rules have some common operation (this is the case presented in Example 4.2.7).

4.2.5 Extraction Rules Templates

Intuition

Extraction rules may have an absolute character, i.e. be expressed relative to the beginning of the document text or to its DOM root element, but often it is easier to express them relative to some easily-indentifiable element, e.g. an HTML tag with a unique `id` attribute, a specific separator or specific text value(s). Such element, separator or value will be further called the *hook of extraction rule*. As discussed in [175], in many cases, using extraction rules that are relative to a hook increases data extraction robustness to Web site modifications.

Data extraction hooks may have different characters. They may be fixed and used to identify some data region. For example, fixed text, such as “results found”, is often a good identifier of the start of data in search engine results. It may be also defined as a part of a user query. For example, an extraction rule may extract all products from a `div` element of a Web page, corresponding to a product category defined in a user query. Finally, it may consist of any previously extracted value. For example, some unique identifier of a record extracted in one location may be used as a hook of an extraction rule used in another location.

While it is often enough for a Web action to use a single hook, in some cases of complex data organization some more specific values may be necessary to identify the data to be extracted. For example in a typical pivot table, some hooks connected both to column and row labels may need to be specified. Moreover, in the case of a pivot table with hierarchical data organization, even more specific hooks can be used.

Similarly, as Web actions are generalizable to Web action templates, similar data extraction rules that differ only by hook value(s) can be generalized into *extraction rule templates*.

Example 4.2.8 XPath Extraction Rule Template With Hook

Let’s assume that in a Web site with a very complex layout and HTML code, each data record is represented as an `<tr>` element, with the first cell containing the car make name and the second one containing car details that we want to extract.

In such a situation we can use the text of the first cell as a hook for extracting interesting values. For example, to extract interesting values for all Audi cars, we could use the following XPath expression: `td[text()='Audi']/../td[2]`. Similar rules can be created for other makes.

All such rules can be generalized into a single *extraction rule template* `td[text()='{make}']/../td[2]` with a *make* input slot. Such ERT can be used to create individual extraction rules. For example, when provided with input value BMW, it will create an XPath extraction rule `td[text()='BMW']/../td[2]`.

It is to be noted that input slot values can be used not only inside the extraction rule specification (as in the above example), but also in the definition of some constructed output slots. For example, if an extraction rule that has an input slot `make` extracts car models of a given make and makes them available as a `Subpattern1` output slot, an additional output slot `car` can be defined to combine these two values, e.g. as follows: `{make}: {Subpattern1}`.

Formalization

We define extraction rule templates formally as follows:

Definition 4.2.9 Extraction Rule Template
Any function $ert : A_1 \times ... \times A_n \mapsto E_x$, where $A_1, ..., A_n \subset \mathbb{S}$, is called an *extraction rule template* for Web site x with n input slots.

4.2.6 Data Records and Data Records Schema

Until now we referred several times to the notion of data records. We stated that a set of Web actions outputs is a subset of a set of data records $\mathbb{R}$. We defined extraction rules output as being included in the power set of $\mathbb{R}$. Finally, we repeated a few times informally that the output of the whole extraction process consists of data records. In this section we analyze the notion of "data records" in more detail and introduce the term *data records schema.*

Intuition

As we wrote previously, data records are composed of string values with associated attribute names. The advantage of such a representation is that it is very flexible. However, missing type and semantic information makes the processing and interpretation of extracted data more difficult. For that reason we introduce the notion of data records schema, which assigns to each attribute the name, its data type and some optional semantic annotation.

As we stated before, some (or even all) attributes of data records may be optional. Thus, it is possible that records that share the same schema have only partially overlapping attributes. In such situations, schema should assign data type and semantic annotation to each of the attributes present in at least one data record.

Example 4.2.9 Different Attributes

Let's assume that the extraction rule expressed in some language returns values with four attributes: First name, Last name, Company name and Birth year, all of which are optional. The records returned by the extraction rule are:

- "First Name" = "Bill"; "Company name" = "Microsoft",
- "Last Name" = "Jobs"; "Company name" = "Apple",
- "First Name" = "Jeff"; "Last name" = "Bezos".

As we can see, each record has a different combination of attributes and Birth year is present in none of them. However, given the definition of extraction rule, we know that it is possible that all attributes are returned for some data record. Thus, the data records schema should assign data type (in this case string) and semantic annotation (e.g. class and attribute from some business-oriented ontology) to all four attributes.

We adopt a very flexible approach to data types (common for script programming languages), in which a single value can represent (be *convertible to*) multiple data types, and that proper input data format is sufficient for interpreting a value as representing specific data type. Out of all data types that the attribute is convertible to, one needs to be selected as the target data type. All extracted values for this attribute will be converted to this type.

Example 4.2.10 Data Types of Extracted Data

Let's suppose that in an e-commerce Web site the following values are provided as the prices of individual products: 100, 75, 30.50 (we assume that currency is implicit or extracted as a separate attribute). These values are convertible to the following data types: string, float and decimal. However, they are not convertible to integer, as value "30.50" contains decimal part. Out of these types, decimal would be the most natural target data type.

If in the same Web site for some products the value in the price column was "Ask the seller!", we would have two options of price extraction. The first would be to extract all the values and chose "string" as target data type. The second would be to have two separate attributes (extracted from the same content): one extracting only decimal values (e.g. by enforcing data format with regular expression) and another one extracting only non-numeric values.

Formalization

We start by formalizing our understanding of data type:

Definition 4.2.10 Data Type
We will denote by *data type* t a function $t : \mathbb{S}_t \mapsto \mathbb{O}_t$, where $\mathbb{S}_t \subset \mathbb{S}$ is a set of all string values that can be converted (are convertible) to data type t and $\mathbb{O}_t$ is a domain of data type t.
We will denote by $\mathbb{T}$ the set of all data types.

Next we define the data records schema:

Definition 4.2.11 Data Records Schema
We will call *data records schema* drs any function $\mathbb{S} \mapsto \mathbb{T} \times \mathbb{S}$ that assigns to each attribute name $a \in \mathbb{S}$ a pair (t, san) of data type $t \in \mathbb{T}$ and semantic annotation $san \in \mathbb{S}$.

In this thesis we do not provide details of how semantic annotation should be represented (i.e. what ontologies and ontology languages should be used and how the actual mapping of values to ontology classes or instances should be assigned). However, we make it a part of our proposed model to let us further develop initial ideas presented in our previous work [9].

Based on the notion of data records schema, we formally define the term data record as follows:

Definition 4.2.12 Data Record
A *data record* dr consistent with data records schema drs is a pair of a) sequence of n pairs $(a_i, v_i); 1 \leq i \leq n; a_i, v_i \in \mathbb{S}$ of attribute value v_i and attribute name a_i, and b) of data records schema drs such that $\forall_{1 \leq i \leq n} drs(a_i)$ is defined and $\forall_{1 \leq i \leq n} (t_i, san_i) = drs(a_i) \rightarrow v_i \in \mathbb{S}_{t_i}$
The data records schema of any record dr will be further referred to as $drs(dr)$.

To simplify some of the further described algorithms, we assume that each data record is executable and that when it is executed it returns itself. In contrast with extraction rules and Web actions that require as input respectively some content and navigation state to be executed, records have no input parameters.

4.2.7 Attribute Sets

In the previous sections we introduced Web actions and Web actions templates that can be used for modeling Web site navigation, as well as extraction rules and extraction rules templates that model well data that can be extracted from the Web site. Together, these abstractions capture well the most important aspects of information extraction from complex Web sites.

However, they have one significant limitation. Both Web actions and data extraction rules reflect how data is organized in the Web site. Meanwhile, answering a user query may require significantly different data organization. For this reason we introduce to our model an additional, data-oriented abstraction called *attribute set*.

Web actions can be generalized into Web actions templates, and extraction rules can be generalized into extraction rules templates. Similarly, *attribute sets* (AS) can be thought of as generalizations of data records. Thus, attribute sets can be understood both as the specifications of data contained in data extraction output records, and as a component capable of generating data records with a shared data records schema. An attribute set's input slots correspond automatically to all attributes of its schema.

Example 4.2.11 Simple Attribute Set

Let *as* be an attribute set with schema *drs* that has three string attributes (`Make`, `Model`, `Vehicle State`) and one decimal attribute (`Minimal Price`). Optionally all of these attributes may have some semantic annotation.

Let's assume that *as* receives through its input slots the following combinations of values:

- `Make` = "Peugeot", `Model` = "207", `Vehicle State` = "new", `Minimal Price` = "10000",
- `Make` = "Peugeot", `Model` = "308", `Vehicle State` = "used", `Minimal Price` = "15000",
- `Make` = "Renault", `Model` = "Megane", `Vehicle State` = "new", `Minimal Price` = "20000".

Then, the attribute set will generate three records consistent with *drs* corresponding 1-to-1 to provided combinations of input values.

Even in such simple form, attribute sets play an important role in our model, as they make possible the merging of data from multiple sources through the provenance-aware join (discussed further in Section 4.4.6). However, to make attribute sets even more powerful we enhance them with basic data filtering capabilities (supporting both projection and selection operations). We demonstrate these capabilities by the following example.

Example 4.2.12 Attribute Set Selection and Projection Operators

Let *as* be defined as in Example 4.2.11. Moreover, we define for it the following filtering operations:

- selection rule: choose only records such that `Make` = "Peugeot" $\wedge$ `Vehicle State` = "new",
- projection rule: return only `Model` and `Minimal Price` attributes.

If the attribute set received the same input as in the previous example, due to application of the selection rule only the first record will be returned (the second record has the wrong vehicle type while the third one has the wrong make). Moreover, the output records will be limited only to two attributes, due to application of the projection rule. Thus, in this setting the attribute set will generate the following record:

- `Model` = "207", `Minimal Price` = "10 000".

In this case, attribute set input slots would correspond to all four attributes of its schema, while its output slots will consist of `Model` and `Minimal Price` attributes, as defined by the projection rule.

Below we present a formal definition of attribute set.

Definition 4.2.13 Attribute Set

Attribute set *as* is a triple $rs = (drs, \sigma, \pi)$. *drs* is a data records schema defining an attribute set's input slots, their data types and semantic annotations. σ, called selection operator, consists of a set of conjunctive predicates. Only inputs that meet all these predicates are used to create the output records of an attribute set. Finally, π, called projection operator, is a subset of *drs* attribute names that are used to create attribute set's output records.

The input slots of an attribute set correspond to all attribute names from *drs*, and all records created by the attribute set have attributes corresponding to the set of attribute names π.

4.3 Data Extraction Graph

In previous sections we introduced the key components of our navigation and data extraction model. We have also presented how individual objects – Web actions, data extraction rules and data records – may be generalized into Web action templates, extraction rule templates and attribute sets.

Until now we described each of these key building blocks of our model separately. However, in real-life scenarios all of them are interconnected by flows of data and control. In this section we analyze these interconnections and present the essential structure of our data extraction model, called the *data extraction graph*.

4.3.1 Data Extraction Nodes, Instances and Atoms

Intuition

Web action templates, extraction rules templates and attribute sets, while covering significantly different aspects of navigation and extraction, share a few common traits. All of them accept some input through zero or more input slots. All of them can use provided input to create one or more executable objects (Web actions, extraction rules and record sets). To be executed, the created objects may require some input of their own (navigation state in the case of Web actions, content in the case of extraction rules). Finally, when executed, all these objects return some values that can be represented as data records.

These similarities enable us to introduce three important abstractions: *data extraction nodes* that generalize WATs, ERTs and ASes, *data extraction instances* that generalize Web actions, extraction rules and record sets, and *data extraction atoms*, each of which consists of a pair of an instance and its input data.

We also introduce the processes of *node instantiation*, i.e. the creation of data extraction instances based on data extraction nodes, *atom creation*, i.e. the assignment of input to a data extraction instance, and *node execution*, i.e. the generation of individual data records.

Data extraction nodes (also called *nodes*) are characterized by their input and output slots. A node's inputs slots correspond to all the input attributes that are needed for its instantiation and for the execution of created instances. Thus, it includes all input slots of WAT, ERT or AS, as well as input slots required by their instances (i.e. navigation state in the case of Web action, and content in the case of extraction rule). A node's output slots are equal to the names of attributes of a data records schema that is shared by all records obtained by executing all atoms created by all the node's instances.

Input provided to some input slots of a node can be used to perform node instantiation, i.e. the creation of instances such as Web actions, data extraction rules and data records. Exemplary instantiations of a WAT and ERT were described in Example 4.2.3 and Example 4.2.8, respectively.

In the majority of cases the instances created by node instantiation still require some input to generate output data records. For example, extraction rule `td[text()='BMW']/../td[2]`, created as an instance of ERT described in Example 4.2.8, still requires some content that it will be executed on.

When an instance is provided with all input values it requires (passed to an other subset of a node's input slots than the input slots used for node instantiation), it forms a data extraction atom. In this example the same extraction rule may be used to create multiple atoms when provided with different Web documents as input. The most important property of an atom is that it can be executed, returning a set of data records.

All records returned by all atoms created by all of a node's instances can be interpreted as node's output for a specific set of input values.

It is to be noted that attribute sets are a very special type of data extraction node. Their instances are data records that require no extra input to be executable. Thus, they are both data extraction instances and data extraction atoms. Moreover, when executed, each data record returns a set composed of a single record (equal to itself).

The following three examples demonstrate the notions of node, instance and atom on examples of nodes corresponding to WAT, ERT and AS, respectively.

Example 4.3.1 Nodes, Instances and Atoms (WAT)

A node corresponding to the WAT of type `TopNavigateToURL` with the definition `http://example.com/cat,{category},pivot.html` will have two input slots: `category` and special input slot `!state` corresponding to the input of its instances (i.e. current navigation state). Its output slots will cover multiple interpretations of changes induced by the Web action (see: Section 5.2.1) including `innerText` (the text representation of the complete Web document), `innerHTML` (the HTML representation of the complete document), `newInnerText` (the new text generated or downloaded as result of the Web action) and `newInnerHTML` (the HTML representation of all page regions that were changed by the Web action).

For input consisting of two values for the `category` input slot ("books" and "cds") and some value provided for the `!state` input slot, there will be two instance Web actions created:

- `NavigateToURL(http://example.com/cat,books,pivot.html)`, and
- `NavigateToURL(http://example.com/cat,cds,pivot.html)`.

Each of them will be provided with the value of the `!state` input slot as its input, thus forming a total of two data extraction atoms:

- `{NavigateToURL(http://example.com/cat,books,pivot.html), !state}`, and
- `{NavigateToURL(http://example.com/cat,cds,pivot.html), !state}`.

Both of them will be further executed, returning a total of two data records with attributes including `innerText`, `innerHTML`, `newInnerText` and `newInnerHTML`.

These two records are the node's output for the provided combination of input values.

Example 4.3.2 Nodes, Instances and Atoms (ERT)

A node corresponding to XPath ERT with the definition `//a` will have a single input slot `!input` corresponding to content on which will be executed its instances, and a few output slots (`@href`, `@title`, `@name`, `innerHTML`, `innerText`) as discussed in Example 4.2.6.

Independent of the input that node receives, it will always generate a single instance `XPath(//a)`. However, this instance will form different atoms, depending on `!input` value. For example, if three different `<div>` HTML elements are provided as values for the `!input` input slot of this ERT, three different atoms will be created. Each of these atoms will be executed, returning zero or more records with a shared data records schema containing `@href`, `@title`, `@name`, `innerHTML` and `innerText` attributes.

The union of all these sets of records will form ERT's output for the provided input values.

Example 4.3.3 Nodes, Instances and Atoms (AS)

A node corresponding to the attribute set from Example 4.2.12 would have four input slots (`Make`, `Model`, `Vehicle State` and `Minimal Price`) corresponding to the input slots of the attribute set. When provided with three combinations of values from the mentioned example, it will generate a single instance, i.e. a data record `Model` = "207", `Minimal Price` = "10 000".

As this data record does not require any additional input, the single created atom will correspond to the record itself. Moreover, when executed, the atom will return the same record as node's output for the provided inputs values.

Formalization

We start by defining data extraction instances.

Definition 4.3.1 Data Extraction Instance

A *Data extraction instance* is a function $\alpha^{(n)} : IS_1 \times ... IS_n \mapsto 2^{\mathbb{R}}$, where $IS_i \subset \mathbb{O}$ for $1 \leq i \leq n$, $\mathbb{O}$ is a set of all possible objects of any data type, and $n \geq 0$ is a number of instance input slots, such that all output data records share the same data records schema, i.e. $\forall_{r \in \alpha^{(n)}(o_1,...,o_n)} drs(r) = drs_{\alpha^{(n)}}$.

Data extraction instances can be interpreted as a generalization of Web actions, extraction rules and data records.

Data extraction atoms can be defined as follows:

Definition 4.3.2 Data Extraction Atom

Data extraction atom ϵ is a pair $(\alpha^{(n)}, (o_1, ..., o_n))$ of a data extraction instance $\alpha^{(n)}$ and single input object o_i $(1 \leq i \leq n)$ for each of its inputs slots.
Operation of the assignment of $(o_1, ..., o_n)$ to a specific data extraction instance $\alpha^{(n)}$ will be called *atom creation* and the operation of calculating $\alpha^{(n)}(o_1, ..., o_n)$ will be called *atom execution*.
The set of all data extraction atoms will be denoted by A.

Finally, below we present the definition of data extraction nodes, which can be interpreted as a generalization of Web action templates, extraction rule templates and attribute sets.

Definition 4.3.3 Data Extraction Node
The *data extraction node* if a function $\gamma : S_1 \times ... \times S_k \times O_{k+1} \times ... \times O_n \mapsto A$, where $S_i \subset \mathbb{S}, 1 \leq i \leq k$ and $O_i \subset \mathbb{O}, k+1 \leq i \leq n$, such that

- $\epsilon = \gamma(s_1, ..., s_k, o_{k+1}, ..., o_n) \leftrightarrow \epsilon = (\alpha^{(n-k)}, (o_{k+1}, ..., o_n))$, i.e. the object inputs of data extraction nodes are used as instance inputs in output atom definition,
- $\forall_{i \in (1,...,k)} s_{i,1} \neq s_{i,2} \wedge (\alpha_1^{(n-k)}, (o_{k+1}, ..., o_n)) = \gamma(s_{1,1}, ..., s_{k,1}, o_{k+1}, ..., o_n) \wedge (\alpha_2^{(n-k)}, (o_{k+1}, ..., o_n)) = \gamma(s_{1,2}, ..., s_{k,2}, o_{k+1}, ..., o_n) \rightarrow \alpha_1^{(n-k)} \neq \alpha_2^{(n-k)}$, i.e. all inputs have an impact on instances that are part of output atom,
- $\forall_{s_1 \in S_1, ..., s_k \in S_k} (\alpha^{(n-k)}, (o_{k+1}, ..., o_n)) = \gamma(s_1, ..., s_k, o_{k+1}, ..., o_n) \rightarrow drs_{\alpha^{(n-k)}} = drs_\gamma$, i.e. for all combinations of string inputs, the instance data record schema is the same and will be denoted as drs_γ.

A set of all extraction nodes will be further denoted by Γ.

An important property of data extraction nodes is that they possess input and output slots as defined below.

Definition 4.3.4 Node's Input Slots
The string input arguments $s_1, ..., s_k$ of data extraction node γ are called *node instantiation input slots* and a node's objects input arguments $o_{k+1}, ..., o_n$ are called *atom creation input slots.*
Together, they are called a *node's input slots.* To assure human-readable character, node's input slots will be referred to with string names $is(\gamma) = (a_i, ...a_n)$, and all names of atom creation inputs slots will be preceded by "!" sign.

Definition 4.3.5 Node's Output Slots
The set of attribute names of data records schema drs_γ will be called a *node's output slots* of data extraction node γ, and will be referred to as $os(\gamma)$.

While introducing extraction rule templates in Section 4.2.3, we demonstrated that an extraction rule may have, apart from fixed output slots such as `innerText`, some automatically generated output slots. Now, we assume that such capability can be generalized to all data extraction nodes, and that some of the output slots can contain not only values generated directly by a given node, but also any combination of values provided to a node's input slots and of additional string constants. We will call such output *constructor output slots.*

As we discussed above, some nodes (parameterless Web action templates or extraction rule templates) do not require any input to be instantiated. Below we define them formally.

Definition 4.3.6 Trivial Data Extraction Node
A data extraction node that has zero node instantiation input slots is called a *trivial data extraction node.*

Trivial data extraction nodes do not require any input data for instantiation. Thus, they play an important role in the data extraction process, as they are starting points for the data extraction algorithm (see: Section 4.4.3). Examples of trivial input nodes include some WATs (such as visiting the home page of a Web site) and all kinds of input nodes, discussed in Section 4.3.6.

4.3.2 Input Mappings

As data extraction nodes have both input and output slots, intuitively outputs generated by a node's atoms (that correspond to a node's output slots) can be mapped onto the input slots of other nodes.

The most basic example of such a connection is an edge going from a WAT towards an ERT, meaning that after each WAT instance is executed, the Web browser's content should be used to instantiate ERT. An edge connecting an ERT and a WAT means that data extracted by ERT instantiation and instances execution should be used as input to the WAT (e.g. defining a part of the URL). In a similar way, an edge may exist between ERT and AS (corresponding to the generation of output data of the whole extraction process), and between two ERTs (corresponding to the extraction rules composition, i.e. the situation where the output of the first ERT instances execution is used as input for instantiation of the second ERT).

Data extraction nodes are an intuitive generalization of WATs, ERTs and ASes. Similarly, different kinds of relations between atoms output and nodes input can be generalized into information flows between nodes called *inputs mappings*.

Input mapping is a description of the information-level dependency between two nodes. A directed input mapping between nodes A and B specifies that each of output atom generated by node A can be used as an input of node B. In the simplest form of input mapping all output slots associated with node A are connected to all input slots of node B. However, more complex connections are also possible, as discussed in Section 4.3.4.

Below we present the formal definition of input mapping.

Definition 4.3.7 Input Mapping
An input mapping λ is an ordered triple $(\gamma_s, \gamma_d, l_\lambda)$, where

- $\gamma_s \in \Gamma$ is an data extraction node called source node, and further denoted by $s(\lambda)$,
- $\gamma_d \in \Gamma$ is an data extraction node called destination slot, and further denoted by $d(\lambda)$, and
- l_λ is an input mapping label $l_\lambda = ((os_1, is_1), ...(os_m, is_m))$ consisting of a sequence of m pairs (os_i, is_i) where each $os_i \in os(\gamma_s)$ is name of the γ_s output slot and each $is_i \in is(\gamma_d)$ is a name of γ_d input slot.

A set of all input mappings will be further denoted by Λ. A set of all input mappings incoming to a node γ of data extraction graph D will be denoted by $iIM(\gamma) = \{\lambda \in IM(D) : d(\lambda) = \gamma\}$ and a set of all input mappings outgoing from γ will be denoted by $oIM(\gamma) = \{\lambda \in IM(D) : s(\lambda) = \gamma\}$.

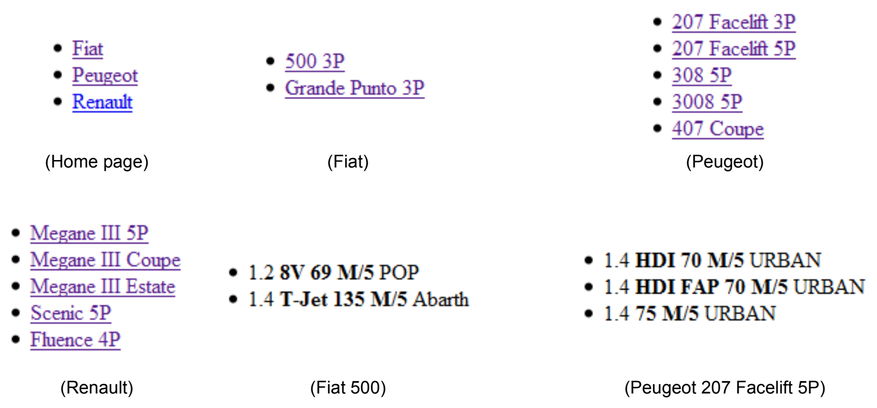
Input mapping concerns specific output slots of its source node, called *input mapping source slots*. Similarly, it provides input for the specific input slots of its destination node, called *input mapping destination slots*. We define both these terms below.

Definition 4.3.8 Input Mapping Source Slots
Given an input mapping $\lambda = (\gamma_s, \gamma_d, l_\lambda)$, the union of all indexes of output slots of source node γ_s defined as $ss(\lambda) = \bigcup_{i=1,\dots,m} os_i$ will be called (the set of) source slots of input mapping λ.

Definition 4.3.9 Input Mapping Destination Slots
Given an input mapping $\lambda = (\gamma_s, \gamma_d, l_\lambda)$, the union of all indexes of input slots of destination node γ_d defined as $ds(\lambda) = \bigcup_{i=1,\dots,m} is_i$ will be called (the set of) destination slots of input mapping λ.

4.3.3 Data Extraction Graph

Together, nodes and input mappings form the data extraction graph that describes how navigation and data extraction can be performed in a Web site. We demonstrate this notion by the following example.



- Fiat
- Peugeot
- Renault

(Home page)

- 500 3P
- Grande Punto 3P

(Fiat)

- 207 Facelift 3P
- 207 Facelift 5P
- 308 5P
- 3008 5P
- 407 Coupe

(Peugeot)

- Megane III 5P
- Megane III Coupe
- Megane III Estate
- Scenic 5P
- Fluence 4P

(Renault)

- 1.2 **8V 69 M/5** POP
- 1.4 **T-Jet 135 M/5** Abarth

(Fiat 500)

- 1.4 **HDI 70 M/5** URBAN
- 1.4 **HDI FAP 70 M/5** URBAN
- 1.4 **75 M/5** URBAN

(Peugeot 207 Facelift 5P)

Figure 4.3: Screen Shots of Six pages of Demo Web Site

Example 4.3.4 Complete Data Extraction Graph
Let us consider a very simple Web site with basic HTML code, which consists of three classes of pages: the home page containing a list of car makes with corresponding links to make pages, that contain a list of models of a specific make with links to model pages where individual car versions are listed. Screen shots of the home page, all three make pages and two exemplary model pages are depicted in Figure 4.3.

In Figure 4.4 we present an example of data extraction graph describing data extraction model for this demonstration Web site. It contains three WATs corresponding to three types of pages, a few extraction rule templates and a single attribute set.

The individual nodes of the exemplary graph are described in Table 4.2. As is visible from the table, WAT1 is a trivial node (it has no input slots), and all other WATs have a single input slot that corresponds to the URL to which navigation will be performed. In the case of all WATs their `userContent` output slot is used, which corresponds to the Web browser's content after Web action execution. As none of the WATs has a `!State` input slot, their execution is not dependent on current client-side navigation state.

ERT1 and ERT2 both are some XPath extraction rules that extract a collection of `<a>` HTML elements. Both of them have two output slots: `@href` attribute is mapped onto the `url` input slot of the following WAT and `innerText` is mapped onto the AS1 input slots `make` and `model`, respectively.

ERT3 and ERT4 are an example of the composition of extraction rules of two types. ERT3 is an XPath rule executed on Web browser's content. The string representation of HTML code of each element extracted by instances of ERT4 (i.e. ERT4's `innerHTML` output slot) are mapped onto the `!Input` input slot of ERT4. ERT4 is a regular expression with five subpatterns. Output slots that correspond to these five subpatterns are mapped onto the `engine displacement`, `engine type`, `engine power`, `gearbox type` and `version name` inputs slots of AS1.

Node	Type	Input Slots	Used output Slots
WAT1	NavigateToURL	-	userContent
ERT1	XPath	!Input	@href, innerText
WAT2	NavigateToURL	url	userContent
ERT2	XPath	!Input	@href, innerText
WAT3	NavigateToURL	url	userContent
ERT3	XPath	!Input	innerHTML
ERT4	RegEx	!Input	Subpattern1, Subpattern2, Subpattern3, Subpattern4, Subpattern5
AS1	Attribute set	make, model, engine displacement, engine type, engine power, gearbox type, version name	n/a

Table 4.2: Description of Nodes of Graph From Figure 4.4

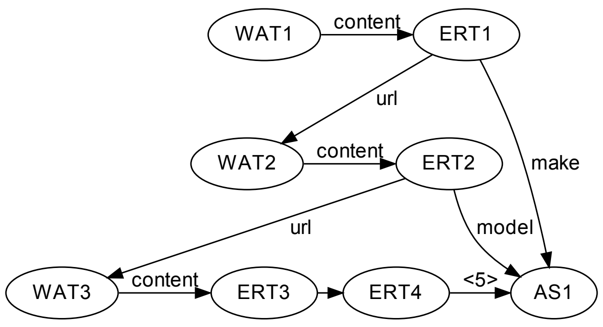


Figure 4.4: An Example of Data Extraction Graph

Formalization

Definition 4.3.10 Data Extraction Graph
We will call the data extraction graph a directed, connected graph $D = (N, IM)$ such that

- N is a set of vertices $N \subset \Gamma$,
- IM is a set of labeled edges $IM \subset \Lambda$, such that all source and destination nodes of input mappings belong to set N, i.e. $\forall_{\lambda \in IM} \gamma_s(\lambda) \in N \wedge \gamma_d \lambda \in N$.

The only formal requirement imposed on the data extraction graph is that it be connected. Therefore, our model covers both acyclic (as in the previous example) and cyclic graphs.

4.3.4 Input Configurations

Intuition

In previous sections we discussed three important properties of the data extraction graph. The first of them is that each node may have multiple input slots. The second is that for instantiation each node requires some input for all of its input slots. Finally, as there may be multiple input mappings with the same destination nodes, in some cases inputs for a node need to be built up as a combination of data coming from multiple data extraction nodes. These three observations led us to the important notion of *input configuration*.

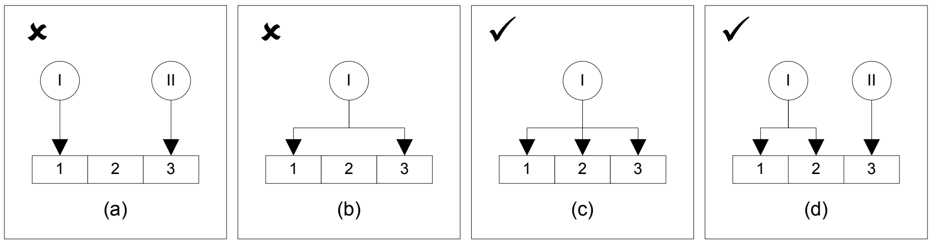


Figure 4.5: Basic Input Configurations

Informally, input configuration is a group of input mappings used together to instantiate a data extraction node. Below we give a few examples of valid and invalid input configurations.

Example 4.3.5 Basic Input Configurations
Figure 4.5 shows four basic examples of correct and incorrect input mappings for a node with three input slots (numbered 1, 2 and 3). In Figure 4.5.(a) and (b) we present two examples of combinations of input mappings that are not complete, and thus do not form correct input configurations. In (a), there are two different input mappings from nodes I and II to slots 1 and 3 respectively, but there is no input provided for the slot 2. As a result, instantiation of the node is impossible. The situation presented at (b) is similar; however, in this case both input for slot 1 and 3 comes from the same node I. Again, as input for slot 2 is missing, this combination of input mappings is not a valid input configuration. In both cases, a correct configuration may be created by adding an input for slot 2.

Two other examples in Figure 4.5 present valid input configurations. (c) represents the situation where all input slots receive input from the same node I. In most cases it would mean that the three distinct output slots of node I would be mapped onto input slots 1, 2 and 3. However, in some cases it is possible that the same output slot is used to fill in two input slots. In both cases, for each output data record a single triple of inputs would be created with one value for each of the input slots. This situation corresponds, for example, to the mappings WAT1 to ERT1 or ERT1 to WAT2 in Figure 4.4.

(d) presents a more complex example, where input to the node is provided from two nodes I (two attributes mapped into slots 1 and 2) and II (one attribute mapped into slot 3). Similarly, input from three nodes exists for node AS1 in Figure 4.4. In this situation, the output data records of nodes I and II need to be combined to form complete input records by applying a special operator, called a provenance-aware join (denoted by $\bowtie$). Thus we can note semi-formally that inputs coming in this case to the node will be calculated as $I \bowtie II$. This topic will be discussed in more detail in Section 4.4.6.

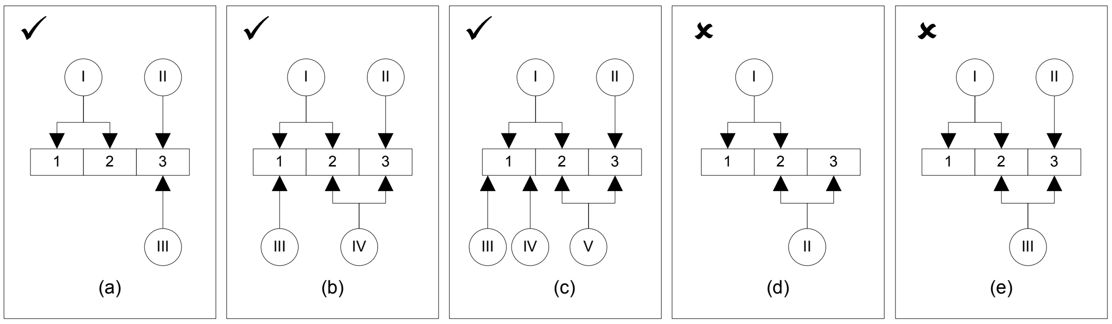


Figure 4.6: Advanced Input Configurations

Until now we discussed the situation when there is a single input for each input slot of the node. However, it is possible that a single input slot receives complementary inputs from multiple sources, and that multiple input configurations exist. We illustrate such situations, as well as two more invalid input configurations, in the following example.

Example 4.3.6 Complex Input Configurations

Figure 4.6 presents five different sets of input mappings coming to a node with three input slots (numbered 1, 2 and 3). The first example (a) shows a situation when two input sources are mapped into slot 3. As a result, two input configurations exist for this node: one consisting of inputs from nodes I and II, and one consisting of inputs from nodes I and III. In such situations we will calculate the actual input to the node as union of inputs received and joined by each configuration separately. Thus, semi-formally we would write that in this case input is calculated as $(I \bowtie II) \bigcup (I \bowtie III)$.

The situation depicted in (b) also contains two input configurations. The difference is that in this case, in contrast with (a), these two configurations do not share any input mapping. In this case, the node's input is the following: $(I \bowtie II) \bigcup (III \bowtie IV)$. (c) combines both previous examples, resulting in three input configurations, and the node's input defined as $(I \bowtie II) \bigcup (III \bowtie V) \bigcup (IV \bowtie V)$.

We already discussed that for a node's instantiation there must exist some input for each node's input slot. While it is a necessary condition, it is not satisfactory, as there are combinations of inputs for which it is impossible to perform a provenance-aware join. We present an example in Figure 4.6.(d).

In this case, there is some input for each input slot. However, the inputs coming from node I and II partially overlap, as both these nodes define some input for input slot 2. While such a case could be handled by mapping it onto a relational join operator, it would significantly augment the complexity of the multiple algorithms applied in our model. Thus, in our model the overlaps of input mappings belonging to a single input configuration are disallowed.

We make an assumption that for each node all its input configurations need to be valid, and all input mappings need to belong to a valid input configuration. Thus, even if we completed the case presented in (d) by adding an input mapping coming to slot 3, as presented in (e), the node still would not have valid inputs, as input mapping from node III would not belong to a valid input configuration.

As it is possible that a few input configurations exist for a single node, different relations that may exist between them. The most simple case is when multiple combinations of input mappings provide input to the same subsets of a node's input slots. For example, this is the case of input configurations (I, II) and (I, III) in Figure 4.6.(a). We will further call such input configurations *equivalent* with respect to their destination slots.

The second possibility is that for two non-equivalent input slots, all sets of destination slots of input mappings from one configuration are equal to or completely included in the destination slots of input mapping of another configurations. For example, if the input mappings presented in Figure 4.5 (c) and (d) existed in the same node, sets (1, 2) and (3) corresponding to situation (d) would be completely included in set (1, 2, 3) corresponding to situation (c). In this case, we would call the configuration from example (d) a *refinement* of the one from example (c).

Finally, if for two non-equivalent input configurations neither is refinement of the other (e.g. (I, II) and (III, IV) in Figure 4.6.(b)) they are independent.

Formalization

As we stated before, input mappings that form an input configuration need to be pairwise disjoint with respect to their destination slots and need to cover all input slots of the destination node. We formalize these rules by the following definition.

Definition 4.3.11 Input Configuration
A set of input n input mappings $isc = (\lambda_1, \lambda_2, \ldots, \lambda_n)$, such that $d(\lambda_1) = d(\lambda_2) = \cdots = d(\lambda_n) = \gamma$ is called the input configuration of node γ if and only if $(ds(\lambda_1), ds(\lambda_2), \ldots, ds(\lambda_n))$ is a partition of $is(\gamma)$, i.e.:

- $\forall_{i \in (1,\ldots,n)} ds(\lambda_i) \neq \emptyset$
- $\bigcup_{i \in (1,\ldots,n)} ds(\lambda_i) = is(\gamma)$
- $\forall_{i,j \in (1,\ldots,n), i \neq j} ds\lambda_i \cap ds\lambda_j = \emptyset$

The destination node of all input mappings included in input configuration isc will be denoted by $d(isc) = \gamma$, and the union of their destination slots will be denoted by $ds(isc) = \bigcup_{i \in (1,\ldots,n)} ds(\lambda_i)$.

If a node has more than one input slot, there exist multiple distinct partitions of a set of its inputs slots. Consequently, as demonstrated previously, there may exist multiple input configurations corresponding to different partitions of node input slots. Given a pair of input configurations isc_1, isc_2 of the same node γ, either one of them is a *refinement* of the other one with respect to their destination slots, wither they are *independent* or they are *equivalent* with respect to their destinations slots. These three possibilities are covered by the following definitions.

Definition 4.3.12 Refinement of Input Configuration
isc_2 is refinement of isc_1 with respect to their destinations slots, if and only if the set of destination slots of each input mapping in isc_2 is a subset of the destination slots of some input mapping in isc_1 and the sets of destination slots of input mappings in both configurations are not identical. More formally, isc_2 is refinement of isc_1 (written $isc_2 \leq isc_1$) if and only if $\forall_{\lambda_2 \in isc_2} \exists_{\lambda_1 \in isc_1} ds(\lambda_2) \subseteq ds(\lambda_1) \wedge \exists_{\lambda_2 \in isc_2} \exists_{\lambda_1 \in isc_1} ds(\lambda_2) \subset ds(\lambda_1)$

Definition 4.3.13 Independent Input Configurations
Two input configurations isc_1 and isc_2 are independent with respect to their destination slots (further denoted by $isc_1 \perp isc_2$) if and only if $\neg(isc_1 \leq isc_2 \vee isc_2 \leq isc_1)$.

Definition 4.3.14 Equivalent Input Configurations
Two configurations $isc_1 = (\lambda_{1_1}, \ldots, \lambda_{n_1})$ and $isc_2 = (\lambda_{1_2}, \ldots, \lambda_{n_2})$ are equivalent with respect to their destination slots if and only if $(ds(\lambda_{1_1}), \ldots, ds(\lambda_{n_1})) = (ds(\lambda_{1_2}), \ldots, ds(\lambda_{n_2}))$.

Example 4.3.7 Relation Between Multiple Input Configurations
The set of input slots of node γ is defined as $is(\gamma) = (a, b, c, d)$. Let:
- $ds(\lambda_1) = (a, b), d(\lambda_1) = \gamma$
- $ds(\lambda_2) = (c, d), d(\lambda_2) = \gamma$
- $ds(\lambda_3) = (a), d(\lambda_3) = \gamma$
- $ds(\lambda_4) = (b), d(\lambda_4) = \gamma$
- $ds(\lambda_5) = (a, c), d(\lambda_5) = \gamma$
- $ds(\lambda_6) = (b, d), d(\lambda_6) = \gamma$

Then there are three valid input configurations of node γ, namely:
- $isc_1 = (\lambda_1, \lambda_2)$
- $isc_2 = (\lambda_3, \lambda_4, \lambda_2)$
- $isc_3 = (\lambda_5, \lambda_6)$

The relations between pairs of these configurations are as follows:
- $isc_2 \leq isc_1$ as two sets in isc_2 are proper subsets of a set in isc_1 ($\lambda_3 \subset \lambda_1, \lambda_4 \subset \lambda_1$) and the remaining sets in isc_1 and isc_2 are equal;

- $isc_3 \perp isc_1 \wedge isc_3 \perp isc_2$ as none of refinements $isc_3 \leq isc_1$, $isc_1 \leq isc_3$, $isc_2 \leq isc_3$, $isc_3 \leq isc_2$ exist

In this example, the complete input provided to node γ will be calculated as follows: $(\lambda_1 \boxtimes \lambda_2) \bigcup (\lambda_3 \boxtimes \lambda_4 \boxtimes \lambda_2) \bigcup (\lambda_5 \boxtimes \lambda_6)$.

Example 4.3.8 Equivalent Input Configurations

Assuming that γ and $\lambda_1, \ldots, \lambda_6$ are defined as in example 4.3.7, let $ds(\lambda_7) = (1, 2) \wedge d(\lambda_7) = \gamma \wedge s(\lambda_1) \neq s(\lambda_7)$. Then the input configurations $isc_1 = (\lambda_1, \lambda_2)$ and $isc_2 = (\lambda_2, \lambda_7)$ are equivalent.

As mentioned previously, the notion of input configuration enables us to define valid input mappings.

Definition 4.3.15 Valid Input Mapping
An input mapping λ is valid if and only if $\exists_{isc} isc$ is a valid input configuration of node $d(\lambda) \wedge \lambda \in isc$.

4.3.5 Executable Data Extraction Graphs

The data extraction graph is a very flexible abstraction that can be used for the data-centric modeling of both complete Web site navigation structure (i.e. all types of navigation that are possible in a Web site), and navigation and extraction structure corresponding to specific user queries. Some data extraction graphs are executable, i.e. they can be used to extract data from a Web site without need of any additional input. However, some of them, including, for example, graphs modeling Web sites with open-domain forms, do not have complete input configurations for some nodes (i.e. they require some user input to be executed), thus being non-executable.

Both executable and non-executable graphs play an important role in our model. An executable graph can be simply used to obtain data from a Web site, thus being the basic data extraction tool. Therefore, all user query plans are executable and capable of extracting data. A non-executable graph can be used to describe Web sites in a general and concise way, and can be combined with a user query in order to build an executable plan for a query[2].

In order to define the notion of Web graph executability, we start by clarifying the notion of executable node, using the following recursive definition.

Definition 4.3.16 Executable Data Extraction Node
A node $\gamma \in N$ is executable if and only if it is trivial, or it has at least one complete input configuration such that the source nodes of all input mappings contained in this input configuration are executable.

[2]We discuss the details of query plans and the query planning process in Section 4.6.

Intuitively a graph is executable if it is possible to traverse it in at least one way from at least one trivial node (*start node*) to at least one node collecting data (*end node* e.g. an attribute set) by visiting only executable nodes. More formally:

Definition 4.3.17 Executable Data Extraction Graph
A data extraction graph D is executable if and only if:

- it contains at least one trivial (inputless) node called *start node(s)*,
- it contains at least one node collecting extracted data (e.g. attribute set) called *end node(s)*, and
- there exists at least one path in the graph that starts at a start node, ends on an end node and contains only executable nodes.

It is to be noted that while all query plans are executable and some Web site models are not, there are also Web site models that are executable (e.g. using only form-less navigation). The execution of such a Web site model can be treated as performing a "get all data" query plan.

The second important observation is that the term graph executability refers only to graph structures, but does not respond to the question of whether or not data extraction will return any data. An executable graph may still return zero results if the Web site contains no corresponding data or if there is a semantic mistake in graph definition (e.g. output corresponding to car model is used as input to the "make" field of an HTML form).

Finally, even if a graph is executable, it is possible that some part of it is not. For example, if a Web site model contains both link-based and form-based navigation it is possible that it is executable, but only in the part based on navigation. This leads us to the notions of *maximal executable subgraph* and *completely executable graph.*

Definition 4.3.18 Maximal Executable Subgraph of Data Extraction Graph
The *maximal executable subgraph* of an executable data extraction graph D, denoted by $Max(D)$, is the largest graph formed by nodes $N_{Max} \subset N$ such that

- all $n \in N_{Max}$ are executable,
- each of $n \in N_{Max}$ belongs to at least one path connecting the start and end node,

and by all input mappings $IM_{Max} \subset IM$ such that $\forall_{\lambda \in IM_{Max}} s(\lambda) \in N_{Max} \wedge d(\lambda) \in N_{Max}$.

Definition 4.3.19 Completely Executable Data Extraction Graph
A data extraction graph D is completely executable if $Max(D) = D$.

Intuitively, the graph is completely executable if it is executable and does not contain any unnecessary nodes or edges.

4.3.6 Graph Inputs and Outputs

Until now we discussed how the data extraction graph can be used to model internal Web site navigation and data organization. We also discussed how the flow between different types of nodes can be represented. We also stated that while some graphs describing a Web site may be executable, they often require additional input that comes from users or external systems. At the same time, the data extraction process is useful if it generates data that can be presented to the user or consumed by other applications. Thus, the way of representing the input and output of the data extraction process is essential.

To complete our graph model we introduce two special categories of nodes: *input nodes* and *output nodes*. Input nodes are responsible for introducing external input to a given data extraction graph. While input nodes need to obtain input from some external source, the data extraction graph model does not define where the input comes from. Thus, we may have input nodes that store data internally (i.e. their definition contains values for specific attributes), as well as nodes that obtain input from a user or from an external database. However, from the perspective of the data extraction graph they are all trivial nodes.

Output nodes are all types of nodes that gather some extracted data in a form that can be presented to the user or sent to an external system. We will consider as extraction nodes, for example, all attribute sets. However, other types of output nodes are possible, such as nodes that automatically save extracted data in a specific file format or insert them to a database.

While input nodes, to make them executable, cannot have input slots, output slots in the case of output nodes are allowed. Thanks to this property, output nodes can be used not only as end nodes on navigation paths (which is the most typical case) but also “in the middle” of navigation paths. Thanks to this, the processing of partial data (e.g. with attributes not yet extracted), useful especially in the case of the *ad hoc* querying of a large source, is possible.

Together, input and output nodes are the basic interface between our graph model and external systems. They are also the basic notion for the process of query planning.

4.3.7 Cyclic Data Extraction Graphs

Until now all discussed examples of data extraction graphs focused on acyclic graphs. Acyclic graphs cover many important navigation and data organization patterns in Web sites, such as different types of linear and hierarchical data extraction patterns. However, cyclic graphs play an essential role in modeling more complex Web sites.

The first situation that needs to be modeled by a cyclic graph concerns pagination, which is a typical mechanism in data intensive Web sites. Typically, Web actions corresponding to visiting consecutive result pages are similar enough to be modeled by the same Web action template. Moreover, multiple result pages have the same structure. Therefore, the same data extraction rules can be applied in all of them. Finally, the number of result pages is not known in advance. For all these reasons, pagination can be only modeled by using cyclic graphs. We present a sample cycle data extraction graph in Figure 4.7, and discuss this in the following example.

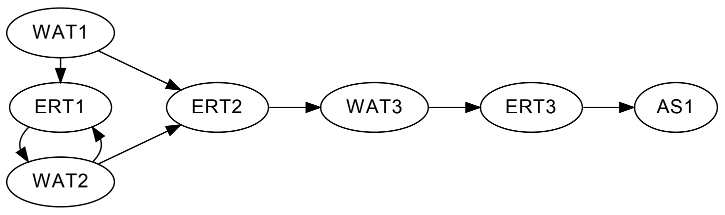


Figure 4.7: Example of Graph Width Cycles

Example 4.3.9 Using a Cyclic Graph for Pagination Modeling

Let's consider the data extraction graph depicted in Figure 4.7. The modeled Web site consists of a search form (WAT1 corresponds to filling in the from) that generates multiple result pages. At each result page, two extraction rules are executed: ERT1 extracts an URL of the next result page, while ERT2 extracts an URL of the details page for each item in the search results.

The output of the ERT1 rule is used to instantiate the WAT2 Web action, which consists in navigating to the next result page. From this page, ERT1 and ERT2 are executed again. Thus, ERT1 and WAT2 form a cycle corresponding to pagination.

The output of ERT2 is used to instantiate WAT3, which corresponds to navigation to the detail page of a specific item. All details of this item are extracted from the detail page by applying ERT3 and sendinf the extraction result to the AS1 output node.

Similarly, cyclic graphs can be used for any other navigation patterns based on next and back buttons. For example, it may be useful to work with many photo galleries, where after choosing the first photo the only option to move to the next one is to use "next" link, as well as in the case of data tables or news stories that were chopped into a few smaller pages.

Another powerful application of cyclic graphs, combined with constructor output slots, concerns hierarchical data structures of unknown depth. Among examples of such situations, there are tree-based representations of a folder structure (e.g. of an FTP server), as well as general-purpose Web site directories with a varying number of levels of subcategories.

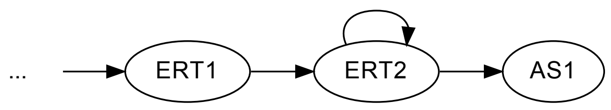


Figure 4.8: Example of Using a Cyclic Graph for the Extraction of Hierarchies of Unknown Depth

Example 4.3.10 Using a Cyclic Graph to Extract Folders Tree of Unknown Depth

Let's assume that some part of the data extraction graph presented in Figure 4.8 is used to extract the names of folders from a page that contains a hierarchy of folders of unknown depth. Let's assume that ERT1 is an XPath expression that extracts location in a document corresponding to the root folder and that ERT2 finds all direct child folders of the folder passed to one of its input slots, and returns the extracted name through the `innerText` output slot.

Let's also assume that apart from an input slot corresponding to the DOM element and automatic output slots, ERT2 has also an input mapping corresponding to the path extracted so far, called `path`, and a constructor output slot `currentPath` defined as follows: `{path}/{innerText}`.

Input mapping from ERT1 to ERT2 is defined so that the DOM element corresponding to the root folder is mapped onto the `!element` input slot and the value of a constant constructor output slot containing an empty string for input slot `path`. In the case of cyclic input mapping from ERT2 to ERT2, the extracted element is mapped onto the `!element` input slot, and the `currentPath` output slot is mapped onto the `path` input slot. Finally, ERT2 `innerText` and `currentPath` are mapped onto the `folderName` and `path` input slots of AS1.

In such a case, execution of the data extraction path will iteratively extract the names of folders and will construct their path by appending the folder name to the path passed from its parent folder.

A similar combination of cycles and constructor output slots can be used to extract location in a directory hierarchy of specific Web pages. However, in this case, apart from ERT having a very similar constructor output slot, as in the previous example, also the Web action should have a constructor output slot that returns an unaltered value for its input slot corresponding to path location within the directory of the parent page from which we arrived.

It is to be noted that all graph-based data extraction algorithms presented in the next section are capable of working with cyclic data extraction graphs.

4.4 Data Extraction Graph Execution

In the previous sections we presented how different aspects of Web navigation and information extraction can be described by using data extraction graphs. In this section we present basic, extensible algorithms that perform actual information extraction by using executable data extraction graphs.

4.4.1 Intuitive Overview

While discussing the notions of data extraction node, instance and atom (see Section 4.3.1), we introduced three operations: node instantiation, atom creation and atom execution. Next, in Section 4.3.4 describing input configurations, we mentioned (without defining) provenance-aware join as a method of merging input data coming from different data extraction nodes. In Section 4.3.5, we also discussed conditions that must be met for a graph to be executable.

In this section we introduce a data extraction algorithm that repetitively performs node instantiation, atom creation, atom execution and provenance-aware join based on the structure of an executable data extraction graph. We start by demonstrating the algorithm on an example, and then we present it in a formal way.

To present intuitively how graph execution is performed, we will use the graph presented in Figure 4.4 and described in Example 4.3.4. We follow the steps that should be done during graph execution in the following example.

Example 4.4.1 Execution of Data Extraction Graph

In Figure 4.4 WAT1 is a trivial Web action template that has no input slots and that always returns a single Web action corresponding to the navigation of Web site's home page. After this Web action is executed, ERT1 accepts as the input the current contents of the Web browser. As ERT1 receives a single input value to its only input slot, instantiation will create a single extraction rule that will be combined with a Web browser's content to form an executable atom. The result of executing this extraction rule consists of three records (corresponding to three car makes) with two attributes each (make name and URL of make page). Make names are passed directly to the result AS1 attribute set, while URLs, passed to an input slot of WAT2, are used to create three Web actions that correspond to navigating to each of the make pages.

Extraction at each make page is performed in a similar way as in case of the home pages. For each of the three input "content" values (corresponding to three make pages) ERT2 generates and executes three extraction rules that extract two, five and five items respectively, corresponding to the model name and URL of the model page. A total of 12 different URLs are passed to WAT3 and enable the generation of 12 Web actions, corresponding to navigating to each of the model pages.

At each model page a composition of two different extraction rule templates is used. Firstly, XPath (ERT3 defined as follows: `//li`) is applied on browser content to extract individual list positions. Secondly, a regular expression with subpatterns (instantiated by ERT4) is used to extract the attributes of each position: namely engine displacement (e.g. 1.2), engine type (e.g. 8V), engine power (e.g. 69), gearbox type (e.g. M/5) and version name (e.g. POP). All these five attributes for all extracted data are passed to attribute set AS1, where they are combined with the previously extracted make and model name.

This example leads us to the following assumptions for the graph execution algorithm. Firstly, we assume that a node is visited only if there is some ready input for it. We also assume that the output of preceding nodes is never changed or replaced. Thus, the input data of a node can be consumed at the moment of input reception.

Secondly, even if previous output cannot be modified, additional inputs to the node can be received from other sources (input mappings coming to other input configurations) or from the same source (if the preceding node receives and consumes multiple input). A node can also receive additional input from the same source if the graph is cyclic, and the next iteration of a cycle was executed.

Therefore, the node in a graph is often visited multiple times. The minimal number of visits to a node is equal to the size of a minimal subset of input mappings used in a node's input configurations, such that each input configuration contains at least one input mapping from this subset.

Finally, for now we assume that the Web site is stateless, and the order of execution of nodes does not have any impact on the results of data extraction. This assumption will be removed in the state-aware version of our algorithm discussed in Section 4.5.

4.4.2 Graph Analysis

Before a data extraction graph can be used for data extraction, it needs to be analyzed. Specifically, four tasks need to be performed:

- Identification of the start and end nodes,
- Identification and validation of the input configurations of all graph nodes,
- Identification of incomplete input configurations, i.e. configurations that need some extra input to enable node execution,
- Verification if a graph is executable and if it is fully executable.

Below we present an algorithm that performs these four tasks. Moreover it initializes the following data structures:

- Configs - the collection that assigns each node the complete list of its complete input configurations,
- IConfigs - the collection that assigns each node the list of all its incomplete input configurations (including configurations including zero input mapping),
- MConfigs - the collection that assigns each input mapping the list of input configurations that contain this input mapping,
- StartNodes - the complete set of start nodes that will be used during graph execution,
- EndNodes - the set of data extraction nodes that will be accessed during graph execution,
- OutputNodes - the set of data extraction nodes used for collecting data, and
- NINodes - the set of all nodes that cannot be instantiated without extra input.

Algorithm 4.4.1 $\textsc{AnalyzeNodeRecursively}(\gamma, IM^\gamma, startIndex, candidate)$

```
Require: Node $\gamma$, list of input mappings coming into node $IM^\gamma$, start index of unprocessed
    input mappings $startIndex$, current candidate input configuration $candidate$.
Ensure: Updated $Configs_\gamma$ item.
1:  for $i \leftarrow startIndex$ to $|IM^\gamma|$ do
2:    if $ds(IM_i^\gamma) \bigcap ds(candidate) = \emptyset$ then
3:      $candidate \xleftarrow{add} IM_i^\gamma$
4:      if $|ds(candidate)| = |is(\gamma)|$ then
5:        (Save configuration for $\gamma$.) $Configs_\gamma \xleftarrow{add} candidate$
6:        for all $\lambda \in candidate$ do
7:          (Save configuration for $\lambda$.) $MConfigs_\lambda \xleftarrow{add} candidate$
8:      else
9:        (Save incomplete configuration for $\gamma$.) $IConfigs_\gamma \xleftarrow{add} candidate$
10:       (Continue recursively.) $\textsc{AnalyzeNodeRecursively}(\gamma, IM^\gamma, i+1, candidate)$
```

The algorithm makes an assumption that the data extraction graph that it receives as argument is connected. For each node, it recursively searches the space of all possible input mappings coming to the node, cutting all search tree branches that would lead to input configurations with overlapping destination slots of input mappings. Whenever a complete input configuration is found it is added to Configs and IConfigs collections. All incomplete input configurations for a given node are stored in the MConfigs collection.

At the same time, the algorithm analyzes node type (to identify OutputNodes), the number of a node's input slots (node is added to StartNodes if it is trivial) and the number of outgoing output slots (it is added to EndNodes if there are none).

Finally, it also verifies if a graph is fully executable (i.e. all its nodes have at least one complete input configuration). If it is not, it further analyzes if it is at least executable.

The algorithm consists of two parts. The first of them (Algorithm 4.4.1), which is executed recursively, is described below.

Algorithm 4.4.2 ANALYZEGRAPH(D)

Require: Connected data extraction graph D.

Ensure: Updated configurations and nodes collections and executable subgraph of $Max(D)$.

```
1: (Initialize configurations collections.) $MConfigs, Configs, IConfigs \leftarrow ()$
2: (Initialize nodes collections.) $StartNodes, EndNodes, OutputNodes, NINodes \leftarrow ()$
3: for all $\gamma \in N(D)$ do
4:     (Set of incoming input mappings.) $IM^{\gamma} \leftarrow \bigcup_{\lambda \in IM(D) \wedge d(\lambda)=\gamma} \lambda$
5:     (Set of outgoing input mappings.) $OM^{\gamma} \leftarrow \bigcup_{\lambda \in IM(D) \wedge s(\lambda)=\gamma} \lambda$
6:     if $|IM^{\gamma}| > 0$ then
7:         (Analyze node $\gamma$.) ANALYZENODERECURSIVELY($\gamma, IM^{\gamma}, 1, ()$)
8:     else
9:         (Add to start nodes.) $StartNodes \xleftarrow{add} \gamma$
10:    if $|OM^{\gamma}| = 0$ then
11:        (Add to end nodes.) $EndNodes \xleftarrow{add} \gamma$
12:    if $\gamma$ is a data collection node then
13:        (Add to data collection nodes.) $OutputNodes \xleftarrow{add} \gamma$
14:    if $|Configs_{\gamma}| = 0$ then
15:        (Save node as non-instantiable.) $NINodes \xleftarrow{add} \gamma$
16: if $|NINodes| = 0$ then
17:     (Completely executable graph.) $Max(D) \leftarrow D$
18: else
19:     (Find executable subgraph.) $Max(D) \leftarrow$ FINDEXECUTABLESUBGRAPH($D$)
```

The algorithm consists of a single loop (lines 2–9) that is repeated for all indexes of input mappings starting in *startIndex*. If input mapping in the current index i is disjoint with the current candidate input configuration *candidate* with respect to their destination slots (line 2), it is added to the candidate input configuration (line 3). If the destination slots of candidate input configuration cover all the input slots of the destination node (or, equivalently, the number of items in both these sets of slots is equal) (line 4), it is added to output collections (lines 5–7). Otherwise, we store the incomplete configuration in the $IConfigs$ collection (line 9) and recursively try to extend the candidate input configuration with any input mappings with an index larger than *startIndex* (line 10).

The recursive algorithm described above is used within the main graph analysis algorithm (Algorithm 4.4.2).

Algorithm 4.4.3 FINDEXECUTABLESUBGRAPH(D)

```
Require: Connected data extraction graph $D$.
Ensure: Returns $Max(D)$ or empty graph in case if $D$ is not executable.
1: (Initialize configurations collections.) $MaxMConfigs \leftarrow MConfigs, MaxConfigs \leftarrow Configs$
2: (Initialize nodes collections.) $MaxStartNodes \leftarrow StartNodes, MaxEndNodes \leftarrow EndNodes, MaxOutputNodes \leftarrow OutputNodes$
3: (Initialize graph nodes and input mappings.) $MaxN \leftarrow N(D), MaxIM \leftarrow IM(D)$
4: (Initialize index of current node.) $i \leftarrow 1$
5: while $i \leq |NINodes|$ do
6:   (Get current node.) $\gamma \leftarrow NINodes_i$
7:   (Remove from $MaxStartNodes$.) $MaxStartNodes \leftarrow MaxStartNodes \setminus (\gamma)$
8:   (Remove from $MaxEndNodes$.) $MaxEndNodes \leftarrow MaxEndNodes \setminus (\gamma)$
9:   (Remove from $MaxOutputNodes$.) $MaxOutputNodes \leftarrow MaxOutputNodes \setminus (\gamma)$
10:  (Remove from $MaxN$.) $MaxN \leftarrow MaxN \setminus (\gamma)$
11:  for all $\lambda \in MaxIM$ such that $d(\lambda) = \gamma \vee s(\lambda) = \gamma$ do
12:    (Remove from $MaxIM$.) $MaxIM \leftarrow MaxIM \setminus (\lambda)$
13:    for all $ic \in MaxMConfigs_\lambda$ do
14:      (Remove from $MaxConfigs$.) $MaxConfigs_{d(\lambda)} \leftarrow MaxConfigs_{d(\lambda)} \setminus (ic)$
15:    (Remove from $MaxMConfigs$.) $MaxMConfigs_\lambda \leftarrow ()$
16:    if $|MaxConfigs_{d(\lambda)}| = 0 \wedge d(\lambda) \notin NINodes$ then
17:      (Schedule node for removal.) $NINodes \xleftarrow{add} d(\lambda)$
18:    if $\neg\exists_{im \in MaxIM} s(im) = s(\lambda)$ and $s(im) \in OutputNodes$ and $s(\lambda) \notin NINodes$ then
19:      (Schedule node for removal.) $NINodes \xleftarrow{add} s(\lambda)$
20:  (Increment $i$.) $i \leftarrow i + 1$
21: if $|MaxStartNodes| = 0$ then
22:   return ()
23: (Initialize list of nodes accessible from $MaxStartNodes$.) $ANodes \leftarrow MaxStartNodes$
24: (Initialize $i$.) $i \leftarrow 1$
25: while $i \leq |ANodes|$ do
26:   for all $\lambda \in MaxIM$ such that $s(\lambda) = ANodes_i$ do
27:     (Add destination node to $ANodes$.) $ANodes \xleftarrow{add} d(\lambda)$
28: if $|ANodes| = |MaxN|$ then
29:   (Build maximal subgraph.) $Max(D) \leftarrow (MaxN, MaxIM)$
30: else
31:   (No executable subgraph exists.) $Max(D) \leftarrow ()$
32: return Max(D)
```

The main part of the algorithm consists of a loop (lines 4–15) that is repeated for all the nodes of graph D. For all the nodes that have any incoming input mappings (line 6), the ANALYZENODERECURSIVELY algorithm is run, starting at index 1 with an empty candidate input configuration. Moreover, all nodes that have zero incoming input mappings are added to the $StartNodes$ collection (line 9), nodes that have no outgoing input mappings are added to the $EndNodes$ collection (line 11), and all nodes that are used for collecting data (such as attribute sets) are added to the $OutputNodes$ collection (line 13). Finally, all nodes that have no complete input configuration (line 16) are stored in the $NINodes$ collection (line 17).

The final part of the algorithm identifies the maximal executable subgraph of graph D. In cases where all nodes have at least one complete input configuration (line 16), $Max(D)$ corresponds to the graph itself (line 17) and the graph is completely executable. In other cases, the FindExecutableSubgraph subroutine is used to identify $Max(D)$ (line 19).

FindExecutableSubgraph (Algorithm 4.4.3) has three distinct parts. The first (lines 1–4) concerns the initialization of data structures by copying values corresponding to graph D. The second (lines 6–20) removes all nodes and input mappings that do not belong to $Max(D)$. Finally, the third part (lines 21–32) verifies if the minimal subgraph contains any start nodes and if it is connected.

The main loop of the algorithm (lines 6–20) is executed as long as there are any non-instantiable node γ that need to be analyzed. It starts by removing the node from all node collections corresponding to $Max(D)$ graph (lines 7–10).

Next, an internal loop (lines 12–19) is performed for each input mapping λ that starts or ends in γ and has not been removed yet. It removes the input mapping from the $MaxIM$ collection of input mappings of $D(IM)$ (line 12) as well as all input configurations that contain λ from the list of configurations for its destination node $MaxConfigs_{d(\lambda)}$ (line 14) and from the list of all λ input configurations $MaxMConfigs_{\lambda}$ (line 15). If after these operations $d(\lambda)$ has no input configuration left (line 16), it is added to the end of the $NINodes$ collection (line 17) and will be removed afterwards. If $s(\lambda)$ no longer provides output for any other node and is not an output node (line 18), it is also added to $NINodes$ (line 19).

After all non-instantiable nodes and their input mappings are removed, it is necessary to check if the resulting graph contains any start nodes (lines 21–22) and is connected. Connectedness is checked by finding all nodes that can be accessed from $MaxStartNodes$ (lines 25–27) and verifying if this set is identical with the set of all nodes $MaxN$ (lines 28–31).

The output of the algorithm consists of the maximal executable subgraph (if it exists) or of an empty graph (if no maximal executable graph exists).

4.4.3 Graph Execution

The graph execution algorithm is the essential part of our data extraction and Web site navigation method. It consists in using the data extraction graph to extract all relevant data from a specific Web source(s). To attain this objective it repetitively performs five key tasks: the instantiation of ready nodes (i.e. nodes that have received some new complete input), atom creation and queuing for future execution, current atom execution, and the propagation of atom execution outputs to all dependent nodes.

The formal algorithm that follows is presented in a top-down way. In this section, we present a generalized, modular algorithm that hides many details in a number of subroutines corresponding to five mentioned key tasks. Than, in the next sections, we provide details of all these subroutines, and discuss their variants (if applicable).

The algorithm accepts as input a completely executable data extraction graph D[3]. Its result consists of the collection *ReturnData*, which assigns each output node accessed during data extraction the set of result objects generated by the execution of its atoms.

Algorithm 4.4.4 ExecuteGraph(D)

Require: Completely executable data extraction graph D.
Ensure: *ReturnData* collection.

```
 1: $StartNodes, EndNodes, OutputNodes, D \leftarrow$ AnalyzeGraph($D$)
 2: for all $n \in StartNodes$ do
 3:   $instance \leftarrow$ Instantiate($n$, ())
 4:   $atom \leftarrow$ CreateAtom($instance$, ())
 5:   Enqueue($atom$)
 6: for all $n \in OutputNodes$ do
 7:   (Initialize $n$ return data with empty lists.) $ReturnData_n \leftarrow ()$
 8: while not QueueEmpty() do
 9:   (Get current atom and remove it from queue.) $atom \leftarrow$ PopQueue()
10:   (Execute current atom.) $atomOutput \leftarrow$ Execute($atom$)
11:   (Node that created atom.) $c \leftarrow$ CreatorNode(CreatorInstance($atom$))
12:   if $c \in OutputNodes$ then
13:     (Save data.) $ReturnData_c \xleftarrow{add} atomOutput$
14:   for all $\lambda$ such that $s(\lambda) = c$ do
15:     (Save data.) StoreData($\lambda, Data$)
16:     (Get $\lambda$ destination node.) $n \leftarrow d(\lambda)$
17:     if NodeReady($n$) then
18:       (Get node's input data.) $inputRecords \leftarrow$NewInputRecords($n$).
19:       for all $r \in inputRecords$ do
20:         (Get node instance.) $instance \leftarrow$ Instantiate($n, r$)
21:         (Create atom.) $atom \leftarrow$ CreateAtom($instance, r$)
22:         (Add atom to queue.) Enqueue($atom$)
```

The few first steps of the algorithm consist in initializing data structures: *StartNodes*, *EndNodes* and *OutputNodes* by using the AnalyzeGraph algorithm (line 1), the queue of atoms to be executed by applying instantiation and atom creation tasks on all start nodes (lines 2–5), and *ReturnData* collection (lines 6–7).

The main part of the algorithm consists of a loop that is repeated while there are ready-to-execute atoms in an execution queue (lines 9–22). At each iteration the current atom is received and deleted from the queue (line 9) and consecutively executed (line 10). Next, the node c used to create an instance that was used to create the *atom* is found (line 11). If it is one of the output nodes (line 12), the atom's output records are added to *ReturnData* for node c (line 13).

[3]While the algorithm is capable of producing valid and complete output also in case of executable graphs that are not completely executable, it does not contain any branch-cutting mechanism. Thus, executing such graphs leads to useless node instantiations and atom executions that are not used to generate extraction output.

The next part of the algorithm, which is responsible for atom outputs propagation (lines 15–22), is executed separately for each input mapping λ outgoing from the node used for the creation of the current atom. After the current input for λ is stored for future reference (line 15), its destination node n is analyzed. If n has all the inputs for at least one of its input configurations (line 17), a set of corresponding input records is collected (line 18), and each of them is used for node instantiation (line 20) and atom creation (lines 21). Finally, the created atoms are added to the queue (line 22).

There are a few aspects of the presented algorithm that require additional explanation. Firstly, as we discussed previously, node instantiation and atom creation are separate tasks, and in some cases it may be possible instantiate the node even if not all the inputs needed for atom creation are already obtained. However, in the proposed algorithm we assume that nodes should be instantiated only if they have received some input for both the node instantiation and atom creation input slots (i.e. if the condition in line 17 is met). There are three reasons for this choice.

Firstly, as node instantiation does not provide any data to other nodes, performing node instantiation and atom creation together has no impact on data received from the data extraction process. Secondly, joining these two tasks significantly simplifies queue management, as there is no need to separately manage pools of instances that are and are not parts of atoms. Finally, deferring node instantiations ensures that the situation when a node is instantiated but the instance never obtains the inputs necessary for atom creation never happens, thus improving the algorithm's performance.

It is important to note that the steps of node instantiation and atom creation (lines 18–22) can be repeated for the same node multiple times if the node has more than one input configuration or if the graph contains loops (see Section 4.3.7). In such a situation, both processes should be performed not for all stored inputs, but only for new combinations of data stored for individual input mappings. This behavior, assured by the StoreData and NewInputRecords algorithms, is further discussed in Section 4.4.7.

Another property of the proposed algorithm is that many details of its behavior depend on the implementation of its subroutines. We distinguish among them three groups. The first of them includes the basic subroutines such as Instantiate, CreateAtom and Execute. This group is discussed in Section 4.4.4.

Two other groups are related to two aspects of query executions state stored at a given moment of time (the algorithm execution stage), i.e. the queue of ready-to-execute atoms and data inputs propagated between nodes. Queue management algorithms, i.e. Enqueue and PopQueue, are further discussed in Section 4.4.5. The discussion of data and provenance management algorithms (StoreData, NewInputRecords and NodeReady) provided in Section 4.4.7 is preceded by the introduction of the StoreData essential component, i.e. the provenance-aware join, in Section 4.4.6.

4.4.4 Basic Subroutines

There are three basic subroutines of the data extraction graph execution algorithm: Instantiate, CreateAtom and Execute. These three types of subroutines correspond to operations described in Section 4.3.1, and demonstrated in Examples 4.3.1, 4.3.2 and 4.3.3.

A separate node instantiation and atom creation behavior is defined for each node and instance type, respectively. Thus, the node to be instantiated and the instance to be used for atom creation are the first arguments to the INSTANTIATE and CREATEATOM algorithms. The second argument passed to both algorithms is a data record that contains a single value for each of the node's inputs used for node instantiation and atom creation.

It is to be noted that multiple records r may be identical with respect to their node instantiations' input values. Thus, for better performance, the INSTANTIATE algorithm may use a dynamic programming approach to avoid records' recreation. However, implementation of this idea belongs to our future research plans.

4.4.5 Queue Management

Queue management has three main responsibilities. Firstly, it stores all created atoms that are ready to be executed in some internal data structure within the ENQUEUE subroutine. While a simple queue or list is the most intuitive representation, in the case of more complex queue management approaches, structures that somehow order or index atoms may be used. Two basic implementations are presented. Other examples include the use of an ordering algorithm that minimizes the need for navigation state switching (see Section 4.5.5), and the use of more complex data structures for distributed execution (see Section 6.5).

Secondly, it implements a node selection method capable of choosing at any time a single atom out of all atoms stored. This task is a part of the POPQUEUE subroutine, which returns the chosen atom and removes it from its internal queue representation. The simplest method returns the first element of the list, but more complex methods may be implemented in which the returned value depends on the time elapsed from the previous HTTP request or on the identifier of EXECUTEGRAPH algorithm instance (in the case of parallel or distributed execution).

Finally, queue management provides a subroutine QUEUEEMPTY that informs if there are no atoms to execute in the queue.

Algorithm 4.4.5 ENQUEUEBFS(a)

```
Require: Atom to enqueue $a$.
Ensure: $a$ is added to pre-initialized list of items $Queue$.
1: (Get number of $Queue$ items.) $c \leftarrow |Queue|$
2: (Add $a$ to the $Queue$ at the end.) $Queue \leftarrow (Queue_1, \dots, Queue_c, a)$
```

The possibility of using different queue management approaches is one of the key elements of EXECUTEGRAPH flexibility. Below we present two simple implementations of ENQUEUE that make EXECUTEGRAPH correspond to breadth-first (Algorithm 4.4.5) and depth-first (Algorithm 4.4.6) graph search, respectively.

Algorithm 4.4.6 EnqueueDFS(a)

```
Require: Atom to enqueue $a$.
Ensure: $a$ is added to pre-initialized list of items $Queue$.
1: (Get number of $Queue$ items.) $c \leftarrow |Queue|$
2: (Add $a$ to the $Queue$ at the beginning.) $Queue \leftarrow (a, Queue_1, \dots, Queue_c)$
```

The key difference between these implementations is in line 2. While EnqueueBFS adds current atom a at the end of queue, EnqueueDFS inserts it at the beginning.

The following algorithm represents the simplest implementation of PopQueue that can be used with any implementation of Enqueue that stores atoms in a single list called $Queue$.

Algorithm 4.4.7 PopQueueBasic()

```
Require: Filled in list of atoms $Queue$.
Ensure: Returns and removes the first item.
1: if $|Queue| = 0$ then
2:     return $\emptyset$
3: (Get first value from $Queue$.) $a \leftarrow Queue_1$
4: (Get number of $Queue$ items.) $c \leftarrow |Queue|$
5: (Remove item at index 1.) $Queue \leftarrow (Queue_2, \dots, Queue_c)$
6: return $a$
```

To complete the discussion, we provide below the basic algorithm for assessing if there are atoms to execute.

Algorithm 4.4.8 QueueEmpty()

```
Require: Filled in list of atoms $Queue$.
Ensure: Returns true if there are no atoms to execute and false otherwise.
1: if $|Queue| = 0$ then
2:     return $true$
3: else
4:     return $false$
```

4.4.6 Provenance-Aware Join

As we mentioned in previous sections, the provenance aware join is one of key operations in our model. It is responsible for combining inputs coming to a single input configuration from multiple nodes. Before presenting the algorithm of its calculation, we introduce this term based on the exemplary data extraction graph depicted in Figure 4.4.

Example 4.4.2 Provenance-Aware Join

The exemplary data extraction graph depicted in Figure 4.4 is relatively simple and its execution is intuitive, as discussed in Example 4.4.1. However, the final step, i.e. merging information coming to AS1 from different nodes, is not trivial.

The most simple possibility would be to perform the Cartesian product on all inputs coming from different nodes. However, it would lead to generating inconsistent records, for example, referring to Peugeot Megane. Intuitively, a combination of AS1 input values is logically valid if both ERT2 and ERT4 values can be tracked down to the same ERT1 output record as the make name provided to AS1, and if ERT4 values can be tracked down to the same ERT2 output record as the model name provided to AS1. The operation that uses such information for merging data will be called the *provenance-aware join* and will be based on tracking the identifiers of individual records that had impact on a value provided for an input slot.

We gathered individual values of make (extracted by ERT1), model (extracted by ERT2) and car details (extracted by ERT4; for simplicity we present all five extracted values together) in Table 4.3, together with identifiers of the corresponding ERT1 and ERT2 records. The table shows that, for example, all Peugeot models have the same ERT1 recordId as the Peugeot make, and that all details of any Peugeot model will have both ERT1 recordId and ERT2 recordId equal as given model. Thus, for example, record 20 can be joined with record 2 (they have the same shared recordId for ERT1) and with record 7 (which has the same values of recordId for both ERT1 and ERT2). At the same time it cannot be joined with record 6, as records 6 and 20 have a shared predecessor node (ERT2) for which they have different provenance (record 6 and 7 respectively).

As can seen from the table, there are a total of three makes, 12 models and 18 details attributes extracted. Their Cartesian product would consist of 648 records. By applying the provenance-based rule, we limit this number to 18 records. It is to be noted that this rule can remove some data completely. In this example, the results do not contain any record corresponding to Fiat Punto, as there were no details with the same recordId.

To make the provenance-aware join possible we need to define a data structure for keeping information on all output records that had an impact on values passed as input to a node. Such a structure, called *provenance track*, will store for each input record r passed through an input mapping λ a set of pairs $(\gamma, recordId)$, and will be written as $pt_r = \{(\gamma_1, recordId_1), \ldots, (\gamma_n, recordId_n)\}$. Interpretation of each of these pairs is that passed value depends on a specific record with $recordId_i$ that was generated by node γ_i.

In Table 4.3 the last three columns correspond to the (partial) provenance track of all generated input records sent to AS1[4]. For example, the single-attribute record corresponding to a record with rID = 4 would have the provenance track $tr_4 = \{(ERT1, 1)\}$, while record with rID = 28 would have the provenance track $tr_{28} = \{(ERT1, 3), (ERT2, 12)\}$. Provenance verification in a provenance-aware join consists in checking if for any two joined records the nodes shared by their provenance tracks have the same record id values.

The algorithm receives as input an input configuration ic and sets of data inputs assigned to each $\lambda \in ic$. The main idea of the algorithm is to consider all combinations of data inputs provided to an input configuration's input mappings. At any moment the currently considered combination of inputs is defined by the *indexes* collection that specifies current index in the collection of input data for each input mapping.

The algorithm is composed of two parts: initialization (lines 1–7) and main loop (lines 8–32). The loop consists of current record creation (9–21) and indexes update (lines 22–32).

[4]For simplicity we skip here records generated by nodes WAT1, WAT2, WAT3 and ERT3 that would normally be also reflected in the provenance track.

rId	attribute(s)	value	ERT1 rId	ERT2 rId	ERT4 rId
1	make	Fiat	1	-	-
2	make	Peugeot	2	-	-
3	make	Renault	3	-	-
4	model	500 3P	1	4	-
5	model	Grande Punto 3P	1	5	-
6	model	Peugeot 207 Facelift 3P	2	6	-
7	model	Peugeot 207 Facelift 5P	2	7	-
8	model	Peugeot 308 5P	2	8	-
9	model	Peugeot 3008 5P	2	9	-
10	model	Peugeot 407 Coupe	2	10	-
11	model	Renault Megane III 5P	3	11	-
12	model	Renault Megane III Coupe	3	12	-
13	model	Renault Megane III Estate	3	13	-
14	model	Renault Scenic 5P	3	14	-
15	model	Renault Fluence 4P	3	15	-
16	details	1.2 8V 69 M/5 POP	1	4	16
17	details	1.4 T-Jet 135 M/5 Abarth	1	4	17
18	details	1.2 75 M/5 URBAN	2	6	18
19	details	1.4 HDI 70 M/5 URBAN	2	7	19
20	details	1.4 HDI FAP 70 M/5 URBAN	2	7	20
21	details	1.4 75 M/5 URBAN	2	7	21
22	details	1.6 HDI FAP 90 M/5 CONFORT PACK	2	8	22
23	details	1.6 HDI FAP 110 M/6 PREMIUM	2	9	23
24	details	2.7 HDI FAP 204 Tiptronic/6 FELINE	2	10	24
25	details	1.5 dCi FAP 90 M/5 Expression	3	11	25
26	details	1.5 dCi FAP 110 M/6 Dynamique	3	11	26
27	details	1.9 dCi FAP 130 M/6 Dynamique	3	11	27
28	details	1.5 dCi FAP 110 M/6 Bose	3	12	28
29	details	1.5 dCi FAP 110 M/6 Dynamique	3	13	29
30	details	1.9 dCi FAP 130 M/6 Carminat Tom Tom	3	13	30
31	details	1.5 dCi FAP 110 M/6 Exception	3	14	31
32	details	1.5 dCi 85 M/5 Privilege	3	15	32
33	details	1.5 dCi 105 M/6 Expression	3	15	33

Table 4.3: Example of Extracted Values With Their Provenance

The algorithm starts by initializing two collections that assign each input mapping the total number of corresponding inputs (*counters*, initialized in line 3) and current index (*indexes*, initialized in line 6). If any of the counters is zero, an empty result is returned from the algorithm (lines 4–5).

Current record creation consists of few steps repeated for each input mapping λ in *ic*. Firstly, data record r at the current index of λ input data is accessed (line 11). Next, the algorithm verifies if for all nodes shared by *newRecord* and r their provenance tracks have the same record id values (line 12). If it is the case, r attributes are added to the *newRecord* (lines 13–14) and the *newRecord*'s provenance track is updated with the complete provenance of r (line 15). Otherwise, the record creation for a given combination of indexes is canceled, as some values for this combination of indexes have different provenance than others (lines 16–18).

If a record was successfully created, the pair of *ic* destination node and id number of *newRecord* is added to *newRecord*'s tracking provenance (line 20) and the record is added

Algorithm 4.4.9 ProvenanceAwareJoin($ic, data$)

```
Require: Input configuration $ic$, collection of lists of input data indexed by input mapping $data$.
Ensure: $ReturnData$ list of records.
 1: (Initialize $ReturnData$.) $ReturnData \leftarrow ()$
 2: for $\lambda \in ic$ do
 3:   (Initialize counters.) $counters_\lambda \leftarrow |data_\lambda|$
 4:   if $counters_\lambda = 0$ then
 5:     return
 6:   Initialize index. $indexes_\lambda \leftarrow 1$
 7: (Initialize loop variables.) $overflow \leftarrow false, p \leftarrow 1$
 8: repeat
 9:   (Initialize record.) $newRecord \leftarrow ()$
10:   for all $\lambda \in ic$ do
11:     (Get value corresponding to $\lambda$ for current index.) $r \leftarrow data_{\lambda, indexes_\lambda}$
12:     if $\forall_{\gamma \in tr_{newRecord}} \gamma \in tr_r \rightarrow tr_{newRecord,\gamma} = tr_{r,\gamma}$ then
13:       for all $sl \in ss(\lambda)$ do
14:         (Add value to $newRecord$) $newRecord \xleftarrow{add} (ds(\lambda, sl), r_{sl})$
15:       (Update $tr_{newRecord}$.) $tr_{newRecord} \xleftarrow{\bigcup} tr_r$
16:     else
17:       (Reset $newRecord$.) $newRecord \leftarrow ()$.
18:       Exit for
19:   if $newRecord \neq ()$ then
20:     (Add $newRecord$'s id $tr_{newRecord}$.) $tr_{newRecord} \xleftarrow{\bigcup} (d(ic), newRecord_{id})$
21:     (Add $newRecord$ to $ReturnData$.) $ReturnData \xleftarrow{add} newRecord$
22:   (Set the rightmost position.) $p \leftarrow |ic|$.
23:   repeat
24:     (Reset $overflow$.) $overflow \leftarrow false$
25:     (Get $\lambda$ at index $p$ of $ic$.) $\lambda \leftarrow ic_p$
26:     (Increase index of $\lambda$ data.) $indexes_\lambda \leftarrow indexes_\lambda + 1$
27:     if $indexes_\lambda > counters_\lambda$ then
28:       (Reset $indexes_\lambda$.) $indexes_\lambda \leftarrow 1$
29:       $overflow \leftarrow true$
30:       $p \leftarrow p - 1$
31:   until $overflow = false \vee p = 1$
32: until $p = 0 \wedge overflow = true$
```

to *ResultData* (line 21).

The indexes update step consists in finding the next combination of indexes that have not been used for record creation yet. The algorithm consecutively attempts to increment by one the index for specific input mapping, starting with the rightmost (line 26). If the incremented value is higher than the values counter for a specific input mapping (line 27), the index is set to one (line 28), and the algorithm moves to next position (line 30) and sets the *overflow* flag to true (line 29). The index update stops (line 31) if new configuration have been found

($overflow = false$) or when all existing positions have been incremented (i.e. $p = 0$) and $overflow$ is still set to $true$. In the second case also the main loop will end (line 32).

While the previously discussed example concerned an acyclic graph, the provenance-aware join algorithm is adapted also to graphs that contain cycles. In this situation, provenance tracking for the specific input slot of a node may contain a set of more than one value of record id. In this context some precisions may be necessary about how multiple values should be added to provenance tracking (lines 15 and 20) and compared (line 12). Adding new items to provenance tracking for a node is done by a union operator. We assume that if both arguments of this union operator (used in lines 15 and 20) contain a set of recordIds for the same node, the result of the operation will contain for this node a single set of recordIds resulting from the union of both sets of recordIds. For example, if $tr_{newRecord} = \{\gamma, \{1, 2\}\}$ and $tr_r = \{\gamma, 3\}$, after executing line 15 $tr_{newRecord}$ would be equal to $\{\gamma, \{1, 2, 3\}\}$. Consequently, the comparison performed in line 12 is true only if both compared sets are identical.

4.4.7 Data Storing

The final group of subroutines of the ExecuteGraph algorithm is related to managing the flow of data between nodes. It consists of the essential algorithm StoreData and two auxiliary subroutines NewInputRecords and NodeComplete.

Algorithm 4.4.10 StoreData($\lambda, data$)

```
Require: Input mapping $\lambda$, new input data $data$ for this input mapping.
Ensure: Updated $NewRecords$ for node $n$ and history of $AllInputs$ for input mapping $\lambda$.
1: (Store data.) $AllInputs_\lambda \xleftarrow{add} data$
2: for all $ic$ such that $\lambda \in ic$ do
3:   (Initialize $ic$ completeness.) $configurationComplete \leftarrow true$
4:   (Initialize $pajData$ for provenance-aware join.) $pajData \leftarrow ()$
5:   for all $im \in ic$ such that $im \neq \lambda$ do
6:     if $|AllInputs_{im}| = 0$ then
7:       (Update $ic$ completeness.) $configurationComplete \leftarrow false$
8:       Exit for
9:     else
10:      (Add $im$ inputs to $pajData$) $pajData \xleftarrow{add} AllInputs_{im}$
11:  if $ConfigurationComplete = true$ then
12:    (Add $data$ to $pajData$.) $pajData \xleftarrow{add} data$
13:    (Get destination node $n$.) $n \leftarrow d(\lambda)$
14:    (Update $NewRecords$ for node $n$.) $NewRecords_n \xleftarrow{add}$
       ProvenanceAwareJoin($ic, pajData$)
```

Intuitively, whenever some new input is provided for an input mapping, it should be passed further to all input configurations that contain this mapping. If by this moment the given configuration has already had some input for all other input mappings, it is possible to generate new input records for the input configuration destination node by using the provenance-aware join. All new data for a node should be stored to enable the node's instantiation. These tasks are performed by Algorithm 4.4.10.

The algorithm starts by adding *data* to the collection of inputs provided for input mapping λ for future reference. The next part of the algorithm (lines 3–14) is repeated for all input configurations *ic* that contain input mapping λ. It starts by variables initialization (lines 3–4), then all input mappings that are complementary to λ in *ic* are checked for the presence of any input (lines 5–10). If for all these input mappings some input exists, the $ConfigurationComplete$ value is set to *true* and $pajData$ is iteratively filled in by complete input for all these input mappings. Next, *data* are added to $pajData$ as input values for λ (line 12) and the result of the provenance-aware join of data $pajData$ for input configuration *ic* is added to $NewRecords_n$ (line 14).

It is to be noted that while for all input mappings but λ complete data are used as input for the provenance-aware join; for λ the algorithm uses only the new data (passed as input to the algorithm), ignoring previous values (if they exist). As a result, if multiple consecutive inputs are provided for the same input mapping λ, only the new values are used to generate records added to $NewRecords$.

Provided that the $NewRecords$ collection is properly filled in by the StoreData algorithm, the implementation of two other data management subroutines is straightforward. In case of NodeReady, it is enough to verify if $NewRecords$ contains any input for given node n. In case of NewInputRecords, the contents of $NewRecords$ for specific node n are returned and removed from the collection. We present both these algorithms below.

Algorithm 4.4.11 NodeReady(n)

```
Require: Node $n$ to be checked for its inputs readiness
Ensure: Returns $true$ if there are any new inputs for node $n$.
1: if $|NewRecords_n| > 0$ then
2:     return $true$
3: else
4:     return $false$
```

Algorithm 4.4.12 NewInputRecords(n)

```
Require: Node $n$ for which new input records should be returned
Ensure: Returns collection of new input records for node $n$ since $NewInputRecords$ has
    been last called for node $n$.
1: (Get new data for node $n$.) $returnData \leftarrow NewRecords_n$
2: (Clean up $NewRecords$ collection for node $n$.) $NewRecords_n \leftarrow ()$
3: return $returnData$
```

4.5 Navigation State Management

As we stated in Section 4.2.1, navigation state consists of information associated by the Web server and Web browser with a current user session. While in simple Web sites navigation state depends only on the most recent Web action, in complex data-intensive Web sites it can depend on complete navigation history. At the same time, content returned the by Web server to the user often depends on the current navigation state. Thus, in many Web sites, to access specific content we need to recreate some navigation state before issuing a specific HTTP request. In this section aspects of Web site navigation related to the complexities of client-side and server-side navigation state evolution are discussed.

4.5.1 Basic Elements of Navigation State Management

As we discussed in Section 4.2.1, navigation state can be divided into intuitively understandable, site-specific logical parts. Some of them are server-side and others are client-side.

Examples of client-side parts of navigation state include the current Web document and all other client-side data (such as JavaScript objects or client-set Cookies) used for session maintenance (or their logical subparts). Parts of server-side navigation state are not directly observable. However, their existence can be induced from a Web site's behavior. Examples of server-side parts of navigation state include shopping carts or current authentication information.

Different parts of the navigation state may be modified by different graph nodes. At the same time, nodes may require that a specific navigation state is set while their atoms are executed. In this section we focus on modeling these two phenomena and on their impact on the data extraction algorithms described in Section 4.4.

We start by analyzing two exemplary parts of the navigation state: the current Web document (part of client-side navigation state) and the content of the shopping cart (part of the server-side navigation state). The following two examples show how these two parts of the navigation state are modified by data extraction nodes and how nodes depend on them.

The shopping cart is a typical e-commerce server-side part of the navigation state. While it is not directly observable, operations on it are intuitive and logically distinct from other actions that modify the server-side navigation state. We illustrate them by the following example.

Example 4.5.1 Shopping Cart Part of Navigation State

To add an item to a shopping cart, one needs to perform a specific Web action such as clicking a button at product page. Such an action *modifies* the content of the shopping cart part of the navigation state by adding the product to the list of products already contained in the shopping cart. Another typical example of Web action that modifies the shopping cart include submitting a form at the order details page (it typically allows the user to change the number of items of each product) or completely removing the contents of the shopping cart. Finally, the Web action of placing an order also removes the contents of the shopping cart.

In an on-line shop there are several types of Web actions that *depend* on the current contents of the shopping cart. For example, Web pages that display shopping cart contents or allow users to place an order have different content depending on the items that are in the shopping cart. Similarly, different product recommendations and details on delivery costs may be presented depending on the actual contents of the shopping cart.

As we mentioned before, a Web action that adds a product to a shopping cart modifies the corresponding part of the server-side navigation state. However, at the same time, it depends on the shopping cart content at the moment of Web action execution. The content of the shopping cart after adding an item would be different if the shopping cart has contained already some products or if it was empty. It means that this Web action performs a *cumulative modification* of shopping cart content, and is *state-dependent* on the shopping cart part of the navigation state.

By contrast, the action of removing all items from the shopping cart always has the same impact on the shopping cart, whatever its content was before the action was executed. Thus, it modifies the shopping cart part of the navigation state but does not depend on it.

Web action's dependence on server-side state may be more or less explicit. For example, navigating to a Web page that displays the shopping cart depends on the shopping cart part of the navigation state very explicitly. For the pages that display products or special offers related to the content of the shopping cart, the dependency is much less explicit.

While server-side navigation state plays a very significant role in the behavior of many complex data-intensive Web sites, it is the current Web document that is the most easily observable part of the navigation state. This part of the navigation state is modified by the majority of Web actions. For example, any top-level navigation Web action completely replaces one Web document with another one, while a frame navigation Web action replaces some part of the Web document.

It is to be noted that not all modifications of current Web document are significant, and some of them may be ignored. For example, loading another document in a frame containing advertisements is typically of no interest in information extraction scenarios. Similarly, information on the number of items in a shopping cart that is updated after adding a product to shopping cart via an AJAX request is often of no use.

Example 4.5.2 Current Web Document Part of Navigation State

Let us come back to the Example 4.2.2 depicted in Figure 4.2. In this situation we will treat $W1$ and $W3$ Web actions as modifying the current Web document part of the navigation state. At the same time, as $W2$ only changes an insignificant part of the Web document, it will be considered an action without impact on the current Web document part of the navigation state.

Let's assume that the regular expression data extraction is performed on the current document at state $S1$. The content of the $S1$ Web document is needed for the creation of extraction rule atoms. However, their execution can be performed even after the current document is replaced by Page 3, as the content needed by the extraction rule had previously been stored by the StoreData subroutine. Thus, this extraction rule is not state-dependent on the current Web document.

Similarly, if $W3$ is a top-level navigation action in Web browser, it is not state-dependent on th current Web document, even if the accessed URL depends on the content of Page 2. By contrast, if $W3$ consists in simulating the click of a specific element of Page 2, it is required that at the moment of execution of any $W3$ atom, Page 2 is loaded into Web browser. Thus, in this situation $W3$ would be state-dependent on the current Web document part of the navigation state.

After an intuitive overview presented above, we formally define below the key terms related to navigation state management.

Definition 4.5.1 Part of Navigation State
Given a navigation state $n = (s, c) \in N_x$ of Web site x, any named, logically distinct part of the server-side navigation state s or client-side navigation state c will be called *part of navigation state* and will be denoted $n^{(p)}$, where p is the name of the part.

We assume that the value $n^{(p)}$ can be accessed for all $n \in N_x$. If given navigation state n does not contain any information for navigation state part p, $n^{(p)} = \emptyset$.

Definition 4.5.2 Node Modifying Part of Navigation State
A node γ *modifies part of navigation state* p if for some atom ϵ created by γ and some navigation state in which the atom is executed n, the navigation state $n^{(p)}$ is modified by executing the atom, i.e. $\exists_{\epsilon=\gamma(\dots)}\exists_{n_1\in N_x} n_1 \xrightarrow{\epsilon} n_2 \rightarrow n_1^{(p)} \neq n_2^{(p)}$.

Definition 4.5.3 Node Depending on Part of Navigation State
A node γ *depends on part of navigation state* p, or is *state-dependent on* p if at least some atom ϵ created by γ has different output depending on value $n^{(p)}$ at the moment of ϵ execution, i.e. $\exists_{\epsilon=(\alpha^{(m)},(o_1,\dots,o_m))=\gamma(\dots)}\exists_{n_1,n_2\in N_x:n_1^{(p)}\neq n_2^{(p)}}\alpha_{n_1}^{(m)}(o_1,...,o_m) \neq \alpha_{n_2}^{(m)}(o_1,...,o_m)$, where $\alpha_{n_i}^{(m)}(o_1,...,o_m)$ denotes the execution of atom ϵ in navigation state n_i.

4.5.2 State-Aware Graph Execution Challenges

The basic objective of introducing state management to our model is to ensure that, for any node that depends on some part of navigation state p, $n^{(p)}$ is identical at the moment of node instantiation, atom creation and atom execution. This task is challenging for at least two reasons.

Firstly, it is necessary to model all possible combinations of state-dependence and state modifications for multiple parts of the navigation state, and to use this information to execute atoms in optimal order.

Secondly, the situation when a node γ both depends on p and modifies it needs to be handled. This was, for example, the case of adding items to the shopping cart in Example 4.5.1, or the situation when $W3$ consisted in clicking an HTML element in Example 4.5.2. We illustrate the complexity of such a situation by the following example.

Example 4.5.3 Navigation State Recreation Need
Let γ be a node that both depends on p (corresponding e.g. to a shopping cart) and modifies it. Let $\epsilon_1, \epsilon_2, \epsilon_3$ be three atoms created by γ (for example, corresponding to adding three products to a shopping cart). Let's also assume that these three atoms are executed directly after their creation (i.e. that no other atoms were executed in the meantime).

We start the execution when $n^{(p)} = n_1$ (e.g. a empty shopping cart). First, we execute ϵ_1, which sets $n^{(p)} = n_2$ (e.g. a shopping cart containing product 1). In this situation n_p is already different than at the moment of atom creation. If we continued the execution of ϵ_2, it would add product 2 to the shopping cart that already contains product 1, rather than to the initial empty shopping cart. Further execution of ϵ_3 would make shopping cart contain three items instead of only product 3. The observed impact on the shopping cart is not only unintended, but also unpredictable – e.g. if ϵ_3 was executed as the first atom, the shopping cart after its execution would contain just one product, and if it was executed as the second, the shopping cart would contain two products.

To assure the expected behavior of the executions, before execution of the second and the third atom the part of navigation state p should be reset to its initial value. We will further call this process *state recreation*. Then, the execution of ϵ_1 would update $n^{(p)} = n_2$; next, the state $n^{(p)} = n_1$ would be recreated (i.e. all items would be removed) and execution of ϵ_2 would make the shopping cart contain only one product. Similarly, execution of ϵ_3 would be preceded by the recreation of state n_1.

The above example brings in two important problems that are addressed in the following part of this section. The first of them is related to how and when the navigation state should be recreated. Intuitively, while in some cases the state can be recreated without any navigation action (e.g. by serializing and deserializing Web document or by using the Web browser's cache), a sequence of navigation actions may be sometimes needed.

It means that state recreation can be an expensive process. Thus, the second challenge is to minimize the number of state recreations that are needed. While some state recreations cannot be avoided, state-aware queue management and query planning in some cases significantly limit their number.

4.5.3 Modeling State-Dependence and State Modifications

To deal with stateful Web actions, we slightly modify the general model described in Section 4.3 and the algorithms presented in Section 4.4. We start by enriching nodes representation with meta-data describing their relation to parts of the navigation state. The information about a node's impact or dependency on specific parts of the navigation state will be stored in a data structure called *node's state annotation*, defined below.

Definition 4.5.4 Node's State Annotation
Node's state annotation sa_γ is a pair $sa_\gamma = (sad_\gamma, sam_\gamma)$ of the set of all parts of the navigation state that γ depends on sad_γ, and the set of all parts of the navigation state that are modified by γ atoms sam_γ.

For each node γ, its state annotation consists of two elements: the set of parts of the navigation state that γ depends on sad_γ, and the set of parts of the navigation state that are modified by the execution of γ atoms sam_γ. If both of these sets are empty, γ is stateless. If only sam_γ contains some elements, the node modifies some part of the navigation state, but depends on none. For example, the Web action that resets the content of the shopping cart belongs to this category. If only sad_γ is not empty, the node depends on some part of the navigation state but modifies none. It is, for example, the case of most of extraction rules. If both these sets are not empty, their content may be disjoint or overlap. Displaying content of the shopping cart in the Web browser is an example of disjoint sets, as this Web action depends on the shopping cart and modifies the Web document part of the navigation state. By contrast, adding an item to the shopping cart, which both depends on and modifies the shopping cart part of the navigation state, is an example of overlapping sets.

4.5.4 State-Aware Graph Execution Algorithms

The main data extraction algorithm ExecuteGraph(D), presented in Section 4.4.3, does not include any features related to navigation state management. However, thanks to its modular character, state-awareness can be added by modifying subroutines CreateAtom and Execute, without any need to modify the main algorithm.

There are four main elements of these modifications. Firstly, a node's state annotation is used to detect state changes for a specific part of the navigation state, to save information needed for its recreation and to assign a unique identifier to any new navigation state. Secondly, if it is necessary, the state before atom execution is recreated. Thirdly, at any moment of graph execution, the identifier of the current value for the state part p is kept in the $CurrentState_p$ variable. Finally, for any atom, the information on the values of all parts of the navigation state when an atom is created is kept in collection $AtomState_{atom}$.

Below we present the Execute($atom$) subroutine adapted to state management.

Algorithm 4.5.1 Execute($atom$)

```
Require: Atom to be executed atom.
Ensure: Collection of records generated by atom data.
 1: (Get creator node.) $\gamma \leftarrow$ CreatorNode(CreatorInstance($atom$))
 2: (Initialize loop variable.) $StateModified \leftarrow false$
 3: for all $p \in sad_\gamma$ do
 4:    if $AtomState_{atom,p} \neq CurrentState_p$ then
 5:       (Some state part has been modified since atom creation.) $StateModified \leftarrow true$
 6: if $StateModified = true$ then
 7:    (Recreate state.) RecreateState($AtomState_{atom}$)
 8: (Execute atom.) $data \leftarrow$ Execute$_{atom}$()
 9: for all $p \in sam_\gamma$ do
10:    (Save state part data for recreation.) $id \leftarrow$ SaveState($p$)
11:    (Update current state definition.) $CurrentState_p \leftarrow id$
```

The algorithm starts by verifying if some of parts of the navigation state that the atom creator node depends on have been modified since atom creation (lines 2–5). If this is the case, the state is recreated (line 7). Then, the atom is executed (line 8). Next, each part of the navigation state p that is modified by atom execution is saved and assigned a new identifier (line 10), which is used to update the $CurrentState$ collection (line 11).

The below CreateAtom subroutine supplements this algorithm by saving the configuration of the navigation state parts at the moment of atom creation for further reference.

Algorithm 4.5.2 CreateAtom($instance, r$)

```
Require: Instance to be used for atom creation $instance$, input data record $r$.
Ensure: Created $atom$.
1: (Create $atom$.) $atom \leftarrow \text{CreateAtom}_{instance}(r)$
2: (Save current configuration of state parts.) $AtomState_{atom} \leftarrow CurrentState$
```

The actual behavior of the Execute algorithm depends on the pair of state saving and recreation routines. There are multiple methods of recreating the navigation state. Server-side state recreation can be based on replaying the sequence of Web actions or HTTP requests. Client-side state recreation approaches vary from the same methods as used for the server-side state recreation, through usage of browser-level or proxy-level cache to Web document serialization. As the details of different approaches are very technical, we discuss them in the chapter describing prototype implementation, in Section 5.3.2.

Out of different state recreation methods, some require more detail on how a given part of the navigation state is modified by the execution of individual atoms. For example, methods based on replaying a sequence of atom executions that led to the original state distinguish five types of impact of each node on any of identified the navigation state parts.

Firstly, a node may have no significant impact on a given part of the navigation state. For example, some mouse actions change the visibility of menu elements without significantly modifying the document that is loaded into the Web browser. Similarly, information extraction nodes do not modify any part of the navigation state. On the other hand, some Web document modifications (such as loading an ad in an HTML frame) are typically insignificant from the perspective of information extraction and can be ignored. Thus, they also fall into this category of nodes.

Secondly, a node (*state reseting node*) may reset all data associated with a specific part of the navigation state. For example, it may remove all items from the shopping cart, load an empty document into the Web browser or end the current authentication session.

Thirdly, a node (*partial replacement node*) may partially modify some navigation state part. Following a link that affects only one frame of the HTML frameset or replacing some part of the Web page by using an AJAX request or JavaScript content generation are examples of partial replacement Web actions with respect to current Web browser documents.

Fourthly, a node (*state-cumulative node*) may add new information without removing other previously stored data for a given navigation state part. Adding an item to the shopping cart is an example of a state-cumulative Web action with respect to part of the server-side navigation state corresponding to shopping cart contents. Downloading additional data to a client-side data grid (for example, after clicking a “Show more items” button) is an example of a node that is state-cumulative with respect to the current Web browser document.

Finally, a node (*complete replacement node*) may completely replace information associated with some part of the navigation state with new values. For example, logging into a Web site that requires authentication completely replaces all previous login data. Client-side examples include following a hyperlink in a frameless Web site or a hyperlink with its `target` attribute set to "top" in a frameset Web site.

Below we provide formal definitions of all four classes of nodes that have some significant impact on some part of the navigation state.

Definition 4.5.5 State Reseting Node
A data extraction node γ is called a *state reseting node* for some part of the navigation state p if the execution of atoms created by this node removes all information stored in part of the navigation state p. The set of all navigation parts for which γ is a state reseting node will be denoted by $sam_{\gamma}^{\varnothing}$.

Definition 4.5.6 State-Cumulative Node
A data extraction node γ is called a *state-cumulative node* for some part of navigation state p if the execution of atoms created by this node adds new information to the part of navigation state p without replacing any previous values. The set of all navigation parts for which γ is a state-cumulative node will be denoted by sam_{γ}^{+}.

An important property of atoms created by state-cumulative nodes is that their impact on a given part of the navigation state does not depend on the order of execution of these atoms. It is to be also noted that state-cumulative nodes are always state-dependent on the part of the navigation state that they modify.

Definition 4.5.7 State Replacement Node
A data extraction node γ is called a *state replacement node* for some part of the navigation state p if the execution of atoms created by this node replaces some information stored in part of the navigation state p. The set of all navigation parts for which γ is a state replacement node will be denoted by sam_{γ}^{*}.

We assume that state replacement nodes are always state-dependent on the part of the navigation state they modify.

Definition 4.5.8 Complete and Partial State Replacement Nodes
A data extraction node γ is called a *complete state replacement node* for some part of navigation state p if the execution of atoms created by this node completely replaces all information stored in part of the navigation state p. γ is called a *partial state replacement node* if it is a state replacement node and is not a complete state replacement node. The set of all navigation parts for which γ is a complete state replacement node will be denoted by $sam_{\gamma}^{\bigstar}$, and the set of navigation parts for which γ is a partial state replacement node will be denoted by $sam_{\gamma}^{\blacklozenge}$

As we can see from the above definitions, complete state replacement nodes are a special case of state replacement nodes, and state resetting nodes are a special case of complete state replacement nodes. Thus, $sam_{\gamma}^{\varnothing} \subset sam_{\gamma}^{\bigstar} \subset sam_{\gamma}^{*}$.

We make an assumption that complete state replacement nodes do not depend on the part of the navigation state they replace. We have never seen real-life examples of Web sites in which this assumption is not met. However, if in some very rare cases the actual behavior of complete state replacement depends on previous navigation state, it should be modeled rather as a partial state replacement node.

It is to be noted that all classes of nodes described above should be always interpreted in the context of a specific part of the navigation state. It is possible that a given node is state-cumulative for some part of the navigation state and resets or completely replaces some other part of the navigation state. Moreover, the same impact of a given node may be interpreted as belonging to different class of state modification, depending on the granularity of the used part of the navigation state. We provide two examples of such situations below.

Example 4.5.4 Transforming Partial to Complete State Replacement

Let's assume that node γ_1 corresponds to a Web action of clicking a button. After the button is clicked, some distinct part of the current Web document that does not contain the button (e.g. one of the frames or a single `DIV` element) is completely replaced by new content. We also assume that the current document has been already loaded by some previous top-level navigation Web action γ_0.

From the perspective of the part of the navigation state p corresponding to the current Web document, γ_1 is dependent on p and partially replaces it, while γ_0 is not dependent on p and completely replaces it. As a result, according to the presented algorithms, if there are multiple buttons to be clicked, before each click the complete Web document needs to be recreated, for instance, by executing γ_0. Thus, performing these click actions can be very expensive.

Now, let's consider small-grained parts of the navigation state: "part of current Web document corresponding to toolbar" ($p_1 \in p$) and "part of current Web document containing data" ($p_2 \in p$). As γ_0 completely replaces p it also completely replaces p_1 and p_2. However, γ_1 can be now considered as dependent on p_1 and completely modifying p_2. As a result, it becomes obvious to state-aware graph execution algorithms that a sequence of button clicks may be performed one after another without any need of navigation state recreation.

Example 4.5.5 Transforming Partial State Replacement Node to State-Cumulative and Complete State Replacement Node

Let's assume a situation very similar to the previous example, however, with γ_1 having a slightly different behavior. Instead of replacing the content of some HTML element, it generates or loads from the server a new record and adds it at the end of an HTML table (without removing previous rows).

Similarly, as in the previous example, if interpreted as a modification of the current Web document, γ_1 is a partial state replacement node. However, if we interpret it as modification of the navigation state part corresponding to the table containing data p_2, γ_1 is a state-cumulative node. Similarly, as in the previous example, thanks to using a fine-grained part of the navigation state it is possible to avoid need of navigation state recreations. However, if any other node γ_2 is dependent on p_2, due to its cumulative character its recreation will be required for each button click.

Fortunately, in this case yet another interpretation is possible. We can consider a special part of the navigation state that corresponds to the last element of table $p_3 \subset p_2$. As each execution of the γ_1 atom adds a new record, it also completely replaces the previous value of p_3. Thus, it makes γ_1 a complete state replacement node for p_3. If γ_2 can be redefined in a way that it depends on p_3, the need of state recreation can be completely avoided.

The above examples demonstrate that the right choice of granularity of navigation state parts can be the key to limiting the need of state recreations. However, it is to be noted that the operations presented above are not always possible. For example, state-cumulative node adding an item to the shopping cart typically cannot be interpreted as complete replacement of part of the navigation state corresponding to the last added item, because no other action can be interpreted as dependent on such a part of the navigation state.

4.5.5 State-Aware Queue Management

The following aspect of state-aware graph execution is related to queue management. Using variants of the ENQUEUE algorithm presented in Section 4.4.5 may lead to need of multiple state recreations that could be avoided. Below we present the variant of the ENQUEUE algorithm that focuses on reducing the number of state recreations. The basic idea of the algorithm is to try to place atoms that are dependent on a part of navigation state before atoms that modify this part of navigation state.

Algorithm 4.5.3 ENQUEUESTATEAWARE(a)

```
Require: Atom to enqueue $a$.
Ensure: $a$ is added to one of pre-initialized queues of atoms $Queue$ and $QueueM$
1: (Get creator node.) $\gamma \leftarrow$ CREATORNODE(CREATORINSTANCE($a$))
2: if $|sam_\gamma| > 0$ then
3:     (Get number of $QueueM$ items.) $c \leftarrow |QueueM|$
4:     (Add $a$ to the $QueueM$ at the beginning.) $QueueM \leftarrow (a, QueueM_1, \ldots, QueueM_c)$
5: else
6:     (Get number of $Queue$ items.) $c \leftarrow |Queue|$
7:     (Add $a$ to the $Queue$ at the beginning.) $Queue \leftarrow (a, Queue_1, \ldots, Queue_c)$
```

The above variant of the ENQUEUE algorithm uses internally two queues instead of one. The first of them, $Queue$, is used to store atoms that do not modify navigation state (and are or are not dependent on some part of navigation state). The second one, $QueueM$, stores atoms that modify navigation state. In both cases the new atoms are added at the beginning of the queue, corresponding to depth-first search graph traversal. However, more complex solutions can be used if needed. For example, atoms created by non-output nodes could be always put after all output nodes' atoms.

Algorithm 4.5.4 PopQueueStateAware()

```
Require: Filled in lists of atoms $Queue$ and $QueueM$.
Ensure: Returns and removes the first item.
 1: if $|Queue| + |QueueM| = 0$ then
 2:     return $\emptyset$
 3: if $|Queue| > 0$ then
 4:     (Get first value from $Queue$.) $a \leftarrow Queue_1$
 5:     (Get number of $Queue$ items.) $c \leftarrow |Queue|$
 6:     (Remove item at index 1.) $Queue \leftarrow (Queue_2, \ldots, Queue_c)$
 7: else
 8:     (Get first value from $QueueM$.) $a \leftarrow QueueM_1$
 9:     (Get number of $QueueM$ items.) $c \leftarrow |QueueM|$
10:     (Remove item at index 1.) $QueueM \leftarrow (QueueM_2, \ldots, QueueM_c)$
11: return $a$
```

These two queues are used by the PopQueueStateAware method presented above. The main idea behind this algorithm is to pop atoms starting with ones not modifying any part of the navigation state (stored in *Queue*), and popping *QueueM* items only if *Queue* is empty.

When used in GraphExecute these two subroutines ensure that after an atom modifying some part of navigation state is executed, and its outputs are used to create other atoms, all of the created atoms that do not modify state are executed. As a result, if any of them is state-dependent, it is assured that it is executed in the same state as when it was created. Thus, state recreation is not necessary.

A state-aware version of the QueueEmpty subroutine is presented below.

Algorithm 4.5.5 QueueEmptyStateAware()

```
Require: Filled in lists of atoms $Queue$ and $QueueM$.
Ensure: Returns true if there are no atoms to execute and false otherwise.
 1: if $|Queue| + |QueueM| = 0$ then
 2:     return $true$
 3: else
 4:     return $false$
```

4.5.6 State Flow Correctness

Until now we assumed that each node can be modeled as modifying any part of the navigation state. However, such an approach can lead to ambiguities, i.e. situations when the actual navigation state at the moment of node instantiation differs for different possible orders of execution of previous atoms. We present an example of such ambiguity below.

Example 4.5.6 Navigation State Ambiguity

Let's assume that

- node γ has two input slots, filled in by input coming from nodes γ_1 and γ_2,
- both γ_1 and γ_2 completely replace part of navigation state p: γ_1 sets p to value v_1 and γ_2 to v_2,
- γ_1 was used to create one atom α_1 and γ_2 to create atom α_2,
- each of these atoms, when executed, returns a single data record (r_1 and r_2 respectively), and
- the provenance-aware join of r_1 and r_2 returns one input record r for node γ,
- there is no input mapping between γ_1 and γ_2 (in any direction), and as a result, they can be instantiated in any order.

Based on presented assumptions, there are two possible situations when γ is instantiated. The first possibility is that α_1 was executed before the execution of α_2. The second one is that α_1 was executed after α_2. In both cases the values passed as input to γ are the same. However, in the first case, the value corresponding to p is v_2 (α_2 was executed as the second and overwrote the previous value v_1), while in the second case p is set to v_1. Thus, three discussed nodes are an example of *navigation state ambiguity*, i.e. the situation when some part of navigation state at the moment of instantiation of a node cannot be unambiguously determined by using the structure of the data extraction graph.

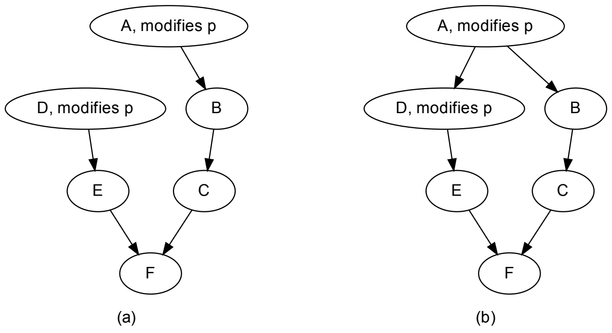


Figure 4.9: Example of Navigation State Ambiguity (a) and State Unambiguity (b)

This is the simplest case of navigation state ambiguity, when ambiguity is caused by two nodes that directly provide input to the third one. However, ambiguity can be caused even by nodes that are a few mappings away from the node where ambiguity occurs.

Example 4.5.7 Complex Navigation State Ambiguity

Let us consider, for example, the graph depicted in Figure 4.9.(a). Let's assume that both node A and D modify completely replace or reset some part of navigation state p, and that this part of navigation state is modified by none of the other nodes. Even if A and D are not directly connected to node F, they will cause state navigation state ambiguity in this node, as graph structure does not determine that atoms created by A are always executed before atoms created by D or *vice versa.*

This observation does not mean that state can be modified only by the path coming to one input configuration of a node. The graph depicted in Figure 4.9.(b) is similar to the previous one. However, an additional input mapping that exists between nodes A and D makes the order of execution of A and D atoms fixed. Thus, the ambiguity is removed.

It is to be noted that ambiguity is present only if the order of performing updates of some part of navigation state p has an impact on the final value of this part of the navigation state. Thus, if all nodes that modify p are state-cumulative, the ambiguity is not present.

Below we provide a formal definition of navigation state ambiguity.

Definition 4.5.9 Navigation State Ambiguity
Data extraction graph $D = (N, IM)$ demonstrates *navigation state ambiguity*, if it contains at least one pair of nodes $\gamma_1, \gamma_2 \in N$ such that:

- γ_1 and γ_2 both modify some part of navigation state p,
- at least one of nodes γ_1, γ_2 is not state-cumulative for p, and
- D does not contain a path from node γ_1 to γ_2 nor from γ_2 to γ_1.

Algorithm 4.5.6 AnalyzeGraphStateAware(D)

Require: Connected data extraction graph D.

Ensure: Updated configurations, nodes collections and executable subgraph of $Max(D)$.

```
1:  (Initialize configurations collections.) $MConfigs, Configs, IConfigs \leftarrow ()$
2:  (Initialize nodes collections.) $StartNodes, EndNodes, OutputNodes, NINodes \leftarrow ()$
3:  for all $\gamma_1, \gamma_2 \in N(D)$ such that $|sam_{\gamma_1}| > 0 \wedge \gamma_1 \neq \gamma_2 \wedge |sam_{\gamma_2}| > 0$ do
4:    for all $p \in (sam_{\gamma_1} \bigcap sam_{\gamma_2})$ do
5:      if $\neg((p \in sam^+_{\gamma_1} \wedge p \in sam^+_{\gamma_2}) \vee$ PathExists$(\gamma_1, \gamma_2) \vee$ PathExists$(\gamma_2, \gamma_1))$ then
6:        return
7:  for all $\gamma \in N(D)$ do
8:    (Set of incoming input mappings.) $IM^\gamma \leftarrow \bigcup_{\lambda \in IM(D) \wedge d(\lambda) = \gamma} \lambda$
9:    (Set of outgoing input mappings.) $OM^\gamma \leftarrow \bigcup_{\lambda \in IM(D) \wedge s(\lambda) = \gamma} \lambda$
10:   if $|IM^\gamma| > 0$ then
11:     (Analyze node $\gamma$.) AnalyzeNodeRecursively$(\gamma, IM^\gamma, 1, ())$
12:   else
13:     (Add to start nodes.) $StartNodes \xleftarrow{add} \gamma$
14:   if $|OM^\gamma| = 0$ then
15:     (Add to end nodes.) $EndNodes \xleftarrow{add} \gamma$
16:   if $\gamma$ is a data collection node then
17:     (Add to data collection nodes.) $OutputNodes \xleftarrow{add} \gamma$
18:   if $|Configs_\gamma| = 0$ then
19:     (Save node as non-instantiable.) $NINodes \xleftarrow{add} \gamma$
20: if $|NINodes| = 0$ then
21:   (Completely executable graph.) $Max(D) \leftarrow D$
22: else
23:   (Find executable subgraph.) $Max(D) \leftarrow$ FindExecutableSubgraph$(D)$
```

The above definition is used directly by a state-aware version of the AnalyzeGraph algorithm (Algorithm 4.5.6). It starts by detecting navigation state ambiguity in the graph passed as argument. If any state ambiguity is detected the algorithm leaves the start, end and output nodes collections empty, and the execution is not continued.

The algorithm differs from Algorithm 4.4.2 by extra lines 3–6. For each pair of nodes that modify some part of the navigation state (line 3) and for each part of navigation state p modified by both nodes (line 4), it verifies if both states are cumulative or if there is a path between these two nodes (line 5). If it is not the case, navigation state ambiguity is detected, the algorithm ends without any results (all configurations and nodes collections are empty) and, consequently, ExecuteGraph ends without extracting any data.

4.5.7 Current Query Model

The state management algorithms presented in the previous section model the fact that some types of modifications to the navigation state occur and that some part of navigation state is used by some graph nodes. However, they do not model the content of individual parts of the navigation state. As a result, actual interpretation of the changes that occur is not possible. If needed, such information can be easily added, and used to optimize query planning, improve path management or filter some part of the extracted data based on the properties of the navigation state they come from.

In this section we present an example of more semantics-rich modeling of contents of some part of the navigation state related to the current query, i.e. the query actually issued to the server, that data contained in current Web document correspond to. It can be used to capture the behavior of typical mechanisms of data-intensive Web sites, such as search forms, dynamic and static filters, pagination and data ordering.

We adopt a relatively simple model of current query, which consists of a set of conjunctive constraints based on equality, information on current data ordering and information on the current scope of presented records (equivalent to the SQL `LIMIT` construct). To express both constraints and ordering information, we use named attributes that belong to an attribute set. This approach captures well the queries that can be issued to the majority of data-intensive Web sites. Moreover, it can be easily extended to other types of constraints in the future (using other comparison operators, similarity or pattern matching).

Definition 4.5.10 Query State
Query state is a part of navigation state $qs \subset n, n \in N_x$ represented as a triple $qs = (qs_c, qs_o, qs_l)$, where

- qs_c is a set of triples $qs_c = ((rs_1, a_1, v_1), ..., (rs_n, a_n, v_n))$ of attribute set rs_i, attribute belonging to this attribute set $a_i \in drs(rs_1)$ and value assigned to this attribute v_i,
- qs_o is a triple $qs_o = (rs, a, asc)$ of attribute set rs, attribute belonging to this $a \in drs(rs)$ and asc is a boolean value that is true if ascending ordering is used and false if ordering is descending,
- qs_l is a pair $qs_l = (l_s, l_e)$ of ranks of start (l_s) and end (l_e) records returned by Web site in current navigation state n.

Query state evolves during Web site navigation. To model this evolution, we attach to each data extraction node a description of the query state modifications it causes.

The modifications to current query state used in our model are:

- adding or replacing a constraint, specified as a triple (rs, a, v) of attribute set rs, attribute name a and constraint value v,
- removing a constraint, identified by attribute set rs and attribute name a,
- removing all set constraints,
- setting the ordering attribute, specified as a triple (rs, a, asc) of attribute set rs, attribute name a boolean flag stating if ordering is ascending asc,
- removing the ordering attribute,
- changing the direction of ordering for currently used ordering attribute,
- setting the start and end record rank of returned records,
- adding or subtracting a specific number from the start and end ranks of returned records.

Each such description can have no argument or use parameters passed to the node through its input slots. For example, if specific Web action concerns visiting a Web page that contains offers of selling used cars, the corresponding constraint would be set to a fixed value (i.e. *type* = "used"). Similarly, if each page of results contains 20 records, the Web action of clicking the "Next" button adds a fixed value (e.g. 20) to the ranks of start and end records.

However, if the type of cars used in filtering is an input slot of the corresponding Web action, instead of setting the value of the *type* constraint to a fixed value, we would set it to the value of respective input slot, i.e. $type = is_{type}$.

4.6 Query Planning

The data extraction graph is an abstraction that can be used both as a Web site model and as a representation of a user query plan. Data extraction graphs representing site models are used to capture useful navigation and data extraction patterns in the Web site. While such a model can be executable and cover only a single pattern of navigation paths in a Web site, in many cases it would be redundant (i.e. including multiple alternative paths for at least some of contained data). Moreover, in some cases it may not be executable and require additional input for at least some of its nodes (e.g. for a Web form with open-domain fields).

In contrast, a query plan graph needs to be executable and, when possible, it should not contain multiple paths leading to the same or partially overlapping data. It is also to be noted that executable Web site models without unnecessary path redundancies can be interpreted as query plans corresponding to extracting all data covered by the Web site model.

Executable query plans are created by the applying query planning algorithm. Its input consists of a user query and a data extraction graph describing a Web site (executable or not). The output contains a fully executable, state-unambiguous data extraction graph corresponding to the user query. In other words, the output is such a data extraction graph, that when it is executed, it returns all data relevant to the query and only data relevant to query.

4.6.1 Basic Query Planning Tasks

In this section we present the simplest approach to query planning. It consists of three main tasks: required subgraph selection, user input provision and data filtering.

Required Subgraph Selection

Date extraction graph that models a Web site can contain multiple output nodes that gather different subsets of data contained in a Web site, as well as multiple alternative paths arriving at a node that differ by inputs they require (e.g. they correspond to the different querying capabilities of a form or to a simple and advanced search facility).

A typical user query is much more specific than Web site model. In many cases it means that some of its nodes are not useful for data extraction. In includes all output nodes providing attributes not used in query, as well as other nodes that lead only to such output nodes. We will call such nodes *unnecessary* nodes. At the same time some nodes may be impossible to instantiate, given user input that is provided in the query. We will call such nodes *inaccessible* nodes.

Required subgraph selection consists in removing all unnecessary and inaccessible nodes together with all input mappings that start and end in these nodes. Once this operation is performed, it is necessary to check if the resulting graph is connected. If this condition is met, required subgraph selection was successful.

The required subgraph has two important properties. Firstly, as the graph is connected and all inputs it requires are provided by user query, it is sure that after a few transformations it will be fully executable. Secondly, as no useful nodes are removed, it is assured that this model corresponds to extraction of the maximal quantity of data contained in the Web site.

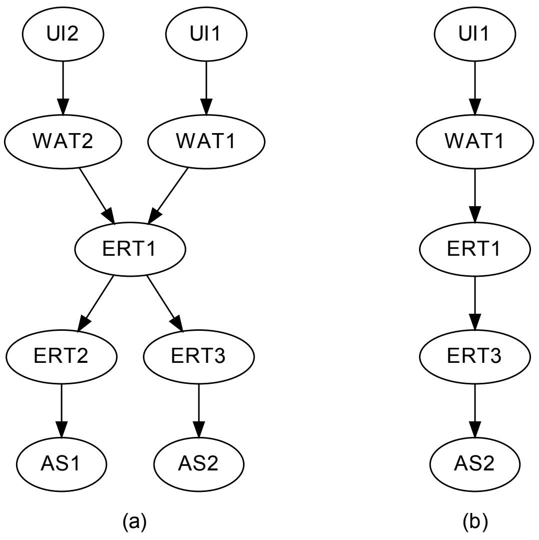


Figure 4.10: Web Site Graph (a) and Selected Required Subgraph (b)

Example 4.6.1 Required Subgraph Selection

Let us consider the data extraction graph presented in Figure 4.10.(a), which describes a simple data-intensive Web site in the automotive domain. Two Web actions templates WAT1 and WAT2 correspond to filling in two different forms: WAT1 provides a search by make and model, while WAT2 enables a search by market segments (such as sport cars, family cars, 4x4 vehicles). ERT1 corresponds to the extraction of the essential part of the results Web page. Then ERT2 and ERT3 are used to extract data records corresponding to new and used cars, respectively. Outputs of ERT2 and ERT3 are mapped into two attribute sets, AS1 and AS2. AS2, corresponding to used cars is similar to AS1, but has a few more attributes (mileage, state, date of last revision). We list the input and output slots of all nodes of this graph in Table 4.4.

Let's suppose that a user query concerns finding the engine details, price and mileage of used cars of a specific make, model and version. In this case WAT2 is an example of an inaccessible node, as the user query contains no input for its "segment" input slot. At the same time, as user query is answered based on the AS2 node, ERT2 and AS1 are examples of unneeded nodes. The required subgraph, constructed by removing these three nodes from Web site graph is depicted in Figure 4.10.(b). As it can be seen, the graph is connected; thus, it is possible to create an executable graph corresponding to user query.

Node	Type	Input Slots	Used output Slots
UI1	UserInput	–	make, model
UI2	UserInput	–	segment
WAT1	NavigateToURL	make, model	userContent
WAT2	NavigateToURL	segment	userContent
ERT1	XPath	!Input	element
ERT2	RegEx	!Input	8 subpatterns
ERT3	RegEx	!Input	11 subpatterns
AS1	Attribute set	make, model, engine displacement, engine type, engine power, gearbox type, version name, price	n/a
AS2	Attribute set	same as AS1 + mileage, state, date of last revision	n/a

Table 4.4: Description of Nodes of Graph From Figure 4.10.(a)

Providing User Input

Input nodes are trivial nodes that encode all needed data internally or automatically obtain them from an external source/user. If the generated subgraph contains some input node, it has no data associated with it. Thus, depending on the actual behavior of a specific user input node, in the case of execution it would pass to other nodes zero records (thus stopping further graph execution), or ask the user for inputs for value for all of its attributes.

To avoid such behaviors, after the subgraph is selected, it is necessary to assign the input node attributes with values specified in user query. In the discussed example, the make and model names would be loaded into UI1 node, so that they can be further used as inputs for WAT1.

Data Filtering

Required subgraph selection is capable of removing from the Web site data extraction graph some part of nodes and input mappings that are not applicable for a specific user query. However, this process does not align the scope of data returned by graph execution. We illustrate this problem by the following example.

Example 4.6.2 Data Scope Modification

Let us come back to the situation presented in Example 4.6.1. While the selection of the required subgraph removes some part of the initial Web data extraction model, the graph still returns data that do not correspond to the user query. Firstly, attribute set AS2, used as an output node in this graph, contains more data attributes than required by user query (e.g. gearbox type and data of the last revision). Secondly, while it assures that make and model conditions are met (both are used as inputs to WAT1), and that only used cars are extracted (as proper attribute set is used) it does not put into action the condition imposed by user query on version name.

The simplest methods to deal with the situation presented in the above example is to modify the attribute set definition to contain selection and projection operators consistent the with user query. In the provided example, the projection would be defined as $\pi =$ (engine displacement, engine type, engine power, price, mileage), while the definition of selection operator will be $\sigma = (version = UQ_{version})$.

4.6.2 Optimizing Data Filtering

Required subgraph selection, user input provision and data filtering techniques can be combined to obtain an executable graph corresponding to user query. While this approach is universal, i.e. it works for all graphs and all user queries that are issued to a single attribute set, there are multiple situations when its performance is very low.

The key performance issue with this approach is that it uses information on constraints incorporated in user query in the node where data is returned and not in the node when it is extracted. As a result, it is possible that many irrelevant records are used for multiple unnecessary node instantiations and atom executions, to provide data that is finally filtered by selection operator. Similarly, it can happen that some part of a graph is unnecessarily executed in order to provide input for attributes that are next removed by the projection operator.

In this section we discuss a few techniques for improving the performance of the proposed solution. They are based on moving data filtering rules (selection and projection) from output nodes to locations in the graph that are as near to nodes that create given values as possible. It can be done in a few ways. Firstly, an attribute set may be added just after the node that generates some data, in order to perform data filtering just afterwards. Secondly, for some nodes (e.g. some of the extraction rules) it is possible to incorporate filtering within the node definition. Thirdly, there are cases when applying the projection directly in the node or after it makes it possible to remove some other nodes that provide unused attributes for the considered output node. On the other hand, input values incorporated in the user input node can be also moved towards the nodes that use them.

Adding Data Filtering Attribute Sets

Any input slot of an output node receives values from the output slot of one or more graph nodes. As a result it is easy to identify the source node(s) of data provided to a given AS. To allow data filtering we will add after each source node γ an attribute set with input and output mappings corresponding to all used output slots of the node. Next, a single input mapping will be created between γ and the created attribute set, and all input mappings that started in node γ will be modified to start in the created attribute set. We illustrate such modifications in two distinct scenarios in the following example.

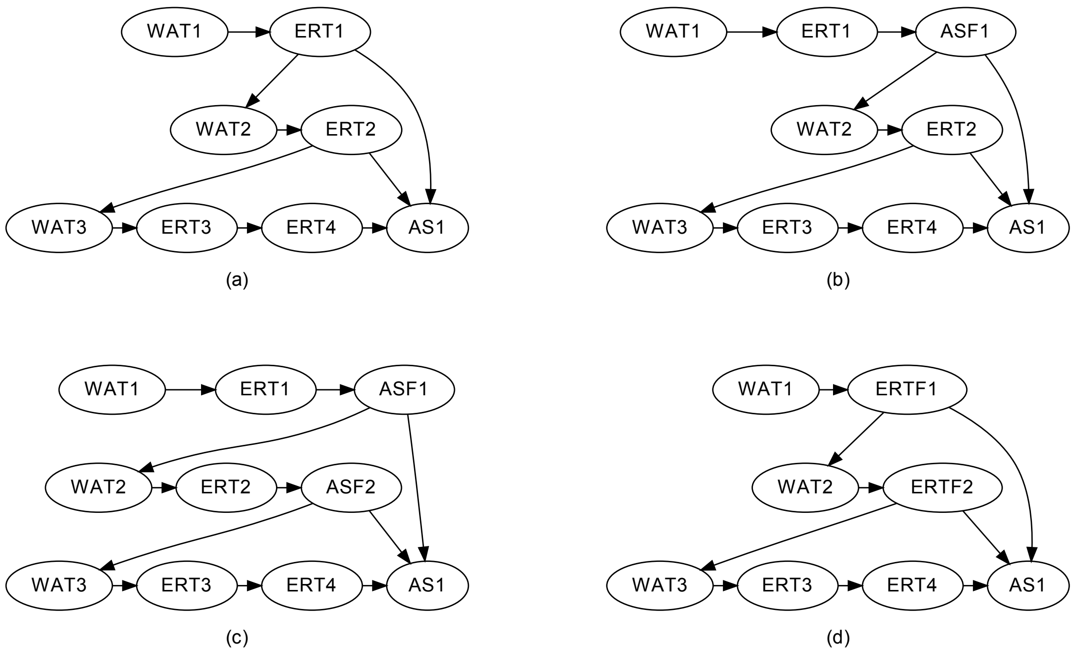


Figure 4.11: Adding Data Filtering Attribute Sets: (a) Original graph, (b) graph with one data filtering attribute set ASF1, (c) graph with two data filtering attribute sets ASF1 and ASF2, (d) the same graph with partial removal of input mappings

Example 4.6.3 Data Filtering Attribute Sets

Let's suppose that a Web site is described by the graph presented in Figure 4.11.(a), identical to of Example 4.3.4 (see Table 4.2 for the nodes' description). Let's also suppose that user query concerns extraction of all the attributes contained in the AS1 attribute set, but only for Fiat cars, and that data returned by the exemplary Web site are the same as in Table 4.3.

The simplest approach, discussed in the previous section, would be to add a selection operator (make = "Fiat") to AS1. However, in that case, for all extracted records in other car makes a sequence of extraction and navigation actions would be performed (WAT2, ERT2, WAT3, ERT3, ERT4), the provenance-aware join would be used to generate AS1 data records and only then all non-Fiat makes would be filtered out. As a result, a total of 18 records would be generated, out of which only two would remain after applying the selection operator. Obviously, this approach would be highly ineffective.

Better results can be obtained by adding an additional filtering attribute set ASF1 directly after ERT1, as illustrated in Figure 4.11.(b). After this modification is done, both input mappings that used to start at the ERT1 node, start at ASF1. As a result, values extracted by ERT1 rule are filtered with respect to their make attribute, and only three out of 19 records are passed to the WAT2 node.

Similarly, if the user query constrained both make and model values, the graph can be modified by adding two data filtering attribute sets. This situation is depicted in Figure 4.11.(c). Apart from ASF1 remaining from the previous situation, ASF2 is added directly after ERT2 in an analogous way.

Incorporating Data Filtering in Node's Definition

As demonstrated above, using data filtering attribute sets can provide significant performance gain. However, in the case of the previous example ERT1 and ERT2 still extract data that are not relevant to the user query and that are filtered only after the extraction is performed. In some situations it is possible to avoid this problem by incorporating data filtering features directly into the node definition. We demonstrate this in the following example.

Example 4.6.4 Incorporating Data Filtering Into Extraction Rule Template
Let's assume that the considered graph is identical to Example 4.6.3, and that the user query imposes constraints on both make and model (e.g. make = "Peugeot", model = "207"). Let's also assume that ERT1 and ERT2 are both defined as very simple XPath expressions `//a`.

In such a case, both constraints can be incorporated directly into ERT definitions. Extraction rule templates with incorporated data filtering would be: ERTF1 = `//a[text()='Peugeot']` and ERTF2 = `//a[text()='207']`. The modified data extraction graph is presented in Figure 4.11.(d).

The capability to incorporate data filtering into the extraction rule template definition depends on used data extraction language and extraction rule specification. Below we present how incorporation can be done in the case of regular expression and XPath rules.

Example 4.6.5 Data Filtering Incorporation in Case of Regular Expressions
In the case of data filtering concerning any subpattern of regular expression, incorporation is typically easy and consists in putting escaped value instead of original subpattern definition. For example, if initial regular expression had the form `([A-Za-z]+): ([0-9]+)` and we want to incorporate the condition "Subpattern2" = 187, the modified regular expression would be `([A-Za-z]+): (187)`.

However, incorporation may be more complex in the case of embedded subpatterns. Let's consider, for example, a subpattern that finds person name and corresponding PESEL number and extracts at the same time year, month and day of birth. For this task the following regular expression can be used: `(([0-9]{2})([0-9]{2})([0-9]{2})[0-9]{5}): ([A-Za-z ]+).`. Its first subpattern corresponds to PESEL number. The second, the third and the fourth subpatterns are embedded in the first one and correspond to year, month and day of birth respectively. Finally, the fifth subpattern corresponds to name of the person.

If the data filtering to be performed is based on the name of the person or on his/her birth year, the incorporation is straightforward and very similar to the previous case. However, if the filtering rule is based on PESEL number (e.g. it requires that it is equal to 82091708291), incorporation would require proper splitting of parts of PESEL number among subpatterns that are embedded in subpattern1. It would result in the following regular expression: `((82)(09)(17)08291):  ([A-Za-z ]+)..`

Example 4.6.6 Data Filtering Incorporation in Case of XPath Extraction Rule

We have already seen a sample case of the incorporation of data filtering into the XPath extraction rule in Example 4.6.4. Below we consider three more complex cases of XPath expressions.

Let's consider an extraction rule template `//a`. Let's suppose that two output slots of this node are used: @name and @href attributes, and that a constraint @name='audi' is imposed. Then, the ERT incorporating this data filtering contraint would be the following: `//a[@name='audi']`.

The previous examples of ERT were very simple and did not contain any XPath predicate. The following ERT: `//a[starts-with(@name,'more')]` selects `A` elements that have a `name` attribute starting with the value "more". Let's assume that two output slots of this ERT are used: @href and element text, and that we want to incorporate a data filtering rule that element text equals "Audi". Then the modified XPath ERT would be `//a[starts-with(@name,'more') and text()='Audi']`.

In the last example we consider the same ERT as previously, but we suppose that instead of filtering by element node, we want to enforce that `@name`="more_audi". The ERT incorporating this constraint would be `//a[starts-with(@name,'more') and @name='more_audi']`. While it is syntactically and semantically correct, a simple analysis leads us to the conclusion that the first predicate `starts-with(@name,'more')` is subsumed by the second one, and can be removed. Thus, the simplified version of ERT incorporating data filtering would be `//a[@name='more_audi']`

Above, we demonstrated how the incorporation of data filtering can be done in a relatively simple way in some basic cases. However, it is possible that some data extraction languages used will have no data filtering incorporation capabilities, or will be able to perform incorporation only for some classes of extraction rules.

Cutting Node and Input Mappings Attributes

In the two previous sections we discussed possible efficiency gains related to the early evaluation of the selection part of a user query. Similarly, in some cases important gains can be related to applying the projection operator not in an output node, but in a previous part of the navigation graph. We demonstrate this situation by the following example.

Example 4.6.7 Removing Useless Input Mappings

Let's suppose that for the same Web site data extraction graph as in the previous examples, the user poses a slightly modified query. The selection operator remains the same as previously (make = "Peugeot", model = "207"), however, the user is interested in receiving only engine displacement, engine type, engine power, gearbox type and version name attributes (all but make and model). The attributes required by the user correspond exactly to the output of the ERT4 extraction rule template.

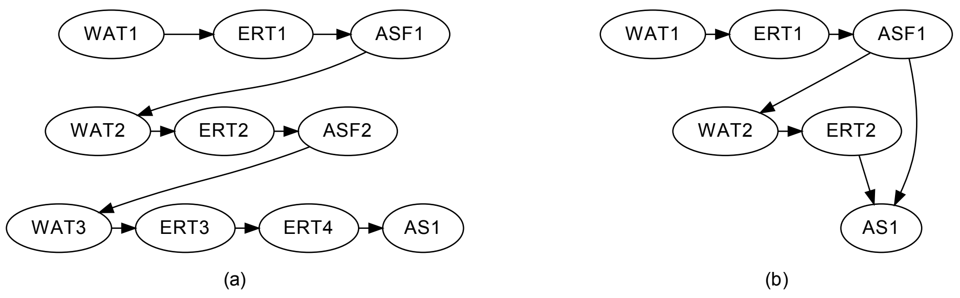


Figure 4.12: Removing Nodes That Provide Not Used Attributes: (a) Graph with removed useless input mappings, (b) graph with removed useless input mappings and nodes

The simplest method of limiting the scope of received attributes is to add a projection operator to the AS1 output node. However, in such a situation a useless provenance-aware join between inputs coming from ASF1 (make), ASF2 (model) and ERT4 (all other attributes) is performed, just before the inputs from ASF1 and ASF2 are discarded by the projection operator.

To avoid this situation, instead of using the projection operator, we can remove all not used attributes (make and model) from the AS1 input slots and remove all input mappings that are no longer necessary (i.e. input mappings from ASF1 and ASF2). This situation is depicted in Figure 4.12.(a).

In the above example the set of unused input slots of the attribute set are covered completely with two input mappings. As a result it was possible to completely remove these input mappings. However, in some situations the overlap between unused input slots and input mapping destination slot is not complete, as demonstrated by the next example.

Example 4.6.8 Removing Useless Destination Slots From Input Mappings

Let's assume that the situation is the same as in Example 4.6.7, but the user is interested in an even smaller subset of AS1 attributes: only engine displacement, gearbox type and version. In this case removing input mappings coming to AS1 from ASF1 and ASF2 is not enough, as there are two more attributes that need to be removed (engine type and engine power).

This can be done by using the projection operator. It is straightforward, but causes some overhead, as useless attributes need to be extracted and passed along input mapping from ERT4 to AS1. However, even this overhead can be removed by modifying the AS1 input slots (removing the engine type and engine power input slots), and the removing corresponding pairs of source and destination slots from the input mapping coming from ERT4 to AS1. Then, this input mapping would contain only three attributes (engine displacement, gearbox type and version) and would not pass any data that is not used by AS1.

In Example 4.6.7 we demonstrated how input mappings can be removed to improve the performance of graph execution, as compared with using the projection operator in a data node. In this case, after removing both input mappings their source nodes still remained required by the extraction process. However, in some situations, removing an input mapping can lead to removal of one or more nodes that are no longer used for data extraction. We demonstrate such a situation in the following example.

Example 4.6.9 Removing Useless Input Mappings and Nodes

Let's consider the same data extraction graph as in the previous example, and a user query that selects URL of make and model name for all Renault cars. As filtering concerns only the make name, only the ASF1 node would be added (ASF2 being unnecessary).

Following the ideas of the previous examples, we would start by modifying the AS1 input slots to contain only two attributes (make and model). Next, it we would remove input mapping from ERT4 to AS1. However, once done, the graph would contain a "branch" that started in the ERT2 node and went to the ERT4 node without providing input for any output node. Thus, nodes WAT3, ERT3 and ERT4 are not used by any output node and can be removed, together with their input mappings. Figure 4.12.(b) displays the complete graph, after all useless input mappings and nodes are removed.

Limiting State Recreations

One of the most important factors of the total costs of executing a data extraction graph, is related to the need of recreating navigation states. In Section 4.5.5 we presented how queue management can be used to minimize state recreation need without any modifications to the data extraction graph. However, in Example 4.5.3 we also presented examples of situations when a node both modifies some part of the navigation state and depends on it, and stated that in such situations, for given graph structure, state recreations cannot be avoided.

In this section we present a method of decomposing nodes that depend on and modify the same part of the navigation state into two distinct nodes in order to minimize need of navigation state recreations. We present an intuitive overview of this approach below.

Example 4.6.10 Node Decomposition Limiting Navigation State Recreations

Let us consider a Web action that consists in clicking a button in the current Web document. When clicked, this button runs some JavaScript code that ends up by performing top-level navigation to another Web page. Thus, it modifies (completely replaces) the current Web document navigation state part, but it also depends on it (the document needs to be loaded to enable mouse click simulation).

While technically it is a single user action, the described Web action consists of two separate logical subactions. The first consists in executing some JavaScript code as a result of a button being clicked. This JavaScript code ends up causing navigation to start.

The second subaction consists in performing actual top-level navigation, which replaces the content of the Web browser. The first subaction depends on the current Web document, but does not modify it. The second one replaces the current Web document but does not depend on it. If we apply state-aware queue management, all atoms of the first action will be executed first (without modifying current Web document) and only then all resulting navigation Web actions will be executed. As they do not depend on thre current Web document there will be no longer any need of navigation state recreation.

To enable node decomposition as described in the example, we implemented a special type of node called request capture. It wraps any Web action dependent on the client-side navigation state. Execution of any of its atoms internally executes the corresponding atom of the original Web action. However, if the execution causes some navigation action, it is canceled and captured URL and POST data (if applicable) are returned as the node's output.

Thanks to implementation of the request capture node, it is possible to deal with any Web action dependent on some part of the client-side navigation state that also completely replaces the same part of the navigation state. It can be automatically decomposed into a request capture node that wraps this Web action and a simple GET or POST navigation action. Between these two nodes an input mapping is created with source and destination input slots corresponding to URL, POST data and request HTTP method (See Section 5.2.3 for details).

It is noted that this approach cannot be applied to Web actions that depend on and have an impact on some part of the server-side navigation state. For example, there is no technical way to decompose an action that both depends on the shopping cart and modifies it.

4.6.3 Query Planning Algorithm

In previous sections we presented a few graph modification techniques important for query planning. Now we present the query planning algorithm that combines all of them.

User Query

We understand user query to mean a declarative specification of the scope of data to be retrieved from the Web site, constraints that need to be met by all items acquired by the query, as well as the ordering of acquired records. User query is expressed in an abstract way and is related to the Web site's data extraction graph only by its input and output nodes.

Definition 4.6.1 User Query
User query consists of a 5-tuple $uq = (uq_{rs}, uq_{\sigma}, uq_{\pi}, uq_o, uq_i)$, where

- uq_{rs} is a single output node used to define requested query output;
- uq_{σ} is a selection operator, i.e. the set of n constraints expressed as pairs $uq_{\sigma} = ((a_1, v_1), ..., (a_n, v_n))$ of attribute a_i belonging to rs and a constant value assigned to this attribute v_i,
- uq_{π} is a projection operator, i.e. a subset of m attributes of rs, $uq_{\pi} = (a_1, ..., a_m), a_i \in rs$,
- uq_o is an optional pair $qs_o = (a, asc)$ of attribute a belonging to rs and a boolean value asc that is true if ordering should be ascending and false if it should be descending,
- uq_i is a set of k triples $uq_i = ((\gamma_1, os_1, v_1), ..., (\gamma_k, os_k, v_k))$ of input node γ_i, the name of output slot of this node $os_i \in os(\gamma_i)$ and the value that input node should return for this output slot v_i.

In the query planning algorithm presented in this section we assume that user a query specifies the input for any number of input nodes, but uses the outputs of a single output node (attribute set). For each input node the query needs to provide user input for each attribute. For the output it needs to specify the projection (i.e. the attributes to be extracted), selection (i.e. constraints that need to be met by the extracted data) and data ordering (i.e. the attribute that needs to be used for data ordering and ordering direction). All selection constraints are represented as a pair of attribute name and constant value. It is important to note that selection constraints can use attributes that are not part of query projection.

Query Planning Algorithm and Its Subroutines

Below we present the query planning algorithm that uses data extraction graph D describing a Web site, and user query uq, in order to generate a completely executable data extraction graph P that corresponds to uq. Similarly, as in the case of the graph execution algorithm, we introduce it in a top-down way, i.e. we start by presenting the main algorithm and discuss its subroutines afterwards.

In this algorithm a few distinct parts can be identified. The first part (lines 1–9) is responsible for the provision of user input to graph input nodes and for the removal of incomplete (thus, useless) input nodes. The second part (lines 10–24) adds data filtering attribute sets and removes attributes not used in query projection or ordering from input mappings. The third part (lines 25–28) removes unused attributes from the graph output node used in the query. The next three lines use subroutines for applying additional graph optimization techniques. Finally, in lines 31–32 selection of the required subgraph is performed by calculating the maximal executable subgraph of D, and the return value P is defined.

The first part, corresponding to input provision, consists of a loop executed for each input node γ. All input values provided for given node γ are passed to the node (line 4) and counted (lines 2 and 5). If a given node did not receive input corresponding to all its outputs slots (line 6), the node and all its outgoing input mappings are removed (lines 7–9).

The second part of the algorithm, adding data filtering attribute sets and removing non-projection attributes from input mappings, consists of a single loop (lines 11–24) executed for each node γ that provides input to uq_{rs}. It starts by creating a data filtering attribute set γ_{asf} (line 11) and adding it to the graph (line 12). Next, each input mapping λ outgoing from γ is analyzed (lines 14–22). Firstly, its source node is changed to γ_{asf} (line 14). Then, a few operations are performed if λ is an incoming input mapping of uq_{rs} (lines 16–22). All pairs of source and destination slots of λ that are not used by the projection of data ordering (line 17) are removed from λ (line 18). If a given input slot is used in a user query selection (line 19), the selection constraint corresponding to this input slot is added to γ_{asf}. If operations performed in lines 17–18 removed all pairs of source and destination slots of λ, λ is removed from the graph (as it is not longer useful). Finally, new input mapping from γ to γ_{asf} is created (line 23) and added to the graph (line 24).

The third part of the algorithm (lines 25–28) consists of removing input slots not used in the projection or as ordering attribute from uq_{rs} input and output slots. It is to be noted that this operations remove also attributes used in selection (as they have already been added to proper data filtering attribute sets). We make an assumption that input and output slot removes automatically removes corresponding selection constraints from node uq_{rs}.

Algorithm 4.6.1 PlanQuery(D, uq)

```
Require: Connected data extraction graph $D$, user query $uq$.
Ensure: Return ready-to-execute data extraction graph $P$ or an empty graph if user query
    cannot be answered.
 1: for all $\gamma$ such that $\gamma$ is an input node do
 2:   (Initialize counter) $c \leftarrow 0$
 3:   for all $os, v$ such that $(\gamma, os, v) \in uq_i$ do
 4:     (Store value $v$ in $\gamma$.) SetInput($\gamma, os, v$)
 5:     (Increment counter.) $c \leftarrow c + 1$
 6:   if not $c = |os(\gamma)|$ then
 7:     for all $\lambda \in oIM(\gamma)$ do
 8:       (Remove input mapping.) $IM(D) \leftarrow IM(D) \backslash \lambda$
 9:     (Remove incomplete input node.) $N(D) \leftarrow N(D) \backslash \gamma$
10: for all $\gamma$ such that $\exists_{\lambda \in IM(D)} d(\lambda) = uq_{rs} \wedge s(\lambda) = \gamma$ do
11:   (Create data filtering attribute set.) $\gamma_{asf} \leftarrow$ CreateASF($os(\gamma)$)
12:   (Add data filtering attribute set to graph.) $N(D) \xleftarrow{add} \gamma_{asf}$
13:   for all $\lambda \in IM(D)$ such that $s(\lambda) = \gamma$ do
14:     (Change source node.) $s(\lambda) \leftarrow \gamma_{asf}$
15:     if $d(\lambda) = uq_{rs}$ then
16:       for all $(os, is) \in l_\lambda$ do
17:         if $is \notin uq_\pi \wedge is \neq a(uq_o)$ then
18:           (Remove not used pair of slots.) $l_\lambda \leftarrow l_\lambda \backslash (os, is)$
19:         if $\exists_v (is, v) \in uq_\sigma$ then
20:           (Add selection operator to $\gamma_{asf}$.) $\sigma(\gamma_{asf}) \xleftarrow{add} (is, v)$
21:       if $|ss(\lambda)| = 0$ then
22:         (Remove empty input mapping.) $IM(D) \leftarrow IM(D) \backslash \lambda$
23:   (Create input mapping from $\gamma$ to new node.) $\lambda_{asf} \leftarrow$
      $(\gamma, \gamma_{asf}, ((os(\gamma)_1, os(\gamma)_1), ..., (os(\gamma)_n, os(\gamma)_n)))$
24:   (Add input mapping to graph.) $IM(D) \xleftarrow{add} \lambda_{asf}$
25: for all $is \in is(uq_{rs})$ do
26:   if $is \notin uq_\pi \wedge is \neq a(uq_o)$ then
27:     (Remove from $uq_{rs}$ input slots.) $is(uq_{rs}) \leftarrow is(uq_{rs}) \backslash is$
28:     (Remove from $uq_{rs}$ output slots.) $os(uq_{rs}) \leftarrow os(uq_{rs}) \backslash is$
29: (Decompose nodes.) $D \leftarrow$ StateAwareNodeDecomposition($D$)
30: (Merge nodes.) $D \leftarrow$ MergeNodes($D$)
31: (Get $Max(D)$.) $Max(D) \leftarrow$ AnalyzeGraph($D$)
32: (Set $P$.) $P \leftarrow Max(D)$.
```

The two following lines (29–30) make D the subject of further optimizations by two subroutines corresponding to the state-aware nodes' decomposition (i.e. adding request capture nodes where useful) and to merging nodes that can be merged (incorporation of data filtering into extraction rule templates, merging extraction rule templates, merging two consecutive attribute sets). Both these subroutines are presented in the following part of this section.

Algorithm 4.6.2 StateAwareNodeDecomposition(D)

```
Require: Connected data extraction graph $D$.
Ensure: $D$ with some nodes that depend on a state and modify it decomposed.
1: for all $\gamma \in N(D)$ such that $sam_\gamma \bigcap sad_\gamma \neq \emptyset$ do
2:   if CanDecompose($\gamma$) $= true$ then
3:     (Change $\gamma$ into request capture node.) $\gamma \leftarrow$ CreateRequestCaptureNode($\gamma$)
4:     (Create navigation node.) $\gamma_n \leftarrow$ CreateNavigateNode()
5:     (Add new node to graph.) $N(D) \xleftarrow{add} \gamma_n$
6:     for all $\lambda \in IM(D)$ such that $s(\lambda) = \gamma$ do
7:       (Change source node.) $s(\lambda) \leftarrow \gamma_n$
8:     (Create new input mapping.) $\lambda = (\gamma, \gamma_n, ((url, url), (requestMethod, requestMethod), (postData, postData), (frame, frame)))$
9:     (Add input mapping to graph.) $IM(D) \xleftarrow{add} \lambda$
```

In line 31, the AnalyzeGraph subroutine (see Section 4.4.2) is used to identify the maximal completely executable subgraph of modified graph D. Finally, in line 33 subgraph linearization by adding pass-through slots is done by subroutine Linearize which is presented further on.

We start the presentation of the three graph optimization subroutines by showing how the state-aware decomposition of some Web actions that both depend on some part of the navigation state and modify it can be done (Algorithm 4.6.2).

The algorithm is performed for all such nodes γ that modify at least one part of the navigation state that they also depend on (line 1). For each of them subroutine CanDecompose is called to assess if for a given node state-aware decomposition is possible (line 2). The following part of the algorithm (lines 3–9) is executed only if this subroutine returns true.

The actual decomposition starts by replacing node γ by a new request-capture node that wraps γ (line 3) and by creating a new navigation node γ_n, capable of executing GET and POST HTTP requests (lines 4–5). Next, the source node of all input mappings outgoing from γ are set to γ_n (lines 6–7). Finally, new input mapping between γ and γ_n, responsible for transferring details of captured navigation action (URL, method, post data and target frame), is created and added to the graph (lines 8–9).

Below we discuss the subroutine of the PlanQuery algorithm responsible for merging pairs of nodes that can be merged (such as pairs of extraction rule templates or pairs consisting of the extraction rule template and data filtering attribute set).

The algorithm consists of a single loop that is performed on all nodes γ_1 that have only one outgoing input mapping (line 1). The CanBeMerged subroutine, specific for a given type of node γ_1, is used to verify if a given node can be merged with the destination node γ_2 of its single input mapping λ (line 4). If it is the case, a new node γ_m is created by calling the MergeNodes subroutine, which is specific for a given type of node γ_1 (line 5) and added to the graph (line 6). Next, the destination node of all input mappings that come to the node γ_1 or to the node γ_2 is set to γ_m (lines 7–8), similarly as source node of all input mappings outgoing from node γ_2 (lines 9–10). Finally, λ, γ_1 and γ_2 are removed from the graph.

Algorithm 4.6.3 MERGENODES(D)

```
Require: Connected data extraction graph $D$.
Ensure: $D$ after merger of all nodes that can be merged.
 1: for all $\gamma_1 \in N(D)$ such that $|oIM(\gamma_1)| = 1$ do
 2:   (Get single outgoing input mapping.) $\lambda \leftarrow oIM(\gamma_1)_1$
 3:   (Get second node.) $\gamma_2 \leftarrow d(\lambda)$
 4:   if CanMergeNodes$(\gamma_1, \gamma_2) = true$ then
 5:     (Get merged node.) $\gamma_m \leftarrow$ MergeNodes$(\gamma_1, \gamma_2)$
 6:     (Add merged node to graph.) $N(D) \xleftarrow{add} \gamma_m$
 7:     for all $im \in IM(D)$ such that $d(im) \in \{\gamma_1, \gamma_2\}$ do
 8:       (Change $im$ destination node.) $d(im) \leftarrow \gamma_m$
 9:     for all $im \in IM(D)$ such that $s(im) = \gamma_2$ do
10:       (Change $im$ source node.) $s(im) \leftarrow \gamma_m$
11:     (Remove $\lambda$ from $D$.) $IM(D) \leftarrow IM(D)\backslash\{\lambda\}$
12:     (Remove nodes from $D$.) $D \leftarrow D\backslash\{\gamma_1, \gamma_2\}$
```

The last query plan optimization algorithm (Algorithm 4.6.4) partially linearizes the data extraction graph by adding pass-through slots and removing some input mappings.

The algorithm consists of two distinct parts. The first, shorter path (lines 1–7) removes from a copy G of graph D all input mappings that cannot be used in linearization. The second, longer part (lines 8–40) finds linearizable parts of graph and linearizes them.

The algorithm starts by creating a copy G of graph D (line 1) used to identify the part of graph D that can be linearized. Then, the queue of ready-to-analyze nodes Q is initialized with $StartNodes$ (line 2).

Next, a loop is performed for each input mapping λ of graph G (line 3–7). First. it checks if there exists any input configuration that has the same destination node as input mapping λ but does not contain it (line 4). If it is the case, λ cannot be used for linearization and it is removed from G (line 5). If, as a result of this operation, all input mappings coming to the destination node of λ are removed (line 6), G gets disconnected and destination of λ becomes a node that cannot receive any input. Thus, we treat it in the same way as start nodes, and add it to the queue of ready-to-process nodes (line 7). It assures that even disconnected parts of G will be analyzed by the second part of the algorithm.

After this part of algorithm is executed, data extraction graph G contains only input mappings that can be extended with extra attributes (e.g. corresponding to pass-through input slots of destination node). While G may be disconnected, it is easy to demonstrate that it is acyclic, even if D contained cycles.

As D is a completely executable extraction graph, its each node can be instantiated, given the input of its predecessors. At the same time, any input mapping that closes a cycle sends input only if all its predecessor nodes (including the first node of the cycle) have already been instantiated. Thus, for the first node of a cycle, there exists at least one input configuration that do not contain the input mapping closing the cycle. Therefore, the first part of the linearization algorithm removes this input mapping, consequently removing the cycle.

Algorithm 4.6.4 LINEARIZE(D)

Require: Completely executable data extraction graph D.

Ensure: D with all possible linearizations performed.

```
1: (Create a copy of $D$.) $G \leftarrow D$
2: (Initialize queue $Q$.) $Q \leftarrow StartNodes$
3: for all $\lambda \in IM(G)$ do
4:   if $\exists_{ic \in ics} d(ics) = d(\lambda) \wedge \lambda \notin ic$ then
5:     (Remove $\lambda$ from $G$.) $IM(G) \leftarrow IM(G) \backslash \{\lambda\}$
6:     if $|\{im \in IM(G) : d(im) = d(\lambda)\}| = 0$ then
7:       (Add node to $Q$.) $Q \xleftarrow{add} d(\lambda)$
8: (Initialize visited input mappings.) $vims \leftarrow ()$
9: for all $\gamma \in Q$ do
10:   (Initialize paths.) $ps_\gamma \leftarrow ()$
11: while $|Q| > 0$ do
12:   (Get first node from the queue.) $\gamma \leftarrow$ POP($Q$)
13:   for all $\lambda \in IM(G)$ such that $s(\lambda) = \gamma$ do
14:     for all $p \in ps_\gamma$ do
15:       (Create $p_\gamma$ by adding current input mapping to $p$.) $p_\gamma \leftarrow (p_1, \ldots, p_n, \lambda)$
16:       (Add $p$ to paths of destination node of $\lambda$.) $ps_{d(\lambda)} \xleftarrow{add} p$
17:     (Mark $\lambda$ as visited.) $vims \xleftarrow{add} \lambda$
18:     if $\neg\exists_{im \in IM(G)} d(im) = d(\lambda) \wedge im \notin vims$ then
19:       (Add $d(\lambda)$ to $Q$.) $Q \xleftarrow{add} d(\lambda)$
20:       (Find candidate input mappings.) $cims \leftarrow \{im \in IM(G) : d(im) = d(\lambda)\}$
21:       if $|cims| > 1$ then
22:         for all $im \in cims$ do
23:           (Get all subpaths that start by $s(im)$ in a non-final position.) $cps \leftarrow$
              $\{(p_i, ..., p_n) : p_i \in ps_{d(\lambda)} \wedge s(p_i) = s(im) \wedge i < n\}$
24:           if $|cps| = 1$ then
25:             (Get the only subpath.) $sp \leftarrow cps_1$
26:           else if $|cps| > 0$ then
27:             (Get minimal length of subpath.) $ml \leftarrow \min_{i=1...|cps|}(|cps_i|)$
28:             (Get all subpaths of length $ml$.) $sps \leftarrow \{sp \in cps : |sp| = ml\}$
29:             (Choose any of $sps$ as $sp$.) $sp \leftarrow sps_1$
30:           if $|cps| > 0$ then
31:             for $i = 1$ to $ml - 1$ do
32:               (Add extra input slots.) $is(d(sp_i)) \leftarrow is(d(sp_i)) \bigcup ss(im)$
33:               (Add extra output slots.) $os(d(sp_i)) \leftarrow os(d(sp_i)) \bigcup ss(im)$
34:               (Add extra source slots.) $ss(sp_i) \leftarrow ss(sp_i) \bigcup ss(im)$
35:               (Add extra destination slots.) $ds(sp_i) \leftarrow ds(sp_i) \bigcup ss(im)$
36:             (Add extra source slots.) $ss(sp_{ml}) \leftarrow ss(sp_{ml}) \bigcup ss(im)$
37:             (Add extra destination slots.) $ds(sp_{ml}) \leftarrow ds(sp_{ml}) \bigcup ds(im)$
38:             (Remove $im$ from $D$.) $IM(D) \leftarrow IM(D) \backslash \{im\}$
```

The second part of the algorithm consists of a single loop (lines 12–40) executed as long as there are any nodes in queue Q of ready-to-process nodes. It uses two additional data structures. ps, initialized in lines 9–10, assigns each node γ a set ps_γ of all graph paths that lead to γ, expressed as a sequence of input mappings. $vims$, initialized in line 8, is a set of all input mappings that have already been visited.

In each iteration of the loop, the first node γ is obtained and removed from queue Q (line 12). Next, each input mapping λ outgoing from γ is analyzed (lines 14–40). Firstly, all paths coming to node γ are copied (line 14), extended with λ (line 15) and saved as paths of λ destination node (line 16). Then, λ is added to the set of visited nodes $vims$. The final part of the algorithm (lines 19–40) is executed only if input mappings incoming into destination node of λ that exist in G have already been visited (line 18).

If the above condition is met, the destination node of λ is added to the queue Q (line 19). Next, all input mappings that come to this node are found (line 20). If there are at least two of them (which is a pre-condition to have an input mapping removed and for a path to be extended with pass-through slots) (line 21), for each input mapping im (line 22), all subpaths that start in its source node, end in its destination node and contain at least one additional node are found (line 23). Is such subpaths exist, linearization of part of the graph defined by im's start and end node is possible.

In cases when a single path corresponding to im is found, it is used as the path to be extended with pass-through slots (line 25). If multiple such paths are found any of these that contain the minimal number of input mappings is chosen (lines 27–29).

If any path was found, actual linearization is performed (lines 31–38). Firstly, pass-through slots are added to all but the last node in the chosen path (lines 32–33) and to corresponding input mappings (lines 34–35). Then, the last input mapping in the path is extended with im's source and destination slots (lines 36–37). Finally, im is removed from D (line 38).

4.7 Summary

In this chapter we introduced our proposed data extraction and navigation model, which is capable of performing data extraction from complex data-intensive Web sites. We started by presenting our research objectives in Section 4.1. Next, in Section 4.2, we introduced the key elements of our data extraction model: Web actions and Web action templates, extraction rules, extraction rules composition and extraction rules templates, as well as data records and attribute sets. In Section 4.3, we presented the notion of the data extraction graph that generalized the aforementioned elements and relations between them. Then, in Section 4.4 we presented a set of algorithms that enable execution of the data extraction graph, i.e. performing Web site navigation and data extraction according to it. Both the data extraction model and graph execution algorithms are further extended to enable data extraction from stateful Web sites, as described in Section 4.5. Finally, in Section 4.6, we completed the proposed data extraction model by presenting a query planning algorithm, which makes it possible to issue arbitrary user queries to data-intensive Web sites by using a Web site's data extraction graph and a number of graph transformation techniques.

Chapter 5

Implementation of Proposed Method

In this chapter we describe an instantiation of models and methods introduced in the previous chapter [152], i.e. their proof-of-concept implementation as a Windows application. This chapter is rather technical; however, it covers our important contributions related to Web actions execution, documents representation and actual data extraction.

Prototype implementation of the proposed solution was done as a Microsoft Windows .NET 2.0 WinForms application in C# language. The application uses a few third-party .NET libraries: the CsExWB Web browser for embedding Internet Explorer COM control, the FiddlerCore library for the proxy-level capturing of transfered content, the QuickGraph library for implementing graph-related operations, .NET wrapper of libcurl for handling some HTTP requests, the GraphViz program for graph visualization, and the PowerCollections library for set-related operations. For some Web browser-based Web actions we used a modified version of fragments of code developed by Dawid Węckowski for his master's thesis [276].

5.1 System Architecture

While the implementation of our data extraction model should be considered rather as a proof-of-concept prototype than a fully-fledged system, we tried to follow industry best practices. Thus, the developed implementation is modular and extensible with new types of nodes and instances, as well as with variants of the ExecuteGraph subroutines.

The architecture of our implementation is presented in Figure 5.1. Its components are shortly presented below and discussed in more detail in the remainder of this chapter.

Four key components of the system are marked by bold font. The first of them, the Data Model, presented in more detail in Section 5.1.1, is responsible for representing the abstractions of our model described in Sections 4.2 and 4.3. A few of these abstractions, namely nodes, instances, atoms and input mappings, are represented as interfaces, enhancing the system's extensibility. These interfaces, as well as data extraction graph representation, are used by all other components of our prototype.

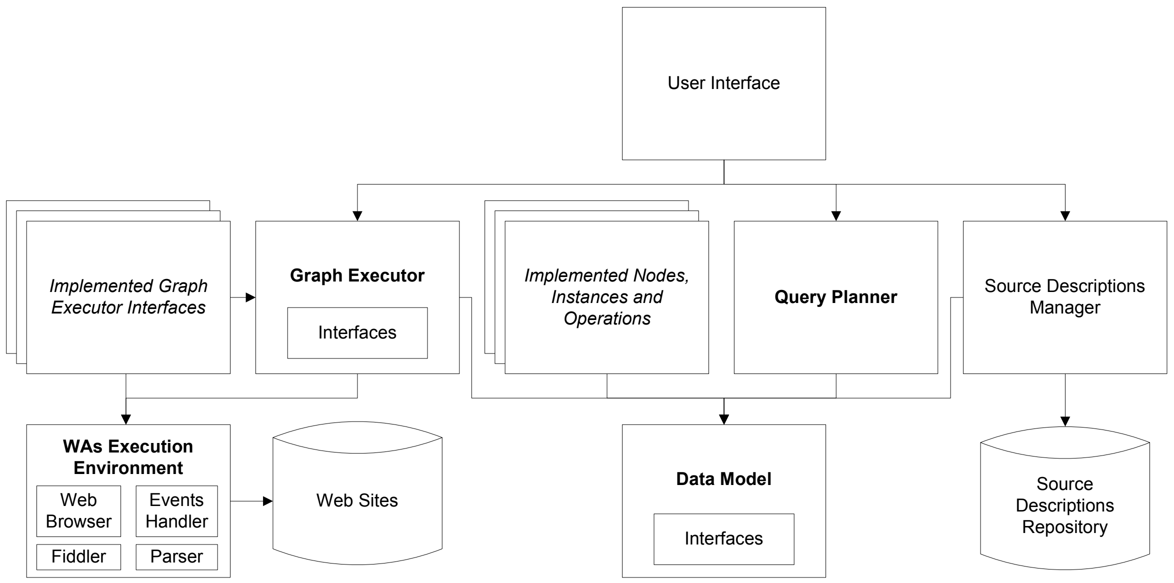

User Interface
Implemented Graph Executor Interfaces
Graph Executor
Interfaces
Implemented Nodes, Instances and Operations
Query Planner
Source Descriptions Manager
WAs Execution Environment
Web Browser
Events Handler
Fiddler
Parser
Web Sites
Data Model
Interfaces
Source Descriptions Repository

Figure 5.1: System Architecture

The second component, Graph Executor, described in Section 5.1.2, implements the algorithms described in Section 4.4. It also implements a dynamic programming approach to the instantiation of nodes using the same inputs as previously. Similarly to the Data Model, the Graph Executor abstracts some of its components in a form of interfaces. They correspond to the queue management algorithms described in Section 4.4.5 and to the data storing routines discussed in Section 4.4.7.

The third component, the Web Action Execution Environment, discussed in Section 5.1.3, is responsible for dealing with the complexities and diversity of Web actions and with different interpretations of their output. By using an embedded Web browser control, it enables the execution of both simple HTTP request-based Web actions, and Web actions relying heavily on user interface interactions. By combining a proxy, an event handling and alignment component, and a Web browser-based document parser, it provides a rich representation of a Web action's output, including string and object representation of the complete Web document, and of the part of the Web document modified by the Web action, as well as a binary representation of content transfered from the server.

Finally, the Query Planer component implements the algorithms discussed in Section 4.6. It combines a data extraction graph describing the Web site's navigation and a user query, in order to generate a completely executable data extraction graph corresponding to a query plan. While multiple implementations of a query planner may exist, in contrast with the Data Model and Graph Executor, we do not base this component on interfaces, as we believe that each implementation will use a different user query representation.

Our implementation of Query Planner follows the algorithms described in Section 4.6.3. It accepts a user query consisting of output node, projection attributes, selection equality constraints, optional ordering attribute and input values for one or more input nodes. It implements the query plan optimization techniques: the removal of unused input mappings and input mappings parts, the adding of filtering attribute sets, pass-trough slots and automatic generation of request capture nodes (see Section 5.2.3), and node merging by using the `INode` methods `CanMergeWithNode` and `MergeWithNode`.

Two other components present in Figure 5.1, namely User Interface and Source Descriptions Manager, discussed jointly in Section 5.1.4, are more important from the perspective of a user of our system than from the perspective of the information extraction process.

Finally, two stacks of components, marked in the architecture scheme by italics, correspond to different implementations of the Data Model and Graph Extractor interfaces. Nodes implemented in our prototype are discussed in Section 5.2, while the implementation of different Graph Executor algorithms are presented in Section 5.3.

5.1.1 Data Model

The Data Model component contains a few interfaces and classes that correspond to basic elements of our data extraction graph model. These interfaces and models are further used by the graph execution, query planning and source description component. This component is schematically depicted at Figure 5.2.

The first group of elements contained in the Data Model component consists of four interfaces: `INode`, `IInstance`, `IAtom` and `IInputMapping`. All instances of `INode` need to implement the `Instantiate` method and provide properties for accessing the node's `InputSlots`, `OutputSlots` and `Id` number. Each class implementing the `IInstance` interface needs to contain the `CreateAtom` method and give access to instance `CreatorNode` and automatically assigned, unique `Id` number. Each implementation of the `IAtom` interface has to possess the `Execute` method and `CreatorInstance`, `CreatorNode` and `Id` properties. Finally, IInuptMapping implementations should give access to `SourceNode`, `DestinationNode`, `SourceSlots` and `DestinationSlots` properties.

Moreover, as source descriptions are serialized to XML, each `INode` and `IInputMapping` implementation should contain the methods `ReadFromXmlNode` and `SaveToXmlNode`.

The second group of Data Model elements consists of classes `RecordSchema`, `Record`, `InputMapping` and `InputConfiguration` that provide the ready-to-use abstractions necessary for defining data extraction algorithms. `RecordSchema` consists of a list of `Attribute` elements (accessible via an iterator or by an index corresponding to attribute name), each of which has a `Name`, `TypeName` and `SemanticAnnotation` properties, as well as `FromString` method, capable of parsing string inputs into proper Attribute data type. `Record` has the following properties: `Schema`, automatically assigned, unique `Id` number (used for provenance tracking) and `this[attributeName]` property for accessing individual values by attribute name. `InputMapping` is a basic implementation of the `IInputMapping` interface that contains no extra data nor functionalities. Finally, `InputConfiguration` is a collection of input mappings, that ensures that all contained input mappings have the same destination node, are disjoint and cover all input slots of the destination node.

Data Model also contains the implementation of the data extraction graph in class `DataExtractionGraph`. It inherits from `BidirectionalGraph` (templated class from the QuickGraph library), using `INode` as node data type and `IInputMapping` as edge data type. Apart

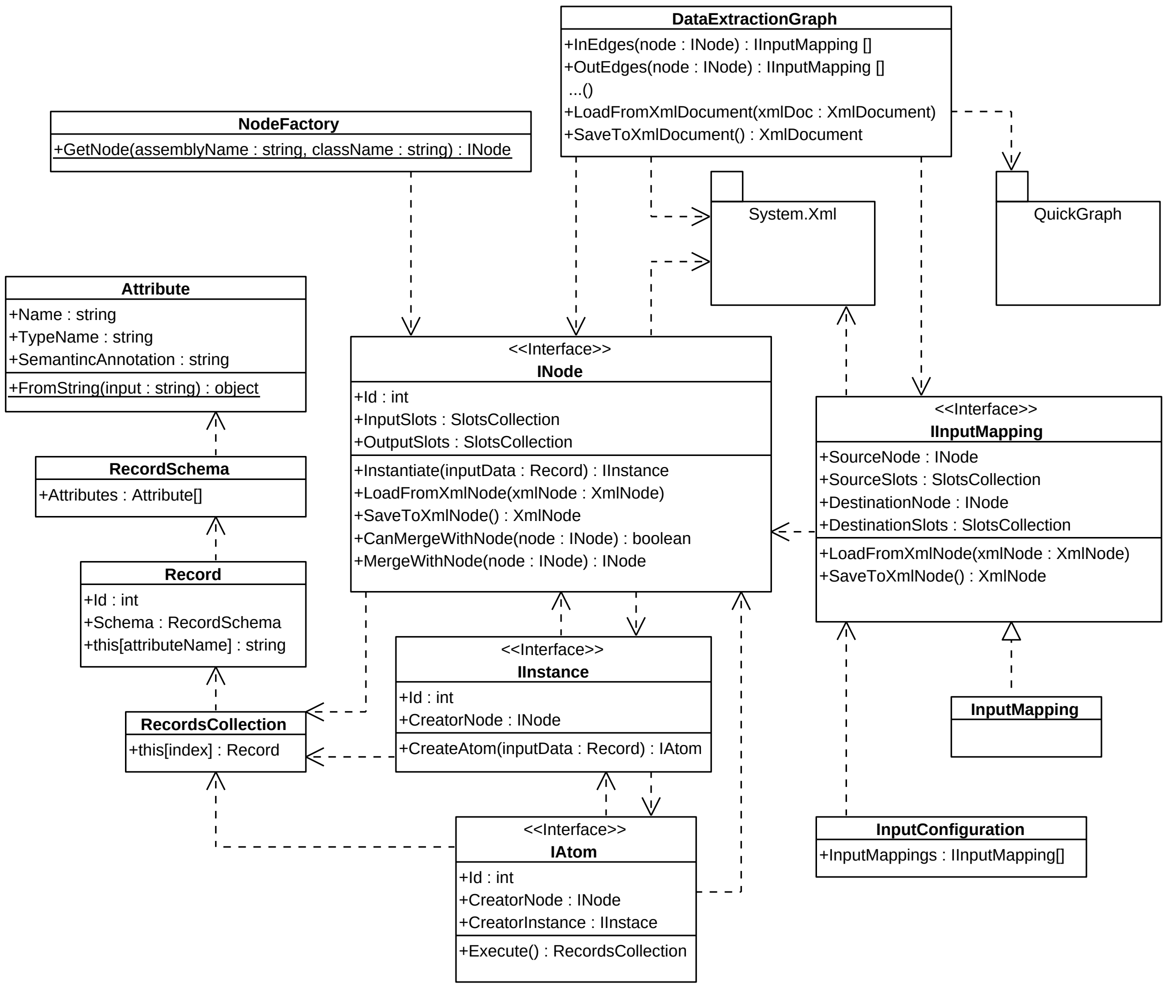

DataExtractionGraph
+InEdges(node : INode) : IInputMapping []
+OutEdges(node : INode) : IInputMapping []
...()
+LoadFromXmlDocument(xmlDoc : XmlDocument)
+SaveToXmlDocument() : XmlDocument
NodeFactory
+GetNode(assemblyName : string, className : string) : INode
System.Xml
QuickGraph
Attribute
+Name : string
+TypeName : string
+SemantincAnnotation : string
+FromString(input : string) : object
<<Interface>>
INode
+Id : int
+InputSlots : SlotsCollection
+OutputSlots : SlotsCollection
+Instantiate(inputData : Record) : IInstance
+LoadFromXmlNode(xmlNode : XmlNode)
+SaveToXmlNode() : XmlNode
+CanMergeWithNode(node : INode) : boolean
+MergeWithNode(node : INode) : INode
<<Interface>>
IInputMapping
+SourceNode : INode
+SourceSlots : SlotsCollection
+DestinationNode : INode
+DestinationSlots : SlotsCollection
+LoadFromXmlNode(xmlNode : XmlNode)
+SaveToXmlNode() : XmlNode
RecordSchema
+Attributes : Attribute[]
Record
+Id : int
+Schema : RecordSchema
+this[attributeName] : string
<<Interface>>
IInstance
+Id : int
+CreatorNode : INode
+CreateAtom(inputData : Record) : IAtom
RecordsCollection
+this[index] : Record
InputMapping
<<Interface>>
IAtom
+Id : int
+CreatorNode : INode
+CreatorInstance : IInstace
+Execute() : RecordsCollection
InputConfiguration
+InputMappings : IInputMapping[]

Figure 5.2: UML Class Diagram of Data Model Component

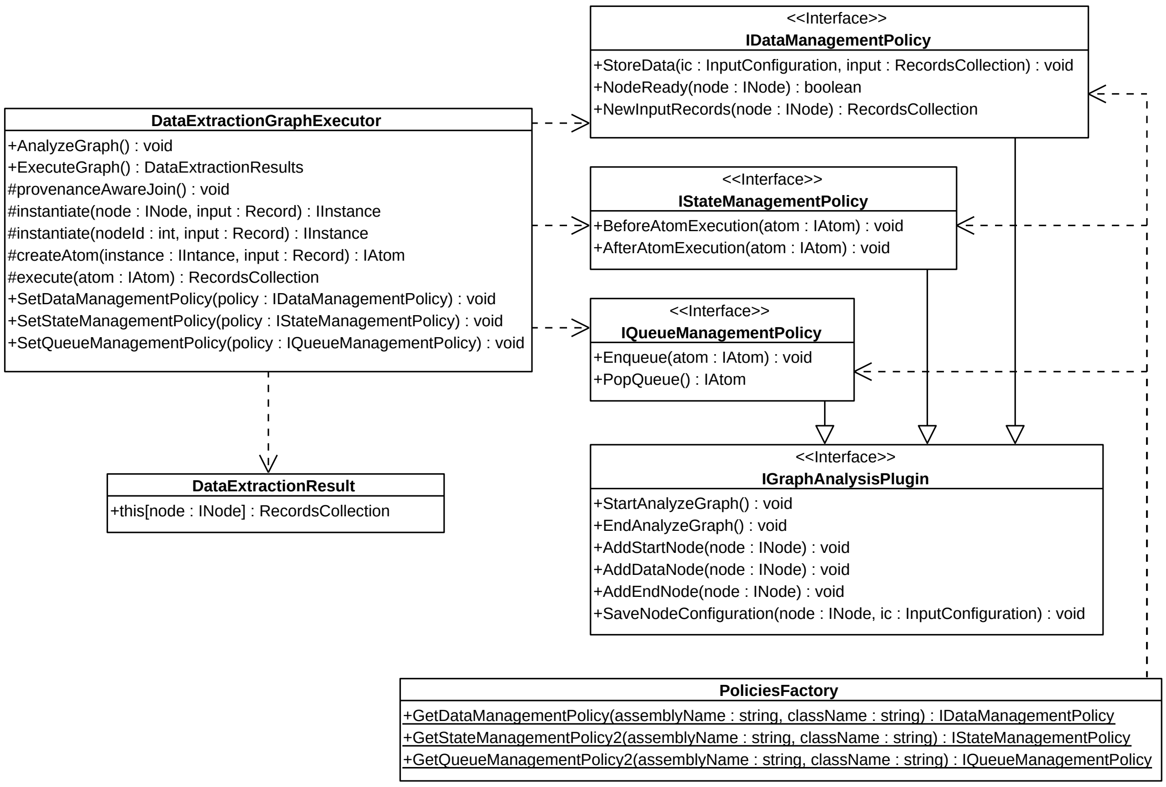
from a number of methods inherited from `BidirectionalGraph` (including, among others, methods for accessing all incoming and outgoing edges of a node), it contains methods for serializing the data extraction graph into, and deserializing it from XML format.

The final component is a support factory class `NodeFactory` that, provided with a name of assembly and class name of a node, uses the .NET Reflection mechanism to load the assembly and instantiate the class. It is used internally by the `DataExtractionGraph` deserialization mechanism, thus enabling the development of pluggable assemblies containing new types of nodes without the need to modify or recompile the main part of the implemented system.

5.1.2 Data Extraction Graph Execution Component

The Graph Execution component, schematically depicted in Figure 5.3, consists of the `Data-ExtractionGraphExecutor` class, a number of interfaces used to implement queue management, data storing, state recreation and state recognition *policies* (i.e. pluggable small components implementing key subroutines of EXECUTEGRAPH algorithm), as well as a few implementations of these interfaces.

`DataExtractionGraphExecutor` implements a few essential methods related to graph execution, corresponding to algorithms described in Section 4.4. These methods include `AnalyzeGraph`, `ExecuteGraph`, `ProvenanceAwareJoin`, as well as variants of `Instantiate`, `CreateAtom` and `Execute` methods that accept respectively a node, an instance and an atom as parameter. It also implements a set of static methods for managing policies used at specific moment of time.



<<Interface>>
IDataManagementPolicy
+StoreData(ic : InputConfiguration, input : RecordsCollection) : void
+NodeReady(node : INode) : boolean
+NewInputRecords(node : INode) : RecordsCollection
DataExtractionGraphExecutor
+AnalyzeGraph() : void
+ExecuteGraph() : DataExtractionResults
#provenanceAwareJoin() : void
#instantiate(node : INode, input : Record) : IInstance
#instantiate(nodeId : int, input : Record) : IInstance
#createAtom(instance : IIntance, input : Record) : IAtom
#execute(atom : IAtom) : RecordsCollection
+SetDataManagementPolicy(policy : IDataManagementPolicy) : void
+SetStateManagementPolicy(policy : IStateManagementPolicy) : void
+SetQueueManagementPolicy(policy : IQueueManagementPolicy) : void
<<Interface>>
IStateManagementPolicy
+BeforeAtomExecution(atom : IAtom) : void
+AfterAtomExecution(atom : IAtom) : void
<<Interface>>
IQueueManagementPolicy
+Enqueue(atom : IAtom) : void
+PopQueue() : IAtom
DataExtractionResult
+this[node : INode] : RecordsCollection
<<Interface>>
IGraphAnalysisPlugin
+StartAnalyzeGraph() : void
+EndAnalyzeGraph() : void
+AddStartNode(node : INode) : void
+AddDataNode(node : INode) : void
+AddEndNode(node : INode) : void
+SaveNodeConfiguration(node : INode, ic : InputConfiguration) : void
PoliciesFactory
+GetDataManagementPolicy(assemblyName : string, className : string) : IDataManagementPolicy
+GetStateManagementPolicy2(assemblyName : string, className : string) : IStateManagementPolicy
+GetQueueManagementPolicy2(assemblyName : string, className : string) : IQueueManagementPolicy

Figure 5.3: UML Class Diagram of Graph Execution Component

There are three key interfaces, corresponding to policies, that are defined in this component. `IQueueManager`, abstracting queue management algorithms described in Section 4.4.5, consists of two key methods: `Enqueue` and `PopQueue`. The `IDataManager` interface contains three methods: `StoreData`, `NewInputRecords` and `NodeReady`, corresponding to algorithms described in Section 4.4.7. The `IStateManager` interface abstracts operations related to state tracking, recognition and recreation. It defines two methods: `BeforeAtomExecution`, called by `DataExtractionGraphExecutor` method `execute` just before an atom is executed, and `AfterAtomExecution` that is executed just after atom execution.

Moreover, all these interfaces inherit from a special interface `IGraphAnalysisPlugin` that defines a few methods called during execution of the `AnalyzeGraph` method of the `DataExtractionGraphExecutor` class. `StartAnalyzeGraph` is called when the `AnalyzeGraph` method starts, and `EndAnalyzeGraph` is called just before this method finishes. Three methods `AddStartNode`, `AddDataNode` and `AddEndNode` are called directly after a node is added to the collections of start nodes, data nodes and end nodes, respectively. Finally, whenever an input configuration is saved, the method `SaveNodeConfiguration` is called.

Finally, similarly as the in case of Data Model, the Graph Execution component includes a static `PoliciesFactory` capable of instantiating policies implementations for dynamically loaded .NET assemblies.

5.1.3 Web Actions Execution Environment

One of the key components of our solution is responsible for executing Web actions and capturing associated content both at the levels of encoded content and user content.

As we described in Chapter 3, some previous solutions used Web browsers to enable complex navigation and JavaScript execution (Denodo) or for the visual parsing of loaded content (DEPTA). While an embedded Web browser is not be the best performing solution, it resolves multiple technical problems, such as handling different types of HTTP and HTTPS requests, dealing with redirects, Cookies and content caching, enabling complex JavaScript and AJAX Web actions, interpreting bad quality HTML, and makes it possible to work with documents represented as DOM trees, a set of boxes or a set of visual items.

Working with Individual HTTP Requests

For these all reasons, we decided to use an embedded Internet Explorer control as the base of our Web actions execution environment. To avoid need of direct work with Internet Explorer COM objects and interfaces, we used the CsExWB .NET control that wraps most of them. The minimal version required by our solution is Internet Explorer 8.

Using an embedded Web browser makes it possible to execute complex Web actions and to capture actual user content that may be created by combining multiple HTTP requests and client-side code execution. However, Internet Explorer (similar to other browsers), gives very limited access to the encoded content of individual HTTP requests.

For that reason, apart from the embedded Web browser we decided to use a specialized Web proxy to capture the encoded content of all requests corresponding to Web actions. Because of its simple integration with C# applications, we used the .NET class library version of Fiddler Web Debuger[1], called FiddlerCore[2], as a content-capturing proxy. As FiddlerCore is embedded in the same Windows process as the CsExWB control, it is possible to run Fiddler as a process-wide rather than system-wide proxy. Thanks to it, only requests coming from a specific instance of CsExWB are captured by the specific instance of FiddlerCore.

FiddlerCore is used not only for capturing encoded content, but also for caching complete requests for specific responses. This enables this component to act as a server emulator, without need of the repetitive sending of requests to the server (thus limiting overhead and the risk of server-side navigation state side-effects).

More details on how CsExWB and FiddlerCore events are connected to acquire maximum information about each executed Web action can be found in Appendix E.

[1]See: `http://www.fiddler2.com/`.

[2]Available at: `http://www.fiddler2.com/core/`.

Web Actions Output Representations

The output of a Web action consists of three parts. The first of them corresponds to the complete document loaded into Web browser. While it can actually reflect the output of multiple consecutive Web actions (e.g. when a few consequential AJAX Web actions load additional content into the main document), this part of Web action output is typically the most useful for data extraction.

In our model, the complete document is represented in four distinct ways, which are used by different data extraction rules and Web actions. Conversion of Web browser objects into these representation is performed by simple parser that is a part of Web actions execution environment component.

The first representation consists of an internal Web browser DOM representation. It has an advantage in that it enables data extraction using structure-based languages such as XPath, and the execution of Web actions based on DOM objects (e.g. simulating mouse click or drag and drop events), however, it requires that the document is loaded into a Web browser. It means that values from this representation are no longer available after a new document is loaded into Web browser.

The second and third representations are string-based and consist of the source code of a Web page and the full text content of a Web page, respectively. Both of them can be used for information extraction by string extraction languages such as regular expressions.

Finally, the document can be represented as a materialized DOM model. As compared with native Web browser representation, it has an advantage in that data can be accessed even after Web browser content changed. However, it does not enable the execution of Web actions on HTML DOM objects. In our prototype, we used .NET framework XML DOM classes to enable such representation.

The second part of a Web action's output consists of data captured at proxy level. It consists of a string representation of all data captured at proxy level with all related request and response headers.

The final part of Web action output consists of all fragments of the Web document that were modified by the Web action. It has a double representation. The first consists of concatenations of the source codes and texts of individual modifications. The second consists of a list of XPath expressions that unambiguously identify the modified data. A richer representation of modifications (e.g. detailed tracking of the scope and type of modifications), belongs to our future development plans.

Politeness Policy

One of the important aspects of interacting with data-intensive Web sites is related to limiting the number of requests issued per unit of time. As in the crawler industry, we will call a set of rules that define how requests should be limited the *politeness policy*.

This is important for at least two reasons: firstly, it handles the risk of causing "denial-of-service" problems with a Web site and thus is important from the perspective of respecting other Web site users. On the other hand, systems that leave irregular pauses between consecutive requests cannot be easily distinguished from users accessing Web sites. As a result, chances of cloaking (i.e. a system being served different content than normal users) decrease.

In our solution, the politeness policy is strictly connected with the Web actions execution environment. It is defined by three parameters: a number of allowed parallel connections to a single host, as well as minimal and maximal time between two consecutive navigation actions.

The number of parallel connections is enforced in the FiddlerCore `BeforeRequest` event. If the number of connections that are occurring at a given moment is maximal, any following request will be suspended in this event until the number of connections lowers (i.e. until at least one `AfterSessionComplete` event is called). It is to be noted that the embedded Web browser has a mechanism of limiting a number of parallel connections of its own. Thus, this parameter is only useful if for some reason it should be lower than the Web browser's.

The time between two navigation actions is enforced in the `execute` method of the `DataExtractionGraphExecutor` class. For each navigation action, a random number is chosen from the interval of minimal and maximal time between navigation actions, and upon each execution of the Web action atom that clearly interacts with the HTTP server (e.g. explicitly corresponding to the NavigateToURL action); it is checked if at least chosen time has elapsed since previous non-canceled `BeforeNavigate2` event. If it is not the case, execution is suspended for the remaining time. Moreover, the same verification and suspension mechanism is embedded in the `BeforeNavigate2` event in order to handle Web actions that interact with the HTTP server in a less explicit way (e.g. some part of Web actions consisting in clicking Web elements).

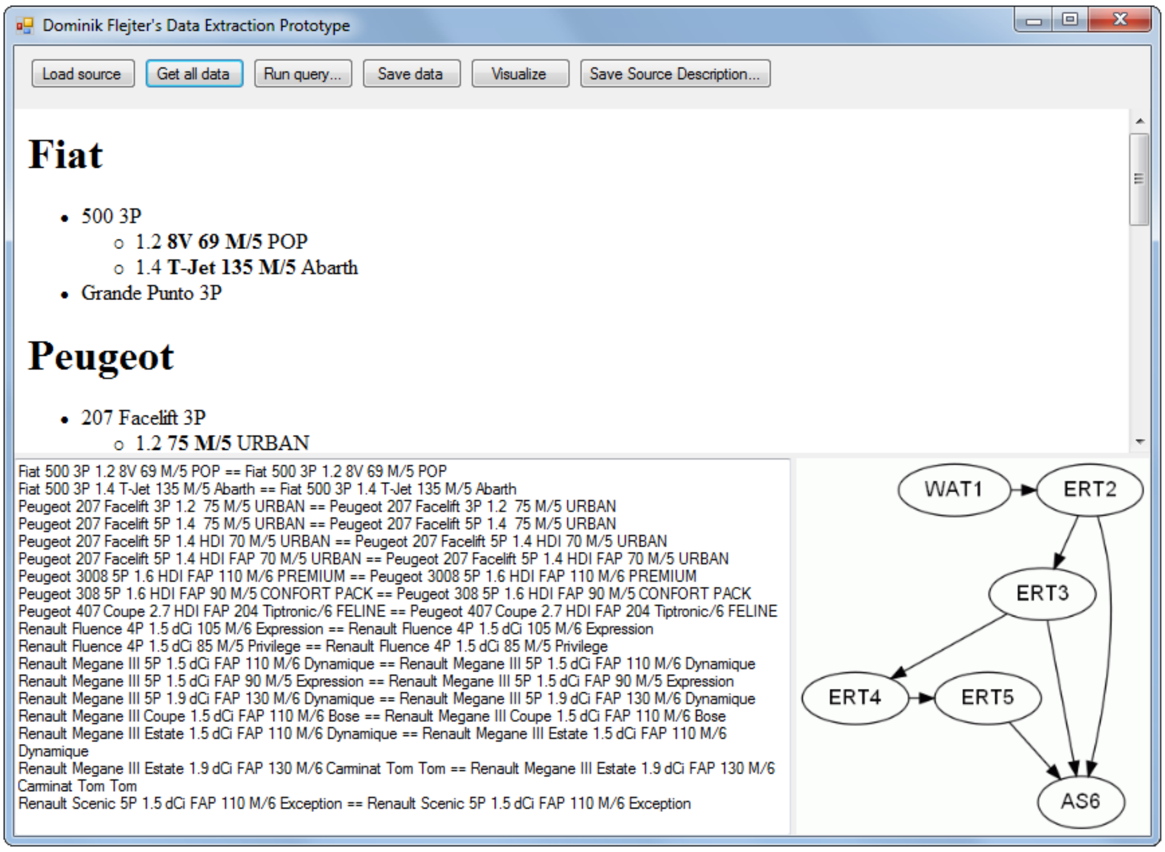
5.1.4 User Interface

On the top of our prototype data extraction model we developed a simple user interface, which allows users to load Web site description, visualize it, define the user query, perform data extraction and save extracted data. It also provides a continuous preview of currently processed Web documents.

The screen shot of user the interface of our prototype is depicted in Figure 5.4. The application consists of four main elements: toolbar, Web browser panel (the top part of the window), data preview panel (left bottom part of the window) and data extraction graph visualization panel (right bottom part of the window).

The toolbar contains six buttons, corresponding to key functionalities of the application:

- The **Load source...** button makes it possible to choose an XML file containing a Web site's description, loads the description and initializes the user query to "get all data".
- The **Set query...** button enables modification of the current query. After a user defines the new query, query planning generates the corresponding data extraction graph.
- The **Save query...** button saves the current user query in XML format.
- The **Load query...** button loads a previously saved user query and runs the query planning procedure.
- The **Extract data** button performs data extraction by executing a graph corresponding to the query plan. Statistics on the extracted data are shown in the data preview panel.



Fiat

- 500 3P
  - 1.2 **8V 69 M/5** POP
  - 1.4 **T-Jet 135 M/5** Abarth
- Grande Punto 3P

Peugeot

- 207 Facelift 3P
  - 1.2 **75 M/5** URBAN

```
Fiat 500 3P 1.2 8V 69 M/5 POP == Fiat 500 3P 1.2 8V 69 M/5 POP
Fiat 500 3P 1.4 T-Jet 135 M/5 Abarth == Fiat 500 3P 1.4 T-Jet 135 M/5 Abarth
Peugeot 207 Facelift 3P 1.2  75 M/5 URBAN == Peugeot 207 Facelift 3P 1.2  75 M/5 URBAN
Peugeot 207 Facelift 5P 1.4  75 M/5 URBAN == Peugeot 207 Facelift 5P 1.4  75 M/5 URBAN
Peugeot 207 Facelift 5P 1.4 HDI 70 M/5 URBAN == Peugeot 207 Facelift 5P 1.4 HDI 70 M/5 URBAN
Peugeot 207 Facelift 5P 1.4 HDI FAP 70 M/5 URBAN == Peugeot 207 Facelift 5P 1.4 HDI FAP 70 M/5 URBAN
Peugeot 3008 5P 1.6 HDI FAP 110 M/6 PREMIUM == Peugeot 3008 5P 1.6 HDI FAP 110 M/6 PREMIUM
Peugeot 308 5P 1.6 HDI FAP 90 M/5 CONFORT PACK == Peugeot 308 5P 1.6 HDI FAP 90 M/5 CONFORT PACK
Peugeot 407 Coupe 2.7 HDI FAP 204 Tiptronic/6 FELINE == Peugeot 407 Coupe 2.7 HDI FAP 204 Tiptronic/6 FELINE
Renault Fluence 4P 1.5 dCi 105 M/6 Expression == Renault Fluence 4P 1.5 dCi 105 M/6 Expression
Renault Fluence 4P 1.5 dCi 85 M/5 Privilege == Renault Fluence 4P 1.5 dCi 85 M/5 Privilege
Renault Megane III 5P 1.5 dCi FAP 110 M/6 Dynamique == Renault Megane III 5P 1.5 dCi FAP 110 M/6 Dynamique
Renault Megane III 5P 1.5 dCi FAP 90 M/5 Expression == Renault Megane III 5P 1.5 dCi FAP 90 M/5 Expression
Renault Megane III 5P 1.9 dCi FAP 130 M/6 Dynamique == Renault Megane III 5P 1.9 dCi FAP 130 M/6 Dynamique
Renault Megane III Coupe 1.5 dCi FAP 110 M/6 Bose == Renault Megane III Coupe 1.5 dCi FAP 110 M/6 Bose
Renault Megane III Estate 1.5 dCi FAP 110 M/6 Dynamique == Renault Megane III Estate 1.5 dCi FAP 110 M/6
Dynamique
Renault Megane III Estate 1.9 dCi FAP 130 M/6 Carminat Tom Tom == Renault Megane III Estate 1.9 dCi FAP 130 M/6
Carminat Tom Tom
Renault Scenic 5P 1.5 dCi FAP 110 M/6 Exception == Renault Scenic 5P 1.5 dCi FAP 110 M/6 Exception
```

Figure 5.4: Screen Shot of User Interface of Our Prototype

During data extraction all the Web actions are visible in the Web browser panel.

- The **Save data...** button saves extracted data in a CSV or XML file format.
- The **Visualize graph** button generates a visual representation of the data source data extraction graph and loads it into graph visualization panel. Visualizations are generated by using QuickGraph visualization features and the GraphViz dot application.
- The **Visualize query** button is similar to the Visualize graph, but generates a representation of graph corresponding to the current user query.
- The **Save source description...** button generates a visualization of the Web site's data extraction graph and LaTeXtables describing the graph's nodes and input mappings.
- The **Run tests...** button automatically extracts data for multiple scenarios and compares them with reference data records. Each scenario specifies the used Web site description, used user query, as well as a set of reference data that should be compared with data extracted from the source. A list of scenarios is loaded from an XML file, and, iteratively, data extraction is performed for each of these scenarios. Extracted data are automatically saved into XML files, and compared with reference data. A log with information on each accomplished scenario is generated.

Our prototype uses a few formats of XML files. One format is used to represent the Web site's data extraction graphs. It consists of elements corresponding to individual nodes and input mappings. Each node specifies its .NET assembly name and class name, which are passed to the `GetNode` method of the `NodesFactory` class. Each node's XML element specifies also meta-data related to state management. The XML element corresponding to a specific node contains different children elements, depending on node type. These elements are interpreted directly by implementation of the `LoadFromXmlNode` of a specific node type.

The second type of XML file is used for saving data extraction output, as well as for representing reference data. In this format, the root element contains the elements `schema` and `records`. Schema is defined by a list of attribute nodes, each of which specifies attribute name, .NET data type and semantic annotation. The `records` element contains a list of records, each of which specifies the value for each attribute as a separate XML element.

The third type of XML files is used to represent user queries. It consists of the root XML tag `query` with attributes corresponding to the identifier of output node, used ordering attribute and ordering direction, as well as embedded lists of query constraints (`selection` element containing a list of `constraint` elements), inputs to input nodes (`input` elements embedded in `inputs` tag) and a list of attributes to be returned (stored in `projection` tag).

Finally, we also use a simple list encoded in an XML file to specify the list of scenarios to be executed by the "Run tests..." button. In this format, the root element contains a list of `scenarios` elements, each of which specifies the name of the Web site description XML file, the name of the file containing the reference data, and (optionally) the name of the file that contains the user query to be used.

Thanks to the extensible character of the XML data format, we were also able to add to each node of data extraction graph XML file information on its impact or dependency on parts of the navigation state. In the case of state-aware descriptions of the Web sites model, three extra sets of tags are added to the XML file.

The first, contained in the `stateparts` tag, consists of any number of `state` tags corresponding to parts of the navigation state. We adopted a convention of embedding parts one inside another by using a point character (as in many programming languages), and putting all parts within `cs` (for client-side) or `ss` (for server-side) collections. Thus, for example, `cs.doc.f1` may correspond to frame `f1` of the current Web document, while `ss.cart` may correspond to the shopping cart.

The second and third set of additional tags connect nodes to parts of the navigation states they modify and depend on, respectively. In the case of modifications, used attributes are `nodeId`, `partId` and modification type (one of: reset, cumulate, replace, partialReplace, completeReplace; see Section 4.5). In the case of state-dependency, the pair of node id number and navigation part id number is used.

5.2 Implemented Nodes

In this section we describe all types of data extraction graph nodes that were implemented as part of our proof-of-concept prototype. We also briefly discuss examples of other nodes that can be implemented to further increase the flexibility of the developed solution.

5.2.1 Web Actions Templates

The first group of nodes implemented in our model contains various Web actions templates, that simulate different user behaviors.

While they are diverse, they share the same outputs slots that correspond to different representations of a Web site's output, as discussed in Section 5.1.3.

Web Actions Templates Output Slots

The first group of output slots contains different representations of a complete Web document. `innerHTML` output slots represents a complete source code of current Web document. If it contains frame or iframe HTML elements, the source code of documents they contain is placed between the opening and end (i)frame tags. `innerText` output slot corresponds to `innerHTML` with all HTML tags removed. `document` output slots gives access to native Web browser representation of the HTML document root node. Finally, `element` output slots corresponds to root element of the materialized XML representation of a Web browser's DOM. Both in the case of `document` and `element`, the elements contained in the root element of any (i)frame document are added as (i)frame element children elements.

The second group of output slots represents encoded content, captured at proxy-level. `capturedContent` provides access to all captured text files that correspond to a Web action (e.g. main HTML file, HTML files loaded into frames, auxiliary files such as styles and script files) as a single string containing request and response headers, POST data (if applicable) and complete returned content. Beginning and end of request headers, POST data, response headers, and file content is marked by an unique marker sequence. `capturedContentInfo` provides a list of requests contained in `capturedContent` with all corresponding start and end markers.

The final group of output slots represent modifications introduced into a Web document by Web action. `newInnerHTML` represents the source code of the modified parts of documents and `newInnerText` provides the same source code with all tags removed. As in the case of `capturedContent`, we use unique markers for separating multiple modifications one from another. Finally, the `modifications` output slot provides a list of modifications as a single string containing tab-separated values for pairs of unique markers and their corresponding XPath expressions. Markers can be further used as a parameter to an ERT that extracts individual modifications, which can be further used to extract data just from a specific part of the extracted data. Similarly, XPath expressions can be used as input to an XPath ERT, to obtain corresponding elements as objects. These objects can be further used as input to some complex Web actions (such as clicking an object).

Implemented Web Actions

All Web action templates implemented as nodes in our prototype generalize a few types of Web action. In this section we discuss these types of Web action and their atom creation input slots. However, Web action template nodes that use them may have additional instantiation input slots.

Web actions implemented in our prototype can be divided into a few groups. The first group consists of a few types of navigation Web actions. The basic `NavigateToURL[url]` Web action corresponds to entering an URL address in a Web browser's address bar, i.e. it performs top-level navigation to the specified URL. This Web action can be performed at any moment and has no atom creation input slots.

The similar Web action `NavigateFrameToURL[url]` a performs navigation action in a specific frame, defined by one of three alternative input slots: `FrameName` for identifying a frame by its name (simple but may be ambiguous), `FramePath` for specifying the full path of the names of embedded frames, and `FrameObject` for providing a reference to a frame or iframe object as returned, for example, by an XPath extraction rule.

We also implemented a few actions that mimic essential Web browser functionalities. The `NavigateToBlank` Web action loads a blank Web page into the Web browser, the `Navigate-ToEmptyState` Web action returns to a blank Web page and removes all Cookies and cache content, and `NavigateBack[steps]` and `NavigateForward[steps]` simulate clicking back or forward button (with the number of clicks determined by steps parameter).

The second group of Web actions enables the creation of complex HTTP requests. `Navi-gateViaPostRequest[url, postData]` is analogous to `NavigateToURL`, but enables POST requests. `NavigateFrameViaPostRequest[url, postData]` performs a similar action in a frame specified by the `FrameName`, `FramePath` or `FrameObject` input slot.

Two other Web actions in this group are `NavigateViaFormSubmition` and `SubmitForm`. Both of them are related to handling HTML forms. Both of them have two alternative input slots, `FormName` and `FromObject`, for identifying the concerned form, as well as input slots corresponding to all form fields. The difference is that in the case of the first Web action, the request is built manually as if the form was used, while in the second case, form fields will be filled in and form submission will be simulated. Thus, if a given form has some JavaScript onsubmit event handler, it will be executed in case of the second Web action, but will not be in the case of the first one.

The third group of Web actions is related to simulating some user actions on specific HTML elements. For example, `ClickHtmlElement` simulates a click on an object provided to its `object` input slot. The actual behavior of the Web browser when the Web action is executed will depend on the type of clicked HTML element, on its JavaScript `onclick` event handler and the boolean value the handler returns (with `false` canceling the normal click behavior of a given element). This group contains a few more mouse and keyboard oriented Web actions that we do not describe in detail here.

One more important Web action belonging to this group is `RunEventHandler[handler-Name]`, which simulates running the `on*` event handlers (such as `onclick`, `onload`, `onfocus`) of a specific HTML element passed as input to the `object` input slot.

The final group contains `RunJavaScript[code]` and `RunJavaScriptInFrame[code, frame]` Web actions, which enable execution of any JavaScript code in the top-level and frame document respectively. These two Web actions provide a procedural extension to our declarative data extraction framework, and can be used, for example, to run some page-specific JavaScript functions that are not directly triggered by any on* event.

5.2.2 Extraction Rules

XPath

We provide two distinct implementations of XPath. They differ with respect to their input and output values. The first implementation works with the materialized XML representation of Web document. The second one works directly with the Web browser's DOM objects.

The first implementation has a single input slot `element` that accepts input, for example, from a Web action's output slot `element` or from another XPath rule using this implementation of XPath language. It provides for each extracted element three values: `innerHTML` corresponding to source code stored within the chosen element, `innerText` providing a plain text representation of the content of the chosen element, and `element` representing the XML DOM element that can be used as input content for other XPath extraction rules.

Internally, this implementation uses .NET framework methods included in the `System.Xml` and `System.Xml.XPath` assemblies. This implementation has much better performance than the second one. However, the objects it returns cannot be used as input to Web actions executed on native Web browser DOM objects (such as `ClickHtmlElement`).

The second implementation has one input slot `object`. It can be provided with values returned by the `document` output slot of Web actions or from the object output slot of other XPath rules using the second implementation. Its three output slots are `innerHTML`, `innerText` and `object`. The last of them corresponds to native Web browser DOM object(s) chosen by the rule.

We developed this version of XPath handling by creating our own implementation of the .NET `XPathNavigator` class[3]. The main advantage of this implementation is that it returns objects that can be input to Web actions such as `ClickHtmlElement`. Unfortunately, it is significantly slower than the first implementations and requires that Web browser state be recreated to perform data extraction.

Regular Expressions

Regular expressions in our model have multiple output slots corresponding to all parenthesized subpatterns (with slot "Subpattern0" corresponding to complete matched content, and all other subpatterns named "Subpattern$X$", where X is the subpattern number).

All extraction rule templates expressed as regular expressions have at least one input slot corresponding to the content on which regular expression will be executed. All other input slots correspond to named parts of the original regular expression definition that should be replaced with properly escaped received values.

Regular expressions accept almost any type of object as input. However, if the passed object is not a string, data conversion is enforced by calling the object's `ToString` method.

Our implementation of regular expressions is based on the .NET System.Text.RegularExpressions library, which implements regular expressions syntax and features almost completely

[3]While developing this code, we used some parts of the code presented by Victor Zverovich at `http://www.codeproject.com/KB/cs/htmlxpath.aspx`, accessed on 03.11.2009.

compatible with those of Perl 5[4]. For convenience, we added to the original .NET regular expressions syntax the possibility to specify modifiers (called RegexOptions in .NET) directly in the pattern, similarly as in the PCRE[5] library.

5.2.3 Request Capture Template Node

In many dynamic Web sites, user actions trigger JavaScript code that generates a simple navigation event (e.g. by changing `window.location` property). As we discussed previously, such Web actions are state-dependent on current Web browser content. Therefore, if there are many Web actions of this type in a single page, execution of each of them requires Web browser's state recreation. As a result, such operation may require important time and transfer resources.

To significantly improve performance in the discussed scenario, we introduced a special type of instance called request capture. It combines some properties of Web actions and extraction rules.

Request capture is defined on top of the Web browser state-dependent Web action (called *contained Web action*). When it is executed, it internally executes the Web action. However, when `BeforeNavigate2` event generated by JavaScript code is triggered, sent requests (URL and POST data, if applicable) are captured and the navigation is canceled. The captured request details are made available via output slots, and can be further used as input for state-independent `NavigateToURL` or `NavigateViaPostRequest` Web actions. As a result, not a single state recreation operation is required.

Request capture instances are generalized into request capture template nodes, which may have some node instantiation input slots for modifying the specification of contained Web action.

5.2.4 Other Nodes

Two other nodes implemented in our prototype are attribute sets and user input nodes. Their implementation is trivial and strictly follows the definitions provided in Sections 4.2.7 and 4.6.1.

5.3 Implemented Algorithms

5.3.1 Queue Management Algorithms

In our solution we created three distinct implementations of `IQueueManagementPolicy`, corresponding to algorithms described in the previous chapter. The two first solutions implement basic depth-first and breadth-first traversal of the data extraction graph, as described in Section 4.4.5. They both use a very simple representation of queue as an instance of the .NET `List<IAtom>` class. The `PopQueue` method of these implementations returns the item stored

[4]See: `http://msdn.microsoft.com/en-us/library/hs600312{%}28VS.71{%}29.aspx`, accessed on 12.12.2010.

[5]Perl-Compatible Regular Expressions, see: `http://www.pcre.org/`.

at index 0 and removes it from the queue by invoking the `RemoveAt(0)` method. These two implementations differ by the way they treat new atoms provided to the `Enqueue` method. In the case of the depth-first version, new atom a is added as the first item of the queue by invoking the `InsertAt(0, a)` method, while the breadth-first implementation adds new atoms at the end by invoking the `Add(a)` method.

The third solution is an implementation of the state-aware queue management algorithm proposed in Section 4.5.5. Both queues used by this version of algorithm are implemented as `List<IAtom>` instances. Both `PopQueue` and `Enqueue` implementation strictly follow the algorithms presented in the previous chapter. It is to be noted that this implementation is strictly dependent on the state annotation of graph nodes. Thus, if state annotation is not present in the Web site description XML file, both methods of this implementation throw exceptions.

5.3.2 State Management Approaches

As shown in Figure 5.3, any implementation of state management policy consists of three methods: `AfterAtomCreation`, `BeforeAtomExecution` and `AfterAtomExecution`. The first of these is used to register the navigation state at the moment when an atom is created. The second is responsible for preparing th graph or other data structures for execution of a specific atom. Finally, the third one can be used to register information on the current navigation state to be used for future states execution, as well as to clean up undesired side-effects of atom execution (if applicable).

We have created two implementations of `IStateManagementPolicy`. The first one consists of three empty functions (i.e. providing no support for state management), and is used together with non-state-aware queue management methods (depth-first and breath-first implementations of queue management policy).

The second implementation of this interface is state-aware and strictly dependent on the state annotation of graph nodes. It uses state annotations to identify sequences of Web action atoms that need to be replayed in order to recreate the proper navigation state.

This implementation keeps track of navigation state evolution by using three additional data structures. Firstly, the sequences of executed atoms are saved in the form of one tree of navigation states per each part of navigation state modeled in state annotation. After the execution of an atom, the state annotation of its creator node is checked for parts of the navigation state that are modified directly (explicitly described in annotation) or indirectly (parts included in modified parts of the navigation states). For each of these parts, the atom is added as a direct child of the current atom in the corresponding tree. When the previous navigation state is recreated (see below), the current node of each tree is updated.

Secondly, the sequence of all executed atoms is saved for future reference. Finally, after the creation of the atom, the position in each tree is saved for it in a hash table structure.

If an atom depends on some part(s) of the navigation state, the state management policy verifies if it has been modified since the atom was instantiated. It is done by analyzing the annotation of the creator nodes of all atoms executed after the navigation state registered at the moment of atom creation. If the state needs to be recreated, we look for the shortest possible sequence of atoms that includes a reset or complete replacement of all needed parts of navigation state, and all subsequent atoms that modify at least one of the needed navigation parts.

This implementation assumes that all atoms that modify some state are Web actions. The used replay approach depends on whether state that needs to be recreated is client-side or server-side. In the case of a client-side navigation state, the sequences of Web actions is replayed in the Web browser with FiddlerCore configured to return cached content instead of sending the request to the server. In the case of a server-side navigation state, the sequence of HTTP requests is replayed outside of the Web browser by using the .NET wrapper[6] of the libcurl multiprotocol file transfer library[7]. Finally, if both server-side and client-side states need to be recreated, the replay is performed in the Web browser, and the FiddlerCore cache mode is turned off.

5.3.3 Nodes Merger Algorithms

We implemented two groups of node merger algorithms. The first of them is responsible for incorporating data filtering into extraction rule templates, i.e. for merging ERTs with data filtering attribute sets. The second group enables the merging of two ERTs of the same type.

Data filtering incorporation was implemented for special cases of both regular expressions and XPath extraction rule templates. In the case of regular expressions, incorporation is possible only if subpattern(s) that correspond to attribute(s) that are used in filtering do not contain any embedded subpatterns. Thus, out of the cases presented in Example 4.6.5, only the first one would be handled, and for the second one, the `CanMergeWithNode` method would return false.

In the case of XPath ERTs, we enable incorporation of data filtering into ERT only if XPath expression does not use any alternative paths and the last element of the used XPath expression does not have any predicates defined. Thus, cases presented in Example 4.6.4 and the first case from Example 4.6.6 are handled, while all other examples from Example 4.6.6 remain not handled.

The merging of two ERTs was implemented only in the case of two XPath expressions without any alternative paths. In this situation, merging is trivial and consists in concatenating both ERTs' definitions.

5.4 Summary

In this chapter we presented the prototype implementation of our Web navigation and data extraction model introduced in Chapter 4. We started by presenting the extensible architecture of the proof-of-concept implementation (Section 5.1), including detailed discussion of the complex topics of Web actions execution (Section 5.1.3). Then we presented implementations of key interfaces corresponding to used data extraction nodes (Section 5.2) and subroutines used by query planning and graph execution algorithms (Section 5.3).

[6]See: `http://curl.haxx.se/libcurl/dotnet/`. Accessed on 19.05.2011.

[7]See: `http://curl.haxx.se/libcurl/`. Accessed on 19.05.2011.

Chapter 6

Discussion and Evaluation of Proposed Approach

In Chapter 4 we presented our data extraction model, and in Chapter 5 we presented the prototype application that implements the proposed model. In this chapter, we evaluate the applicability in real-life Web sites and generality of the proposed solution, as well as its ability to limit the overhead of data extraction. Our evaluation combines the evaluation techniques of simulation, scenarios analysis and informed argument [152].

Firstly, in Section 6.1 we contrast our approach with a few of the most similar previous systems. Then, in Section 6.2 we present the set of test Web sites developed to perform the actual evaluation of the system's generality and performance. In Section 6.3, we use the results of the analysis of handled and unhandled Web sites to provide a qualitative overview of the generality of our approach with respect to set objectives, Web information extraction challenges, different types of data-intensive Web sites, as well as different technical and business usage scenarios. Next, in Section 6.4, we use a few complex test Web sites to compare the performance of our solution with the most similar previous systems. The evaluation is performed separately for stateless and stateful Web sites. We also verify the impact of some query plan optimization techniques. Finally, in Section 6.5 we present a few ideas for how challenges that were not directly addressed by our solution can be handled, and present our plan of the future research and implementation improvements to our solutions.

6.1 Comparison of Our Approach with Previous Systems

In this section we present the key similarities and differences between our Web information extraction approach and three chosen previous solutions. The three systems that we compare our solution to are: Araneus (described in Section D.10), Denodo Platform (described in Section D.30) and DWDI (described in Section D.38).

Each of these systems has some important traits in common with our approach. While Araneus is a system that covers also the generation of data-intensive Web sites, its core functionalities are focused on data extraction. Similarly, as in our case, the Araneus approach is based on an abstract, graph-based model of a Web site's navigation and data.

Denodo Platform should be considered more as a Web site crawler than an information extraction environment. However, its approach to the technical issues of accessing Web content and navigating in complex Web sites is the most similar to ours out of all reviewed solutions.

Finally, the DWDI approach is one of a few systems that explicitly consider both Web navigation and data extraction, and is capable of acquire records dispersed among multiple pages of Deep Web sources, based on a modeling approach. At the same time, it is the only solution out of all those reviewed in Section D that has some support for stateful Web sites.

We start the comparison of our approach to these three systems by applying the comparison scheme introduced in Section 3.1. Next, we contrast the used methods in more detail.

6.1.1 Analysis Based on Comparison Scheme

The comparison scheme presented in Section 3.1 covers three areas: extraction output characteristics, extraction rules characteristics and approach to extraction rules management.

Output Characterisitcs

Table 6.1 compares our approach with Araneus, Denodo and DWDI with respect to output characteristics. Table values and column names are taken from Table 3.2. For all four systems we analyzed: the usage of named attributes (column N), the usage of typed attributes (column T), support for the extraction of records (column R), lists of values (column L), hierarchically nested data (column H), graphs (column G), content blocks (column B) or complete documents (column D), as well as output format of the extracted data.

System	N	T	R	L	H	G	B	D	Output format
Araneus	●	○	●	○	○	○	○	○	ADM
Denodo	-	-	-	-	-	-	-	●	?
DWDI	●	○	●	○	○	○	○	○	XML
Our solution	●	◐	●	○	○	○	◐	◐	.NET objects, CSV, XML

Table 6.1: Comparison of Our Approach With Araneus, Denodo Platform and DWDI With Respect to Output Characteristics

As can be seen from Table 6.1, our approach has a relatively similar output data model to Araneus and DWDI. It is based on relational records and uses named attributes. While they can be typed, it is not required and by default all values are extracted in string format.

Our approach can be used to extract lists, hierarchies and graphs. However, in output format they are always represented as a single string (e.g. the comma-separated elements of a list, the XML representation of a small graph) or as additional records in a relational model (e.g. a list represented as a Cartesian product of a record and all list values, or a graph represented as two sets of records: objects and edges). Thus, they are not supported as a part of the output format. In the case of block contents or complete documents the situation is similar; however, in this case the representation as a very simple relation is not that cumbersome, so we treat this aspect of output format as partially addressed.

As Denodo Platform is rather a Web crawler than data extraction system, its output consists of downloaded documents. Thus, it is completely different from our solution.

Extraction Rules Characteristics

The second area of the comparison scheme used in Chapter 3 focuses on the characteristics of used data extraction rules. In Table 6.2 we compare the values for three reference systems (copied from Table 3.3) with the characteristics of our solution. The considered aspects are: support for rule-based (column +) or ruleless (column —) extraction; usage of linguistic resources (column l) or previous data (column d); declarative (D), procedural (P) or mixed (M) character of extraction formalism (column f); ability to work with fully (column F) or partially (column P) specified wrappers; ability to perform context-based (column x) and content-based (column t) extraction; as well as reliance on grammars (column G), finite automata (column A), regular expressions (column R), logic-based formalisms (column L) or XPath language (column X).

System	+	—	l	d	f	F	P	x	t	G	A	R	L	X
Araneus	●	○	○	○	M	●	○	●	○	○	○	○	○	○
Denodo	●	○	○	○	M	●	○	●	○	○	○	○	○	○
DWDI	●	○	○	○	D	●	○	●	○	○	○	○	○	●
Our solution	●	○	○	○	D	●	○	●	◐	○	○	●	○	●

Table 6.2: Comparison of Our Approach With Araneus, Denodo Platform and DWDI With Respect to Extraction Rules Characteristics

As shown in Table 6.2, there are similarities between Araneus, Denodo, DWDI and our approach. All four systems are rule-based, use fully-specified wrappers and enable context-based data extraction. Moreover, all of them use at least a partially declarative way of describing navigation and data extraction: with (Araneus and Denodo) or without (DWDI and our approach) procedural language components.

The presented systems use different data extraction formalisms. Araneus uses its own approach. In case of Denodo, information extraction is very limited and consists *de facto* in downloading Web documents. Both DWDI and our solution use XPath expressions. However, we implement also regular expressions (including basic content-based rules). Moreover, thanks to extensible architecture, any other data extraction approaches can be added, including grammar-based or automata-based languages or partially specified rules.

Wrapper Management

The final area of comparison scheme concerned wrapper development and maintenance, as presented in Table 6.3. It covers: used wrapper construction techniques (column W) with possible values being manual (M), GUI-based / interactive (G), supervised learning (L), unsupervised learning (U) and automated extraction (A); systems requirement of expert (column E), user (column U), labeled pages (column L), unlabeled pages (column P) and a domain model (column D) to create wrappers; as well as the existence of long-term adaptation mechanisms (column A), and the effort needed to switch to other domains (column S), with possible values of low (L), medium (M) and high (H).

System	W	E	U	L	P	D	A	S
Araneus	M	●	○	○	○	○	○	H
Denodo	M	●	○	○	○	○	●[1]	H
DWDI	M	●	○	○	○	○	○	H
Our solution	M	●	○	○	○	○	○	H

Table 6.3: Comparison of Our Approach With Araneus, Denodo Platform and DWDI With Respect to Wrapper Creation and Management

As can be seen from Table 6.3, the four compared systems are very similar with respect to wrapper creation and management. Like our approach, Araneus, Denodo and DWDI focus on navigation and data extraction and they do not provide any advanced wrapper creation or maintenance facilities. All of them require manual wrapper creation that requires expert work. Thus, the effort of applying these solutions to a new domain is high. While the team of Denodo authors presented work on maintaining wrappers based on previously extracted instances, it is unclear if and how it was integrated into the whole solution.

6.1.2 Similarities and Differences Between Used Approaches

In the previous section we demonstrated that four compared systems (Araneus, Denodo, DWDI and our solution) are relatively similar with respect to high-level output, extraction rules and wrapper management characteristics. However, while each of the analyzed previous systems is similar to our approach with respect to some aspects of data extraction, there are also significant differences. In this section we discuss these similarities and differences.

Araneus

As Araneus was proposed in the late 1990s, it ha not support for any composite Web documents, GUI-based Web actions or complex server-interaction patterns. Similarly, data extraction methods used by Araneus, based on a mixture of procedural language with very simple regular expressions, is incomparably less adapted to robust data extraction than our approach based on well-recognized regular expressions and XPath languages.

At the same time, Araneus resolved a few problems out of the scope of this thesis, including data integration from multiple sources and hypertext construction from relational data.

The key similarity of Araneus to our approach concerns the area of navigation graph modeling. Both Araneus and our solution treat a navigation-oriented Web site model as the

[1]automated, based on previous instances

essential element of the whole solution, and consider accessing Web resources as a topic at least as important as pure data extraction. Finally, both approaches are data-centric and handle the extraction of records that span the borders of a single page.

There are four key differences between both graph models. Firstly, in Araneus graphs, nodes correspond to classes of pages sharing data schema or a few alternative schemas (in the case of heterogeneous union), and edges between them represent navigation patterns between these pages. Thus, this model represents strictly the navigation possibilities of a Web site. In contrast, in our approach the nodes correspond to possible actions that can be performed in a Web site, and the edges model structured (multi-attribute) data flows between these actions. Thus, our model allows us to model both the data-driven selection of navigation actions that exist in a Web site, and the construction of such actions based on previously extracted data.

The second difference concerns the modeling of extraction rules. In Araneus, data extraction is not a part of the navigation model, while in our approach the same graph models both different types of navigation actions and all data extraction operations. While the Araneus approach offers better separation between different tasks, in our approach optimization of navigation based on data extraction (i.e. the quick cutting of navigation paths) is possible.

Thirdly, our approach to modeling the actual scope of data present in a page, which is based on current query, is significantly more expressive than the modeling data containment between two types of pages that is implemented by Araneus. While our approach can be easily used to model queries containment, there are multiple other current query aspects (e.g. overlap between different queries, data ordering, set of current constraints) that are represented in our model and are not in Araneus.

Finally, at the moment of development of the Araneus model, the common assumption was that Web sites are stateless. Thus, the Araneus does not provide any concepts useful for navigation state modeling, while our approach handles this aspect in a complete way.

Denodo Platform

While Araneus partially inspired our data-driven navigation model, the approach presented by authors of Denodo Platform helped us shape many of te technical aspects of our solution.

Denodo Platform ignores many aspects that are central to our approach to data extraction from complex Web sites. It does not provide direct support for extracting data from individual documents; it has a simplified, partially procedural approach to navigation modeling, without support for data-driven path cutting; and it does not provide any universal querying capabilities nor stateful Web sites navigation facilities.

On the other hand, Denodo supports a few tasks that are not handled by our approach. Firstly, it provides the forms parsing and analysis facilities typical for Deep Web data extraction systems. Secondly, it provides an advanced formalism for the specification of the querying capabilities of HTML forms. Finally, the Denodo platform is capable of performing the parallel and distributed execution of data extraction tasks.

The most similar aspects of the Denodo Platform and our solution are related to handling technical issues such as the representation of multi-frame documents, the dealing with JavaScript, Cookies, HTTPS and certificates. Similarly to our solution, Denodo Platform uses a rich document representation based on an embedded Web browser application. It also uses the Web browser capabilities in order to handle different types of JavaScript code and events, as well as Cookies and accessing HTTPS.

These features were not well addressed by any system before Donodo Platform. Thus, the Denodo approach was a direct inspiration for the technical solutions we implemented in our prototype. However, by combining an embedded Web browser with a Web proxy, we further improved the basic ideas of Denodo's approach to dynamic and composite content.

Two areas where we plan to develop our approach based on Denodo experiences are the description of querying capabilities and the distributed execution of navigation tasks.

DWDI

The third solution that is a reference point for our approach is DWDI, which was a direct inspiration for us in the areas of navigation modeling and execution. Like our approach, DWDI provides a complex, formalized approach to navigation modeling that enables data extraction from Web sites with cyclic navigation graphs and the construction of data records from attributes dispersed in multiple pages. It is also the single solution preceding our system that provides partial support for stateful Web sites.

As in the case of Araneus, the graph model used in DWDI consists of nodes (states) corresponding to generalized Web pages, and of edges (input symbols) that model hyperlinks and the filling in of HTML forms. This approach is significantly less expressive than our graph consisting of Web action and data flows between them modeled as input mappings.

In contrast with Araneus, DWDI proposed a clear graph execution algorithm based on finite state automata. It was a direct inspiration for our approach to navigation plans executions. However, our algorithm is more modular (the graph search strategy can be easily adapted by replacing some subroutines), and better fitted for distributed or parallel execution.

Another feature of DWDI concerns basic query issuing capabilities. However, as compared with our query planning support, the DWDI approach is very limited. It allows only for the substitution of some parts of specific URL patterns or HTML form fields by user-provided values, while our approach enables the rewriting of arbitrary queries using any extractable attributes.

Finally, one of the specificities of the DWDI approach is that for each HTTP request a complete path of preceding requests is recreated. Thanks to this approach, DWDI is capable of dealing with some stateful Web sites. It handles all sites where all modeled stateful Web action trigger the replacement of some navigation state part, as well as all graphs where start input symbols cause a complete reset of the navigation state.

While this approach can work well for many stateful sites, it also causes a relatively high overhead, especially in the case of stateless Web sites. Therefore, our approach that uses explicit modeling of the impact and dependency on specific parts of th navigation state is much more cost-effective. Moreover, if we mark all nodes as both modifying some part of the navigation state and depending on it (e.g. because no detailed information on Web site behavior can be obtained and we prefer to err on the safe side), the original DWDI behavior can be reproduced in our model.

6.2 Evaluation Based on Test Web Sites

The evaluation of traditional information extraction systems was relatively simple. It was enough to randomly or manually select some real-life documents, and annotate them or manually assign reference data to each of them.

In the case of complex data-intensive Web sites, evaluation is much more difficult. Theoretically, there are three options of creating data sets for evaluation purposes. The first possibility is to directly use some randomly chosen real-life Web sites. While tempting, this approach has a very limited evaluation value. Due to the quick evolution of the contents, templates and technologies used by Web sites, in this approach it is impossible to properly annotate test Web sites, and no repetitiveness of experiments can be assured.

The second possibility is to make an exact copy of a random set of complex data-intensive Web sites, and to perform the evaluation on this copy. While this approach assures repetitiveness and is the ideal solution for comparing different data extraction approaches over time, making such a copy is challenging for a few reasons. Firstly, many data-intensive Web sites are large and hard to crawl completely. Secondly, many of them use dynamic client-side interaction paradigms, making them relatively hard to crawl. Finally, the most difficult challenge consists in reproducing a Web site's behavior. While it can be simple in the case of basic data-intensive Web sites, it is much more difficult in case of Web applications using technologies such as AJAX, and almost infeasible in the case of complex stateful Web sites.

As a result, we believe that only the third option – simulation using a set of artificial Web sites with well known behavior – assures complete and repetitive validation of solutions. Moreover, such an approach allows us to cover all important challenges by using a relatively low number of Web sites. Moreover, it simplifies the detection of challenges where a specific solution fails, and assures comparability with future data extraction solutions.

6.2.1 Created Web Sources

In this section we present a set of 40 manually created Web sites in the domain of "new cars", representing different combinations of the challenges we identified in this thesis. We use them to evaluate how our solution deals with the challenges of data extraction from complex data-intensive Web sites. While these Web sites were developed specifically for the evaluation of our solution, they can be used as a test data set for any future data extraction system.

The created Web sites are characterized by different data organization into pages, level of statfulness, used technologies, and the type of data duplicates they contain. At the same time, they have very simple templates. The actual performance of low-level data extraction in our approach depends mostly on the data extraction languages used, that have been widely studied before, and that can be easily replaced by other languages when needed. Thus, testing the performance and expressiveness of low-level data extraction is not our primary objective.

It is to be noted is that in the case of all Web sites that have some "random" or "unpredictable" behavior (e.g. additional navigation elements occurring from time to time), to assure comparability and repetitiveness, we make these irregularities occur always at the same pages and location.

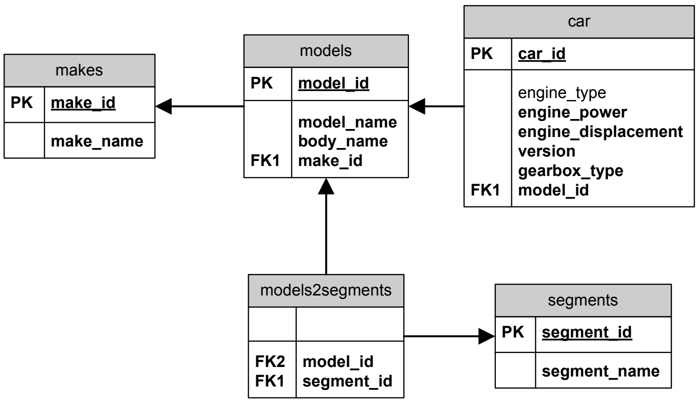


Figure 6.1: Data Model Used by Test Web Sites

All created Web sites use a shared automotive database containing new cars. The entities used by all Web sites are: car makes (such as Fiat or Renault), car models (such as Panda, 500, Scenic, Megane) and the specification of car version, types of gearboxes and engines parameters. Additionally, a few Web sites use cars categorization into market segments (one model can belong to more than one segment). The shared data model is depicted in Figure 6.1.

The created Web sites with short descriptions are gathered in Appendix F, Table F.1. They cover a wide range of challenges related to our research objectives. Some of them serve all data as a single large document (TS0), provide navigation based on hyperlinks (TS1, TS3, TS21), JavaScript (TS2, TS8), HTML forms (TS4, TS10, TS11) and AJAX (TS15). Specific challenges present in these Web sites include: various types of data duplication (TS5, TS6, TS7), HTTP-level issues (TS9, TS13), client-side and server-side statefulness (TS14, TS15, TS16), navigation complexities (TS20, TS21, TS22, TS26, TS27), the limits of a number of requests (TS38, TS39) and issues related to user interaction (TS31, TS32, TS35, TS36). A few Web sites focus on data presentation challenges including template irregularities (TS18, TS19), the use of pivot tables (TS23, TS24), the use of frames (TS29, TS30), different types of obfuscation of encoded or user content (TS17, TS33, TS34), hierarchies of unknown depth (TS25) and the dispersion of data about a single record in multiple locations in a page (TS37).

6.2.2 Wrapping Test Web Sites with Our Solution

For all Web sites described in the previous section, we attempted to develop wrappers using our solution that extract complete, non-redundant data. The results are gathered in Table F.1. For each Web site we specify if it has been wrapped completely (i.e. extraction of all needed and only needed data is possible), partially (i.e. data extraction is possible but there are some performance or duplicate data issues) and not at all (i.e. there are some aspects of the Web sites that are not handled by our system and make data extraction impossible).

We use a three-level scale (completely wrapped, partially wrapped and not wrapped at all) instead of precision and recall for two reasons. Firstly, precision and recall values depend on the data contained in a Web site and on how frequently specific challenges occur in a Web site. As our data set does not aim at representativeness, we would not be able to generalize the figures of precision and recall to all Web sites. Secondly, the key objective of this study is to understand the generality of our solution, rather than to assess its effectiveness in a specific data set. The same, relatively low precision value may result from a single unaddressed challenge, present in many pages of a Web site, or from three unhandled challenges that happen less often. In such cases, we judge it more important to understand the reasons for failure, than to provide quantitative measurement of their scale.

As shown in Table F.1, out of 40 test Web sites, 30 are completely wrapped, six are partially wrapped and four are not wrapped at all. The first group of partially wrapped Web sites covers three Web sites that contain duplicates (TS5, TS6 and TS7). For all of them our current solution returns repeated records, and requires a post-filtering of extracted data. In the case of lattice-based navigation, the duplicates could be handled by analyzing overlapping current queries; two other cases result directly from data duplication in the Web sites, and can be handled only by incorporating data filtering into data extraction.

Another group of partially wrapped Web sites contains both sites with pivot tables (TS23, TS24). While in this case the extraction is technically possible, the extraction rules used are very complex and not intuitive. Moreover, if our Web sites used complex combinations of `colspan` and `rowspan` the task of information extraction would become almost infeasible.

The final partially wrapped Web site is TS38 (having a global limit of requests that can be issued to it within any 12 consecutive hours). While our wrapper was capable of extracting data from it, the result was incomplete. This problem cannot be completely avoided. However, we plan to use two techniques to decrease its severity. Firstly, proper path management policy can be used to focus on extracting information of most value to the user. Secondly, by performing distributed data extraction from different IP addresses, it is possible to increase several times the actual limit of a number of requests.

Out of four Web sites that could not be handled at all, two (TS35 and TS36) use timer-based triggering of client-side events, one (TS22) uses CAPTCHA tests, and one (TS34) uses an obfuscation techniques that require visual data extraction. All three topics: handling timer-triggered Web actions, enabling user interaction for handling CAPTCHAs and enabling visual data extraction belong to our plans of future work, described in Section 6.5.

6.3 Generality of Proposed Solution

Our main objective, defined in Section 4.1, was to develop a solution general enough to work with different types of complex data-intensive Web sites, in several business and technical scenarios. In this section we demonstrate the generality of proposed solution. In Section 6.3.2 we discuss how the proposed solution addresses challenges defined in Section 4.1. Next, we analyze the applicability of our proposed solution for different types of data-intensive Web sites (Section 6.3.3), different business scenarios (Section 6.3.4) and technical tasks (Section 6.3.5).

6.3.1 Research Objectives

In Section 4.1.1 we list a number of challenges that we took into consideration while designing our solution. In this section we discuss how these challenges are addressed by our solution.

In Table 6.4 for each challenge we specify if it can be addressed by extensions to the proposed model (1), if it is addressed by the core model and its algorithms presented in Chapter 4 (2), and if the challenge is addressed by the prototype application described in Section 5 (3).

Challenge	1 / 2 / 3
2. Combining multiple extraction languages using different features	●/ ●/ ●
3. Combining content-based and context-based extraction rules	●/ ●/ ●
4. Connecting data from multiple pages into records	●/ ●/ ●
5. Dealing with AJAX-like requests	●/ ●/ ●
8. Dealing with non-deterministic navigation patterns	●/ ●/ ●
9. Dealing with technical issues of HTTP(S) (Cookies, redirects, encryption, compression, error codes, certificates)	●/ ●/ ●
11. Dealing with Web site's statefulness	●/ ●/ ●
15. Extracting hierarchies of unknown depth	●/ ●/ ●
22. Using both encoded and user content for data extraction	●/ ●/ ●
24. Working with composite (multi-frame, using JavaScript and CSS) Web content	●/ ●/ ●
25. Working with interactive content (Web actions based on interaction with user interface)	●/ ●/ ●
14. Extracting data from complex data presentation structures	●/ ●/ ◐
17. Limiting number and frequency of requests	●/ ●/ ◐
7. Dealing with different file formats	●/ ●/ ○
12. Enabling multiple alternative rules for augmented robustness and better maintenance	●/ ●/ ○
1. Adopting a flexible approach to choosing the most valuable data during extraction (utility maximization)	●/ ◐/ ◐
18. Supporting data extraction of graph data	●/ ◐/ ◐
21. Using additional server query capabilities	●/ ◐/ ◐
6. Dealing with data duplicates	●/ ◐/ ○
10. Dealing with timer-triggered Web actions	●/ ◐/ ○
20. Taking advantage of typical constructs of Web sites: pagination, in-built ordering and dynamic filters	●/ ◐/ ○
23. Using incompletely specified rules	●/ ◐/ ○
13. Enabling user interaction during data extraction (e.g. in order to handle CAPTCHAs)	●/ ○/ ○
16. Implementing advanced, state-aware query planning with multiple optimization steps for both navigation and extraction	●/ ○/ ○
19. Supporting distributed data extraction	●/ ○/ ○

Table 6.4: Challenges Addressed by Proposed Solution: (1) by model or its extensions, (2) by core model and algorithms, (3) by our proof-of-concept implementation

As it can be seen from the table, our model directly addresses the majority of challenges and can address all the remaining ones by using some extensions. At the same time, our prototype implementation focused only on a subset of model features. In the remaining part of this section we discuss each of the challenges separately.

Flexible Approach to Choosing the Most Valuable Data (Utility Maximization)

In very large Web sites it is not always possible to download all data corresponding to a user query. Even if no technical limitations of numbers of requests exist, the download time of all data may be significantly too long. Thus, in such cases, an important property of the data extraction algorithm should be the ability to prioritize data of the highest utility to be extracted first (based both on the current query and already extracted attributes) and to give access to already extracted records while data extraction continues.

In our model data prioritization can be attained thanks to the extensible character of the queue management algorithms. Ideally, utility-maximization queue management policy should combine some navigation heuristics, information available in the current query model (including data ordering and limit parameters of current query) and the query-specific utility measurement function, based on already extracted attributes. For example, the algorithm could use the depth-first approach until some data allowing utility estimation is extracted, and then continue to use the depth-first approach for the highest utility items and postpone the execution of other items. It could also rely on the current query model in order to use the descending ordering of data by creation date or item popularity, and make the utility function inversely proportional to the record number in the result set. While our model gives multiple possibilities of implementing utility-maximization functions, in this thesis we provide no specific algorithms. It is also to be noted that the use of algorithms based solely on utility maximization would work well in stateless Web sites. In the case of stateful Web sites the algorithms would be much more complex, as they would need to combine utility maximization and states recreations minimization.

While we provide no specific implementation of utility-driven queue management, for partial data reception we provide not only the static model, but also the necessary algorithms and their implementation in our proof-of-concept prototype. We support two aspects of partial data reception. Firstly, data generated by any data node is returned whenever it is generated – even if it covers just a part of the records to be returned (i.e. the node is going to be instantiated multiple times). Secondly, use of data nodes (attribute sets) in the "middle of the graph" (i.e. not as end nodes), makes it possible to obtain incomplete data records.

In current proof-of-concept implementation, data nodes are manually specified in the Web site's data extraction graph. However, it would be possible to automatically add such nodes during query planning, depending on specific user query (as with data filtering attribute sets added by one of proposed query plan optimization algorithms).

Combining Multiple Extraction Languages

The possibility of combining different data extraction formalisms is one of the core features of the proposed model. The proposed solution is very flexible with respect to the data extraction languages it can use. It is capable of using both single-slot and multislot extraction languages, executable on text or DOM content. Moreover, optional rules for incorporating data filtering into the extraction rule template node can be separately defined for each data extraction language. Such a flexible approach makes it possible to reuse many of the previous works focused on data extraction languages and formalism using different features.

While our current implementation includes only XPath and regular expressions data extraction languages, new ones can be easily added. Indeed, we plan to integrate into our solution a framework supporting information extraction from text (such as GATE [85] or SPRouT [95], both widely used at Poznań University of Economics) to support grammar-based or vocabulary-driven information extraction, as well as a simple formalisms supporting visual data extraction (based on geometrical and CSS features).

Combining Content-Based and Context-Based Extraction Rules

Given the aforementioned ease of incorporating new data extraction languages, it is clear that our model supports both content-based and context-based rules. Out of two implemented data extraction languages, XPath is much more adapted for context-based data extraction rules, and regular expressions fit better into the ideas of content-based extraction. However, both of them are flexible enough to support also some rules of the other type, and data extraction languages specifically adapted to context or content-based rules may be added in future.

Connecting Data from Multiple Pages into Records

The ability to connect attributes extracted from multiple pages into complete records is one of the key characteristics of the proposed model. In our approach, input mappings used for modeling data flow can span page boundaries, and the provenance-aware join operator assures that only consistent data are joined together to form records.

It is to be noted that while the proposed solution naturally handles situations when attributes are dispersed between multiple pages (e.g. list and details pages) and connected via hyperlinks or HTML forms, it can be also used to connect data from different parts of a Web site or even different Web sites. For example, we may use the extracted product code as a query to another database and then connect data obtained from both sources. However, such connections are limited to the situations when the vocabulary or identifiers used in different Web sites are consistent. In other cases, it would be a much better solution to extract data from both sources independently, and to apply existing record linkage methods afterwards.

Dealing with AJAX-Like Requests

We presented methods that support AJAX-like requests both at the model and the implementation level. In our model, AJAX-like requests are considered jointly with the client-side Web actions that trigger them. The impact of individual AJAX-like requests on specific parts of a Web document can be modeled by using the navigation state management features of our data extraction model.

In our prototype implementation, AJAX-like requests are triggered by using GUI-based Web actions, such as mouse or keyboard actions performed on specific Web page elements. Our implementation assures access to changes induced by AJAX-like Web actions thanks to a rich representation of Web actions output.

Dealing with Data Duplicates

Duplicate detection is an important aspect of the majority of tasks based on information extraction. As many data analysis techniques rely heavily on aggregate functions, leaving duplicates in extracted data may significantly change the results of analysis.

During interactions with data-intensive Web sites, duplicates occur for two distinct reasons. Firstly, multiple queries issued to the Web site may partially overlap. Secondly, the source itself may contain some duplicate records (with respect to extracted attributes).

Currently our method provides a complete solution to neither of these two aspects. However, the current query model presented in Section 4.5.7 surely is a base for handling overlapping queries. By modeling the current query of a given navigation state, it is possible to analyze the relation of multiple queries (e.g. subsumption) and to define additional data filtering rules. While the current query model is a part of the model described in this thesis, query planning and execution algorithms that rely on it are still to be developed in our future work.

The problem of duplicates present in the original source cannot be completely resolved in an efficient way. The simplest solution, i.e. duplicate removal after the extraction ends or after each new group of records is received, is simple and universal and allows to a multitude for existing deduplication and data cleansing techniques. On the other hand, it does not give any support for cutting navigation paths that contain duplicates.

Another imaginable solution that can be easily integrated into our method is to detect and remove duplicates in data management routines. However, it is applicable only if there are functional dependencies between multiple attributes, and the repeating value for some attributes means that a complete record would be a duplicate (e.g. repeating PESEL or social identity numbers normally lead up to repeating first and last names).

Dealing with Different File Formats

Our solution has no direct support for dealing with file types other than HTML. However, thanks to the navigation and file download capabilities of our model and implementation, adding support for other file types is a relatively simple task. It requires only implementing a document parser, a specific document model and specific format extraction rules. Moreover, for some types of documents (e.g. XML, JSON and, to some extent PDF) it would be possible to reuse some part of the existing document model and extraction languages.

Dealing with Non-Deterministic Navigation Patterns

Navigation in a Web site is non-deterministic, if for at least Web action its results differ over time in a random way. For example, in many Web sites clicking a link that points to a news article occasionally loads a page containing an advertisement. Then, clicking "next" or "skip the ad" button may be required to access the desired content.

Our solution provides no explicit semantics to handle non-deterministic navigation patterns (as it is in the case of the current query model). However, such situations can be handled by using the core components of our model.

The solution is to use a special category of content-based extraction rules, called recognizers, that return the complete document, provided that it contains some characteristic element. Non-deterministic navigation issues can be resolved by mapping the output of a Web action that loads a page into two separate recognizers corresponding to both normal and "exceptional" situations (e.g. when no ad is present and when the ad is loaded). Then, an additional Web action executed on an "exceptional" document can be added, looping back to the original Web action. As a result, this extra loop will be repeated until the normal content is loaded.

Dealing with Technical Issues of HTTP(S)

This challenge covers dealing with Cookies, HTTP redirects, data encryption and compression, HTTP error codes handling and dealing with certificates. While this challenge is mostly interesting from an implementation perspective, it is one of the most important ones in complex data-intensive Web sites. Our prototype addresses almost all listed challenges by combining an embedded Web browser control with a proxy server based on the FiddlerCore library.

The only issue that remains implemented only partially, is related to handling certificates. Using Fiddler to capture and modify content makes all certificates invalid (as they are issued for other hosts than localhost), whether they were originally valid or not. Our current solution ignores all certificate failures at the Web browser level. However, it decreases the security of the proposed solution. Thus, in our future development work we plan to overwrite the original certificate at proxy level by a valid or invalid certificate for the localhost server, depending on the validity or invalidity of the original certificate for the remote host.

Dealing with Timer-Triggered Web Actions

While complex data-intensive Web sites that rely heavily on client-side technologies are mostly event-based, with a traceable relation between cause and consequence of an event, in some situations they use deferred JavaScript actions based on timer-triggered events. This brings in two important issues. Firstly, it makes the causal connection between action and result hard to detect. Secondly, in some Web sites it makes users wait a long time (e.g. several dozens of seconds) before the effect of the Web action becomes apparent.

A possible solution to timer-triggered events in our current model is to introduce a special type of node combining Web action and a recognizer extraction rule, which would execute the Web action and wait until the recognizer extraction rule is capable of producing some output. While this solution is relatively simple, it can resolve almost all situations, including issues related to loading data through persistent background connections to the HTTP server (e.g. the Comet or AJAX push libraries or HTML5 server-sent events).

However, in the sequential, linearized approach to graph execution presented in this thesis, such nodes may cause significant data extraction delays. Thus, ideally this solution should be combined with distributed graph execution (discussed further in this section), so that even if a single thread pauses until the deferred content is loaded, the other threads continue to be executed.

Dealing with Web Site's Statefulness / State-Aware Query Planning

A model capable of representing a Web site's statefulness, and algorithms that use this model for query planning and data extraction, are the key novelties of our data extraction model. Our approach to stateful Web sites is based on data extraction graph annotation, which is discussed in detail in Section 4.5. It provides a way to describe how specific Web actions modify distinct parts of the navigation state and how they depend on it. The proposed model distinguishes between different types of navigation state modifications, and introduces a granular, hierarchical approach to representing navigation state parts.

This description is rich enough to model the need of recreating of (some parts of) the navigation state, as discussed in Section 4.5.2. Combined with some optimization techniques, it minimizes the recreation needs and improves the performance of graph execution (as described in Section 4.5.5). Therefore, our approach is capable not only of extracting data from stateful Web sites, but also of using navigation state annotation to minimize overhead in such Web sites.

Alternative Rules for Better Robustness and Maintenance

The concept of using alternative extraction and navigation patterns is based on the simple observation that in many Web sites there are multiple ways of accessing the same data, both at the navigation and data extraction level, and often it is hard to decide which of these ways will be the most stable over time. At the navigation level there may exist multiple navigation paths corresponding to the same query (e.g. one based on HTML form and another one based on links). Similarly, at the data extraction level different extraction rules (e.g. using different languages and features) may correspond to exactly the same data.

In traditional information extraction systems a single representation would be chosen out of each set of alternative navigation or data extraction patterns. The idea of alternative rules consists in memorizing multiple patterns corresponding to the same data and using them all occasionally and comparing the results. Then, any differences between results should lead to some adaptation of the wrapper to the Web site's changes or to discarding alternative paths that are not stable enough over time. While applying this approach requires the occasional use of more bandwidth and processing power, it would allow for an early detection of Web site changes, and of continuous adaptation of data extraction rules and Web site navigation description.

The proposed model has limited support for alternative navigation or data extraction rules. Thanks to the current query model, it supports the explicit description of alternative data extraction graph paths, i.e. paths that correspond to the same user query. Therefore, data extraction graphs can be used to model alternative extraction and navigation rules. However, the algorithms checking rules consistency and performing the adaptation of data extraction graphs are to be developed as part of our future work.

Enabling User Interaction During Data Extraction

The incorporation of user interaction into the data extraction model concerns the rare situation of tasks that cannot be completely automated, such as handling different types of CAPTCHA tests. Such user interaction components can be easily added to our prototype as a new type of node that accepts the values to be presented to the user (e.g. a fragment of HTML code containing a CAPTCHA image) as input, and returns as output the value(s) entered by the user into a text box. This type of node is completely compatible with our data extraction model, and requires only implementation work.

Extracting Data From Complex Data Presentation Structures

The ability to extract data from complex data presentation structures, covering different types of hierarchical data presentation formats, pivot tables and complex graphs or charts, is mostly dependent on the expressive power of the data extraction languages udes.

The set of data extraction languages implemented in our proof-of-concept application (XPath and regular expressions) is capable of handling most hierarchical data presentation formats. Even the majority of HTML pivot tables can be processed by XPath expression; however, they tend to be very complex and rely on non-intuitive constructs such as "choosing the `<td>` element with the same index within its parent `<tr>` as the `<th>` element with content corresponding to some specific label within its parent `<tr>` node". Therefore, data extraction from complex presentation structures could benefit a lot from implementing specific data extraction languages based on tables parsing or on visual and geometrical features. As we discussed in the section related to using different data extraction rules, once such a language is implemented, adding support for it to the model is very simple.

At the same time, the proposed solution provides a relatively good support for data extraction for these complex graph or charts that are based on Java or Flash control downloading data from the server in simple structured data formats (such as XML or JSON). In such cases, which are typical for complex graphs and charts representation, our prototype's ability to use not only user content but also encoded content is crucial.

Extracting Hierarchies of Unknown Depth / Supporting Data Extraction of Graph Data

Dealing with hierarchies of unknown depth is one of the challenges that to the best of our knowledge were not handled by any previous solution. Our approach is fully capable of dealing with such data structures by combining cyclic data extraction graphs with the use of constructor output slots, as we demonstrated in Example 4.3.10.

In the case of graph data (such as social networks), no separate solution is included in our model. However, our approach is flexible enough to process graph data by separately extracting data objects (as in non-graph Web sites) and the relations between them.

Both in the case of extracting hierarchies of unknown depth and graph data, the output of data extraction is limited to the relational model. For example, hierarchical data are not represented by a special type of data structure, but are encoded as string values. Thus, depending on the application of the extracted data, different types of post-processing may be needed.

Limiting Number and Frequency of Requests

As issuing requests to HTTP servers is the most time consuming part of all data extraction algorithms, limiting the number and frequency of requests is one of the key challenges addressed by our solution. We include several methods of limiting the number and frequency of requests.

Firstly, our approach is focused on cutting out useless navigation paths as quick as possible. It is possible thanks to the introduction of data filtering attribute sets and the merging of data filtering directly into data extraction nodes.

Secondly, thanks to state-aware queue management the number of required navigation state recreations can be significantly limited. In some cases it may allow us to avoid several dozens of requests per each state recreation.

Thirdly, depending on the Web site, it may be possible to block the downloading of all or the majority of images, styles and JavaScript files, by specifying request filtering rules (as in many Web site materialization tools, such as WinHTTrack). While we have not implemented such filtering yet, it can be easily done within Fiddler proxy request and response events.

Finally, the frequency of requests is limited by applying the politeness policy, i.e. by introducing some minimal time between two consecutive requests to the server and the limitation of a total number of parallel requests.

Supporting Distributed Data Extraction

The graph execution algorithm that we presented in Chapter 4 is a sequential, linear algorithm. However, the proposed model and execution algorithm are *de facto* based on an event-based approach. As, for example, in the case of Petri networks, every instantiation of a data extraction node is triggered by inputs arriving to the node. Therefore, the design principles of our model support parallelization, and the only requirement is to develop a parallel version of the proposed graph execution algorithm.

Performing data extraction graph execution in a parallel or a distributed way has multiple advantages. The parallel version of the algorithm would be much less fragile to the delays of individual Web actions. Thus, even if one thread hung, that algorithm would continue to be executed. Even more advantages would result from distributing algorithm execution to multiple agents working in different network locations. As requests would come from multiple IP addresses, it would make data extraction harder to detect. At the same time, per IP limits of a number of returned values or total transfer would be much easier to overcome. Moreover, as the limits of a number of requests per a unit of time (politeness policy) in our implementation are attached to a single IP address, data extraction could be performed much quicker.

The trivial approach to parallel or distributed execution would be to use multiple agents using a single shared queue. Such a solution can be implemented in a very simple way, with the only challenges being purely technical and concerning communication between multiple agents.

However, for good state management and user mimicking two additional considerations are necessary. Firstly, each agent should model a client and server-side navigation state of its own, enabling the state-aware distributed execution of the data-extraction graph. Secondly, each thread of agents should consist of Web actions that preferably follow some existing navigation path in the Web site. As discussed above, the challenges and key aspects of the possible solution are clearly defined, so we plan to make distributed graph execution a part of our future research.

Using Pagination, Data Ordering and Dynamic Filters

Pagination, in-built data filtering and ordering facilities are important concepts present in the majority of data-intensive Web sites. Their modeling provides not only a better understanding of a Web site's behavior, but also more tangible results while dealing with duplicates and for enabling alternative extraction rules (as discussed above).

We addressed part of this challenge by extending our original model with meta-data allowing us to model the current query, and its evolution caused by executing consecutive Web actions. However, we leave the query planning and graph execution algorithms capable of using current query information to our future work.

Using Additional Server Query Capabilities

In many Web sites, the server site Web application logic is capable of handling more complex queries than those allowed by user interface. For example, it may allow us to constrain multiple attributes at once (even if GUI allows us to query one attribute at time), to modify the number of results in each result page or to provide more alternative constraint values for an attribute. For some queries, using such additional query capabilities may help avoid the unnecessary download of irrelevant records that can be filtered out only after the extraction is performed.

As our model is capable of working not only with GUI-based but also with request-based Web actions, building data extraction graphs that use additional query capabilities is possible. Moreover, it is also possible to modify requests captured by request capture nodes, thus allowing us to use additional query capabilities even in the case of some GUI-based Web actions.

The most important missing component, which concerns both query capabilities exposed by the user interface and additional ones, is a formal way of specifying the allowed combinations of constraints that can be used to query the Web site. In future we plan to reuse existing solutions from the database community and data extraction field (e.g. the Denodo solution).

Using Both Encoded and User Content / Working with Composite Web Content

The possibility of extracting information from both the encoded content and user content is one of the key innovations of our data extraction model. It is assured by combining the rich representation of Web documents (based on the Web browser's DOM model and textual document representation) and the proxy-level capture of encoded content.

It is to be noted that the document representation we use also supports composite content. If many frames exist in a single Web document, they are represented by a single DOM tree with alternative representations as HTML source code (with frame documents embedded within an `<(i)frame>` tag) and plain text of all documents.

Moreover, as we implemented also a mechanism for detecting changes induced by a Web action, our model enables us to limit data extraction only to the new content, received after a Web action had been performed. Thus, our approach deals well with the partial replacement and cumulative modifications of a current Web document navigation part.

Using Incompletely Specified Rules

Incompletely specified rules that combine some specific information on items to be extracted with heuristic or probabilistic search algorithms, can be very useful in the case of Web sites with significant template-level irregularities.

While we do not implement any extraction method based on incompletely specified rules, they can be easily incorporated into the data extraction graph as a new type of node. The only limitation is that the schema of data to be extracted needs to be known in advance.

However, if needed it is possible to overcome this limitation in two ways. Firstly, it is possible to extract records consisting of pairs of keys and values rather than individual attributes. While this approach is very flexible and can work with very irregular content, the result of extraction may require significant post-processing to connect individual values to complete records. The second possibility is similar to the one used in the case of hierarchies of unknown depth or graph data. It consists in representing some part of the attributes (e.g. all attributes that are not known in advance) as string values in some flexible language (e.g. XML or RDF), while attributes that are known in advance may remain represented by the individual attributes of attribute sets.

Working with Interactive Web Content

One important characteristic of our solution is a very flexible definition of Web actions. While it covers HTTP requests, link following and form submission, which are traditionally understood as elements of navigation path or navigation session, it also includes a number of GUI-based actions, as discussed in Section 5.2.1. Moreover, thanks to our state management approach, our solution is capable of representing the scope of changes caused by GUI-based Web actions on both the client-side and server-side navigation state.

As a result, our model and prototype implementation are fully capable of dealing with interactive Web content. The only important capability that is missing in our current model is related to dealing with Flash and Java controls. Web actions that would deal with such components are covered by our theoretical model and algorithms described in Chapter 4. Therefore, the challenges that remain to be addressed are purely technical, and we plan to resolve them by using low-level mouse and keyboard events simulation and a special type of recognizer based on visual analysis, as it is done in the iMacros Web automation system.

6.3.2 Addressed Challenges

In the previous section we analyzed how our solution attains the objectives defined in Chapter 4. In this section we compare our solution to top-performing previous systems with respect to the implementation level of key groups of challenges listed in Table C.1.

To compare our approach with previous ones, we applied the same challenges-based analysis as for all previous solutions (see Section 3.2.1). Similarly, as in the case of previous systems, we assigned for each of the challenges one of three values: zero points for a challenge that is not addressed, half a point for partially addressed challenges, and a point for a completely or almost completely addressed challenge.

In Table 6.5 we compare the score received by our solution with selected previous solutions listed in Table 3.1. For comparison we have selected the solutions that received one of the highest scores in at least one category of our classification of Web information extraction challenges (see Table C.1). The categories that we analyzed are: different data models (DM), server interaction (SI), server-side business logic (SS), client-side business logic (CS), different content types (CT), data extraction (DE), navigation planning (NP), navigation plan execution (NE) and usage scenarios specificities (US). We also provide the total percentage of addressed challenges (Tot).

System	Tot	DM	SI	SS	CS	CT	DE	NP	NE	US
IM	4%			17%						**21%**
WebOQL	11%	**39%**	6%		9%	6%	11%	10%	12%	
Stalker	8%	**23%**				6%	**23%**			
SoftMealy	8%	17%				10%	**24%**			
DEByE	12%	21%	5%		17%	6%	**25%**	7%		
EXALG	7%	**39%**					13%			
Denodo	15%		29%	12%	**44%**	6%	9%	13%	5%	12%
DEPTA	6%	6%					**23%**			
ViNTs	7%	19%					**23%**			
Thresher	6%	11%				**17%**	14%			
DWDI	9%		5%		**20%**		5%	**20%**	**19%**	
DEQUE	12%	6%	5%	13%	**24%**		11%	14%	7%	19%
Avg of prev. solutions	–	12%	4%	1%	7%	6%	12%	4%	3%	4%
Our Approach	37%	27%	51%	8%	71%	69%	24%	47%	31%	12%

Table 6.5: Addressed Challenges – Comparison with Previous Solutions

As can be seen from the table, the proposed solution is significantly more universal than previous systems, with the total score (37%) being much higher than the Denodo Platform, the previous leader (with 15% of addressed challenges). The individual challenges vary significantly with respect to their importance and difficulty, and the "partial addressing" of a challenge means different things in different situations. Thus, as we discussed in Section 3.2.1, the provided scores give only an approximation of the implementation level of significant challenges. However, the more than twofold difference between Denodo and our solution shows that our solution is significantly more general with respect to challenges identified in this thesis.

This result is also supported by results for individual categories of challenges. In five out of nine categories, our solution has scores significantly higher than the best-performing previous solution; in one case the score is similar to so far the highest result, and only in three categories is the score significantly lower than previous best results. The results are even more significant, when we remind that out the of previous solutions only three (Information Manifold, Denodo and DWDI) were among the best-performing solutions in more than one category of challenges.

Data Model Challenges

The data model that we adapt is based on the relational model and is relatively simple. While a set of relations can be used to model very complex data structures, our solution does not provide any direct support for many complex data structures, including embedded records. Thus, a significant part of data model challenges is addressed by our solution only partially, and a few solutions have a significantly higher (EXALG, WebOQL) and sightly higher (W3QS, RoadRunner, DeLa) implementation level of challenges in this area.

Server Interaction

The area of server interaction is one of the topics where the advantages of our approach are most apparent. Out of three main subareas of this challenge (dealing with single requests, dealing with multiple requests, detectability) two have almost complete support. As a result, the implementation level of challenges in this area is almost twice as high as the previously best solution (Denodo Platform).

In the case of the challenges of dealing with single HTTP requests, thanks to combining Web browser and Web proxy the score is almost maximal, with a partial support for only two challenges: certificates (only support for certificate validity checking is missing) and different methods of authentication (some types of authentication need to be implemented in future). We also address all detectability challenges apart from support for issuing requests from different IP addresses.

In the area of dealing with multiple requests, only dealing with synchronous and asynchronous requests and challenges of politeness policy are almost completely addressed. However, dealing with Web site limits and distributed execution are to be added in our future work.

Server-Side Web Application Logic

By contrast, the implementation level of challenges of server-side Web application logic (8%) is the lowest result of our solution in all areas of challenges. It is due to the fact that this group of challenges focuses mostly on learning and modeling querying capabilities – the areas that we plan to implement in our future work.

At the same time it is to be remembered that this is the area that is rarely well handled by previous solutions, with a 17% implementation level being the top score (attained by Information Manifold), and 1% being the average. Thus, our score, while very low, is still significantly higher than average and there are only three previous systems (Information Manifold, DEQUE and Denodo) that have better support for challenges in this area. It underlines the fact that this area (especially the empirical learning of complex descriptions of querying capabilities) remains one of the open research fields of Web information extraction.

Client-Side Web Application Logic

The challenge of dealing with client-side Web application logic is the area where our solution attained its highest implementation level (71%), but also the area where the previous highest score (44% of Denodo Platform) was the maximal score in Table 3.1.

Out of four subareas of client-side Web application logic challenges, two are handled by our solution almost completely, one is handled well and the smallest sub-area has no support at all. As our solution is based on an embedded Web browser, it has complete support of challenges related to emulating Web browser behavior. Therefore, it implements both GUI Web actions and manually constructed Web actions and has complete support for different request construction techniques.

The area that is handled partially concerns user interface interactions. On one hand, using a Web browser and implementing GUI Web actions gives almost complete support for interaction with different types of HTML-based controls (with support for CAPTCHAs and controls with "suggestion" functionality being "partial", as they are included in the model but not in our prototype). It also makes possible the emulation of different triggers of JavaScript events (with drag-and-drop included in the model but not in the implementation, but with missing support for timer-triggered events). At the same time, currently the support for Flash, Java or embedded controls is very limited. While no Web actions based on the current state of embedded control are supported, it is possible to execute simple click Web actions and to capture the data downloaded by the control at the proxy level.

Extracting Data from Different Content Types

The score of our solution in this area is almost as high as in the previous area (69%) and is over four times higher than the so far most advanced solution (Thresher). Our advantage, in comparison with previous solutions, is related mostly to the ability to handle complex and dynamic Web documents (i.e. documents using frames and JavaScript or AJAX updates to the content), as well as our support for "content capture", i.e. working with different representations of Web document (including the capture of encoded content and different representations of user content).

While our result in this area is impressive when compared with previous solutions, it is to be noted that the score of some of the previous solutions that rely on Web browser components (including Denodo Platform) may be slightly underestimated. As some of the concerned challenges are highly technical and can be addressed in a relatively simple way by relying on a Web browser's mechanisms, it is probable that they were implemented but not described in research papers. However, even if it was the case, our implementation level would still significantly exceed previous maxima.

Data Extraction

Data extraction is the area where multiple previous solutions have attained similar implementation levels in the range between 20% and 25%, and where most solutions have an implementation level of at least 5%. It is also the area with many challenges related to learning algorithms, that are out of scope of our solution.

Consequently, in this area we did not attain any break-through results. Our implementation level of 24% is comparable with top performing existing systems (Stalker, SoftMealy, DEPTA, ViNTs, and DEByE – the only system with a higher result). However, the focus of our solution is significantly different. While all of these systems have some extraction rules learning or dynamic data discovery components, the score of our approach results mostly from the flexibility of the extraction of composite data, the moderate support of different data presentation models, and some support for using different features during data extraction and some extraction robustness challenges.

Navigation Planning

Challenges of navigation planning are among the most important ones present in complex data-intensive Web sites. However, it was rarely addressed by previous solutions, with the average implementation level of 4% and the best result being 20% (attained by DWDI).

While our approach obtained s score more than two times as high as DWDI (47%), it is relatively low as compared with other key areas of challenges. This is due to the fact that this area contains multiple challenges related to wrappers learning, that are out of scope of the approach presented in this thesis.

Out of three subareas of navigation planning challenges, one is addressed completely, one has almost complete support for non-learning challenges, and in one case only part of the challenges are handled. The proposed solution addresses all challenges related to query rewriting. It is to be noted that, while it means only that the solution is complete with respect to different types of query rewriting tasks, the model still can be extended to include more complex types of queries.

Our approach also has almost complete support for the challenges of dealing with Web site modeling other than those related to wrapper learning. The only challenge that is not handled at all concerns system-initiated Web actions. At the same time, our solution provides only partial support for the current query model (which is modeled but not implemented), and consequently it does not support yet the modeling of complementary data in other parts of Web sites.

In the case of the third group of challenges, concerning choosing the right navigation scope, four challenges (related to including all needed and only needed pages) are addressed completely, ten challenges (covering topics such as avoiding downloads of unnecessary files, supporting personalization and adaptiveness and working with Web sites too large to be processed) have just a partial support, and only five (working with a Web site's limits, using a Web site's querying capabilities, using statistics and in-built data ordering facilities for better query planning) are not handled at all.

Navigation Execution

The second navigation-related area covers the execution of the created navigation plan. As in the case of navigation planning, our implementation significantly outperforms DWDI, which in turn had much a higher result than any other system. However, in this area our implementation level (31%) is lower than in the case of navigation planning.

Our current solution implements only part of the challenges in this area. While it provides a way of interpreting the current navigation state, it can be done only based on the history and use of recognizers. Thus, we currently have no support for recognizing current navigation state based on the URL of the accessed page or on any type of automated analysis of page template.

More significantly, our solution handles only a limited number of tasks related to adapting the navigation plan to real Web site behavior. While it supports the most important tasks of adapting to the actual number of cycles needed in the case of pagination or hierarchies of unknown depth, it has no support for unexpected or unplanned situations, such as passing site limits (of a number of requests or a number of results), layout or data model evolution and data moving between result pages. It has also only limited support for handling duplicate data.

Usage Scenarios

Out of the usage scenarios considered in the last group of challenges, only ad hoc querying is within the direct focus of our thesis (with partial implementation of both challenges related to data prioritization and partial data reception). Our solution has also partial support for some tasks specific to data materialization (building complete queries for Web sites based on navigation and closed domain HTML forms). At the same time, it provides no direct support for challenges specific to data monitoring or integrated querying. However, as discussed in Section 6.3.5, both these tasks can be built on top of our solution.

While our implementation level in this area is low (12%), it is to be noted that only four systems (Information Manifold, DEQUE, TSIMMIS and HiWE) attained significantly higher results, with Information Manifold's implementation level (21%) being the highest.

6.3.3 Applicability in Different Types of Data-Intensive Web Sites

In Section 1.2 we presented a classification of key types of data-intensive Web sites, and presented their specificities. In this section we discuss how well our solution deals with all of these classes of Web sites. We gather the results of our analysis in a synthetic way in Table 6.6.

While our solution is mostly oriented towards more complex Web sites, it is also capable of extracting information from **basic data-intensive Web sites** (including **e-commerce Web sites**). Thanks to the used combination of XPath and regular expression extraction rules, the proposed solution is capable of dealing with diverse Web sites. Moreover, the planned incorporation of GATE or SPRouT rules will even increase this flexibility, by allowing user to combine Web information extraction methods with tools for information extraction from text.

Type of Data-Intensive Web Site	1	2	3
Basic data-intensive Web sites	●	●	●
E-commerce Web sites	●	●	●
Deep Web sites	●	◐	◐
Personalized Web sites	●	●	◐
Adaptive Web sites	●	●	●
Web applications	●	●	●
Collaborative filtering Web sites	●	◐	◐
GUI-centric Web sites	●	●	●
Social networking Web sites	●	●	◐
Generalized data-intensive Web sites	◐	◐	◐

Table 6.6: Types of Data-Intensive Web Sites That Can Be Handled by Proposed Solution: (1) by model or its extensions, (2) by core model and algorithms, (3) by our proof-of-concept implementation

Our solution addresses also many of the challenges of accessing **Deep Web** databases. It supports the filling in of different kinds of Web forms and user controls, including GUI elements that could not be handled by previous solutions. It proposes an approach to navigation between multiple result and details pages that uses different types of Web actions. It also addresses the key challenge of extracting and merging data from multiple pages into records and record sets, as well as the challenge of cutting navigation paths that are not relevant to a user query as quickly as possible, thus limiting the overhead.

There are a few tasks specific to dealing with Deep Web sources that we do not support but that can be easily implemented on top of our solution. They include Deep Web databases probing, dealing with open-domain fields (through different query expansion techniques), integrating data from many Deep Web sources, and Deep Web sources discovery. At the moment it also does not support the modeling of Deep Web sources querying capabilities, but we plan to add this feature by reusing existing approaches.

The proposed model and implemented prototype have also almost complete support for data extraction from **personalized Web sites**. On the one hand, the prototype supports basic authentication methods (relying on any combination of HTML forms and JavaScript code), and other types (such as HTTP authentication or authentication based on a random subset of password characters, as used by many e-banking solutions) can be easily added. On the other hand, thanks to the complete state management proposed as a part of our solution, all issues related to session maintenance and reset are handled.

Navigation state management is also the key feature of our solution that makes it useful for dealing with **adaptive Web sites** and **Web applications**. Both these categories are examples of stateful Web sites that were very hard to use for data extraction by existing solutions. Key elements of our approach to navigation state management, i.e. the representation of the impact and dependency of each data extraction node on specific parts of the navigation state, the description of different types of navigation state modifications, and a few techniques of navigation state recreation, make our solution capable of dealing with almost all types of adaptive Web sites and Web applications.

The navigation state management we propose is limited to modeling the navigation state modified by the navigation actions of a single user. Thus, it cannot be used to model navigation state evolution in **collaborative filtering Web sites**, as it often depends on the actions of thousands of users. However, collaborative filtering content typically evolves slowly and the impact of a single user navigation session is very limited (and in some cases, e.g. when filtering is based on completed transactions, it can be completely avoided). Thus, in many cases, information stored in collaborative filtering Web sites can be treated in the same way as any other content and extracted without any navigation state modeling.

Another important category of data-intensive Web sites consists of **GUI-centric Web sites** that use composite, interactive Web content and AJAX-like techniques. Our solution handles such Web sites in a very complete way. Firstly, our model covers GUI-based Web actions, and our prototype implements a significant number of them. Secondly, thanks to rich Web document representation, the majority of types of composite Web content are handled. Thirdly, thanks to the mechanisms of detecting changes in Web documents and to the ability to work with both user and encoded content, our solution handles well almost all applications of AJAX-like technologies.

Two final groups of data-intensive Web sites discussed in this thesis are **social networking Web sites** and **generalized data-intensive Web sites**. The proposed model and prototype provide a solution for both these categories of Web sites. However, in both cases the actual use of extracted information may require some post-processing.

In the case of social networking Web sites, cyclic graphs with a simple duplicate data handling approach can be used to continuously crawl user profiles (or any other entities linked into a graph), and two attribute sets can be used to represent a graph's vertices and edges (e.g. users and the relations between them). However, for many applications, such an output format would need to be transformed into other data models.

In the case of generalized data-intensive Web sites, the support is even more limited. Our model can be used, for example, to extract records consisting of relation name, attribute name, record identifier and attribute value. However, in that case significant post-processing may be required to make these data useful to the user or any other software component.

6.3.4 Applicability in Different Business Usage Scenarios

In Section 1.3.3 we presented six distinct usage scenarios of applying Web information in business. In this section we discuss how the solution presented in this thesis can be used to handle all of these scenarios. Concise representation of the results of this analysis can be found in Table 6.7.

Suppliers monitoring typically concerns a relatively low number of data-intensive Web sites that do not evolve very quickly. At the same time these Web sites typically require user authentication and gather precise usage statistics.

Therefore, the key challenges in the case of suppliers monitoring consist in limiting as much as possible the number and frequency of HTTP requests, and in making navigation sessions very similar to user behavior. It requires the ability to issue very specific queries with support for the early cutting of irrelevant navigation paths, an implementation of a politeness policy with relatively long times between two consecutive Web actions, and with the ability to choose the most valuable navigation paths based on their utility. As discussed in the previous section, all of these challenges but utility-driven data selection are addressed by our data extraction model and implemented in our proof-of-concept application.

Business Scenario	1	2	3
Suppliers monitoring	●	●	●
Competitors monitoring	●	◐	◐
Foreign markets monitoring	●	◐	◐
Distribution network monitoring	●	◐	◐
Market data collection	●	◐	◐
Data migration between co-operating parties	●	◐	◐

Table 6.7: Business Scenarios Addressed by Proposed Solution: (1) by model or its extensions, (2) by core model and algorithms, (3) by our proof-of-concept implementation

The challenges of **competitors monitoring** and **foreign markets monitoring** are slightly different. In this case the number of Web sites to monitor is typically much higher, and they tend to have much less strict user tracking than authentication-based suppliers' Web sites. Thus, in this case the topics of quick wrapper development and wrapper maintenance become much more important.

At the same time the higher number of sources and the frequent focus on end-users increase the chances that some of the Web sites will use advanced GUI-based and stateful navigation patterns, as well as composite Web content and complex data presentation patterns. Therefore, the requirements of the technical and model-level universality of used solution increase significantly.

The proposed solution currently does not include any features for wrapper induction or maintenance, apart from the possibility of modeling alternative rules (that we believe will be a significant step in the area of wrapper maintenance) by using the current query model. The feasibility of wrapper induction and maintenance in the proposed model are briefly discussed in Section 6.5 of this thesis. By contrast, our approach deals very well with the technical diversity of Web sites, as well as with their statefulness. Therefore it is currently applicable in the best way in the case of a limited number of competitor or foreign market Web sources.

It is to be noted that while data obtained from suppliers Web sites requires integration, competitors and foreign markets monitoring makes these tasks much more difficult because of the higher number and quicker evolution of Web sites of interest. However, as we stated before, the problem of information integration is out of the scope of this thesis, and we believe that it can be handled well by existing research and commercial solutions.

The scenarios of **distribution network monitoring** and **market data collection** are even more challenging, as they combine the challenges of both suppliers and competitors monitoring. Moreover, it may require different types of queries (e.g. queries analyzing products' exposition, such as choosing all products that are exposed better than the item of our interest). Thus, while our solution can be used in this scenario, the space for future optimizations stays significant.

In contrast with already discussed scenarios, in the case of **data migration between co-operating parties** the number of Web sites to process is low, and the problems of being detected as a bot, mimicking user behaviors and limiting the number of requests have secondary importance. However, in this scenario the key aspect concerns the completeness and timeliness of data download. Thus, apart from all mechanisms that our solution already provides (including the capability to create a complete set of queries in Web sites without open domain form fields), it requires a well-performing approach to duplicates removal. As discussed previously, such an approach can be based on current query model that we introduced in Section 4.5.7.

6.3.5 Applicability in Different Technical Usage Scenarios

In Section 2.3.4 we introduced five types of task that rely on Web information extraction. In this section we analyze if our solution provides complete data extraction support for these five tasks. Table 6.8 represents the results of this analysis.

The first of these tasks, which is the **ad hoc querying** of data-intensive Web sites, is central to our interest in this thesis. The proposed model and prototype implementation enable not only data extraction, but also the issuing of typical queries directly to the Web site. Apart from queries that can be directly mapped onto a Web site's GUI elements (e.g. HTML forms), our solution supports also the efficient querying of attributes not present in query forms (with the early cutting of irrelevant navigation paths). Thanks to the flexibility and modular character of our approach, it can be adapted to support more complex queries by reusing existing techniques of query rewriting using views [141].

In our solution we did not include any special support for **integrated data querying**, assuming that the integration layer should be built on top of the querying capabilities of our model. Our approach does not support integration-specific tasks, such as queries rewriting (source selection, query decomposition) and result merging (data cleansing, duplicates removal, schema mapping, values mapping). However, existing algorithms for these tasks use any kind of data extraction solution that have support of simple queries, thus they can be seamlessly integrated with our solution.

Business Scenario	1	2	3
Ad hoc querying	●	●	●
Integrated data querying	●	●	●
Web monitoring	●	◐	◐
Web data materialization	●	◐	◐
Web information restructuring	◐	◐	◐

Table 6.8: Technical Scenarios Addressed by Proposed Solution: (1) by model or its extensions, (2) by core model and algorithms, (3) by our proof-of-concept implementation

The capability of our approach to answer relatively simple queries in an efficient way makes it also directly usable in **Web monitoring** and **Web data materialization** tasks. While both of these tasks require some extra mechanisms (revisit frequency calculation, records linkage, duplicates removal), they can use the unmodified versions of our algorithms for issuing queries to data sources.

Moreover, future performance improvements based on the current query model can be introduced to provide even better support for both these tasks. For example, in the case of Web monitoring, ordering information could be used for the more efficient discovery of monitored entities or for index construction. In the case of Web data materialization, it could improve dealing with duplicates resulting from overlapping queries.

Finally, our solution offers two separate approaches to **Web information restructuring**. Firstly, it is possible to build output in any arbitrary string data format (e.g. XML or RDF) by using constructor output slots. However, it is rather cumbersome and hard to apply for more complex data structures. Secondly, our system can be used as a source of data for any external Web information restructuring tool. While such approach would be flexible and easy to implement, it would also probably have a poor performance.

6.4 Performance of Proposed Data Extraction Algorithm

In this section we analyze the performance of our solution. By this analysis, we aim at both comparing the performance of the proposed solution with previous systems, and at checking impact of the optimization techniques we proposed.

Historically, the performance of information extraction systems was typically measured for low-level information extraction from specific documents. However, as we argued in Section 4.1.1, in the case of navigational information extraction systems, the number of HTTP requests, traditionally excluded from performance analysis, is much more important than the actual time of extracting individual records or attributes. Thus, in this section we use the number of issued HTTP requests as the only measure of performance, assuming that systems capable of limiting the number of requests would have a significantly lower execution time than systems with high overhead.

The experimental setup of our analysis is described in Section 6.4.1, and the results are discussed in Section 6.4.2.

6.4.1 Experimental Setup

In order to demonstrate how the performance of our solution differs from previous work, we compare it with two data extraction systems. The first of them (S1) is DWDI – one of the few existing data extraction systems with strong navigation capabilities, the only one with some support for stateful Web sites, and at the same time one of a few systems that were available to the author of this thesis (both in a form of source code and as a compiled version). During our experiments we used our own implementation of DWDI algorithms, based on its source code. The second system (S2) was developed specifically for this thesis, and simulates the behavior of systems that perform the crawling of data-intensive Web sites with a *post factum* data extraction from all obtained pages (as in the case of the Deonodo Platform).

These two systems are compared to two variants of our algorithms, differing by the set of used optimization techniques. The first algorithm (A1) uses filtering attribute sets as a single query plan optimization technique. The second variant of our algorithm (A2) additionally uses automatically added request capture nodes to limit the need of navigation state recreation.

It is to be noted that in this experiment we focus solely on measuring the number of HTTP requests. Thus, we do not analyze here node mergers and partial linearization, as these two techniques decrease only the total number of nodes and the scope of needed provenance-aware joins, and have no impact on the number of HTTP requests.

In our experimental verification of performance we use a number of scenarios. Each of these scenarios consists of a Web site and a specific user query. In our analysis we use six out of the 40 Web sites described in Section 6.2, which reflect different navigation patterns. Three of them are stateless (TS1, TS3, TS6) and three are stateful (TS2, TS14, TS16).

TS1 is the simplest case of a Web site with hierarchical navigation based on hyperlinks. It contains three classes of pages. The first of them is the home page that contains a list of car makes in our database. Each of the makes forms a hyperlink that leads to a page belonging to the second class. This class consists of pages that contain the list of models and car bodies in a specific make. Each combination of make and body is a hyperlink pointing to the page that contains the list of individual versions with all associated engine and gearbox parameters.

TS2 has exactly the same structure as TS1. However, instead of using traditional hyperlinks, it relies on two types of JavaScript links (links using `javascript:` protocol, and links with attached `onclick` event handler). As a result, it becomes both dependent on and having an impact on the current Web document, as presented in Example 4.6.10, and is an instance of client-side statefulness.

In TS3 the home page and the list of versions are identical, as in the case of TS1. However, the list of models and bodies within a specific make is represented in a very different way. Instead of the list of models and bodies contained in a single page (as in case of TS1), models and bodies are organized into a sequence of paginated results (one combination of model and body per page). As a result, accessing the ith model and body combination within a given make, requires accessing the first page and following the "Next" link $i - 1$ times. As in TS1, each model and body combination forms a link to the page containing the versions of a given model and body.

TS6 is an example of a Web site with alternative navigation patterns. It provides two distinct ways to navigate from the home page to a page containing the list of specific car versions. The first way is almost identical as in the case of TS1. At the home page one can select a specific make (by clicking a link), and then on the next page choose a combination of model and engine displacement. Alternatively, a user can choose the engine displacement in the home page, and the combination of make and model on the second page. Thus, TS6 is a prototypical example of faceted navigation, which is used widely for multi-step data filtering.

TS14 is an example of complex server-side statefulness. It implements a clipboard for saving interesting cars. Once a combination of model and body is placed in the clipboard, it becomes possible to access the contents of the clipboard that contains all versions corresponding to the added model and body combinations. However, accessing the list of versions without placing an item in the clipboard is impossible. To complete the stateful functionalities, TS14 provides also a special Web action that resets the contents of the clipboard (i.e. removes all items it contains). The provided features correspond well to the shopping cart functionality of many on-line shops, which make some part of information (e.g. transportation fees) available only after items are placed in the shopping cart.

Finally, TS16 basically mimics the navigation structure of TS1. However, it introduces two important innovations. Firstly, instead of performing a link-based complete page reload, it uses AJAX to download new content. Thus, once we choose a specific make, the list of its models and bodies is loaded via an AJAX request and similar asynchronous server communication is used to load the list of versions in s specific model and body combination. Secondly, each loaded list of versions does not replace the previous one, but is inserted at the beginning of the main part of the page, and i marked as "new content" by different background color.

No	Constraints	Attributes to Extract
Q1	-	all
Q2	`make_name = 'Peugeot'`	all
Q3	`make_name = 'Peugeot'`	`model_name`, `body_name`
Q4	`body_name = '5D'`	all
Q5	`engine_displacement = '1.4'`	all

Table 6.9: Queries Used for Performance Assessment

In each of the Web sites we execute five different queries, as described in Table 6.9. The first of them (Q1) consists in downloading all data available in the Web site. Three queries (Q2, Q4 and Q5) aim at extracting all attributes (i.e. all attributes listed in Figure 6.1, except for id numbers) and differ only by the attribute used to define a constraint. Finally, one query (Q3) concerns only a limited subset of all attributes.

A total of 30 scenarios were executed for each of the tested systems and algorithms. For each scenario the number of all HTTP requests issued was counted (including HTTP requests used for navigation state recreation). The results of these experiments are discussed in the following section.

6.4.2 Results

In this section we gather the results of our experiments. We start by comparing the performance of our system to DWDI and crawler-like systems, based on five Web sites and five queries. Then, we analyze the impact of the request capture technique on the data extraction overhead, based on five queries and the single test Web site where this technique was applicable.

In Table 6.10 we compare the number of HTTP requests needed in our solution with the number required by DWDI (S1) and systems based on the basic crawling approach (S2). In this table we provide the results for only five out of six Web sites. The reason for this is that TS2 was not only unhandled by any of previous systems, but was also the only test Web site that enabled us to demonstrate the performance effects of capture request techniques. Thus, the results for TS2 will be discussed separately.

Web site	S1	S2	Query	A1	Δ S1 to A1	Δ S2 to A1
TS1	1431	511	**Q1**	511	64,3%	0,0%
			Q2	22	98,5%	95,7%
			Q3	2	99,9%	99,6%
			Q4	168	88,3%	67,1%
			Q5	511	64,3%	0,0%
TS3	5294	955	**Q1**	955	82,0%	0,0%
			Q2	41	99,2%	95,7%
			Q3	21	99,6%	97,8%
			Q4	955	82,0%	0,0%
			Q5	955	82,0%	0,0%
TS6	4569	1610	**Q1**	1610	64,8%	0,0%
			Q2	113	97,5%	93,0%
			Q3	83	98,2%	94,8%
			Q4	1610	64,8%	0,0%
			Q5	199	95,6%	87,6%
TS14	2385	n/a	**Q1**	1464	38,6%	n/a
			Q2	61	97,4%	n/a
			Q3	2	99,9%	n/a
			Q4	435	81,8%	n/a
			Q5	1464	38,6%	n/a
TS16	n/a	n/a	**Q1**	511	n/a	n/a
			Q2	22	n/a	n/a
			Q3	2	n/a	n/a
			Q4	168	n/a	n/a
			Q5	511	n/a	n/a

Table 6.10: Comparison of Our Solution Performance With Previous Systems

Table 6.10 has the form of a cross table. Its rows correspond to all 25 combinations of the test Web sites (TS1, TS3, TS6, TS14, TS16) and queries (Q1, Q2, Q3, Q4, Q5). Table columns correspond to the number of requests required by previous systems (S1, S2) and by our solution (A1). In the two last columns we demonstrate the percent reduction in the number of HTTP requests of our approach by comparison with S1 and S2. It is to be noted that in this table we do not provide any results for A2, as the capture request optimization technique was not applicable in any of the listed Web sites.

Both DWDI and crawler-like systems focused on accessing Web sites (including Deep Web sources) and extracting data from them. However, they did not provide any navigation path cutting facilities (except for the removal of duplicate paths). As a result, any data filtering was performed after the data were completely extracted from the source. Thus, in the case of both S1 and S2, the number of requests needed by all five test queries was equal and corresponded to the download of all data.

As our approach is capable of performing navigation path cutting based on partially extracted data, the results of our approach are different for each Web site and query. The only exception is TS16, which needs always exactly the same number of requests as TS1. Even if it is an example of a stateful Web site, we attained this result by transforming its cumulative statefulness (with navigation state part corresponding to the main part of the page) into a replacement statefulness of navigation state part corresponding to the newly loaded data (see Example 4.5.5 for details).

While all Web sites contain basically the same data, the number of requests necessary to access all of them differs significantly. It is visibly higher in the case of TS3 (which uses pagination) and TS6 (where we modeled both alternative navigation paths, which resulted, for all systems, in the extraction of duplicate data). The number of requests in the case of extracting all data was the lowest in the case of TS1.

There are also significant differences between the number of requests required by DWDI and crawler-like systems. While the approach of DWDI, based on complete path recreation, ends up with a number of requests up to over five times higher than in crawler-like systems (TS3), it also makes DWDI capable of handling some server-side stateful Web sites, including TS14, which cannot be navigated by S2.

In the case of downloading all data (Q1 query) our approach has exactly the same results as S2 (except for TS14 and TS16 that cannot be handled by crawler-like systems). At the same time all results of our approach are significantly better than in the case of DWDI.

Even in the case of TS14, where both our solution and DWDI need to perform additional navigation state resets, our approach needs almost 40% less HTTP requests than DWDI. This results from the different modeling of state resets. In the case of DWDI, a reset Web action needed to be added as the first step of navigation. Thus, in DWDI each complete navigation path was a sequence of navigation state reset Web action, navigation to home page, navigation to make page, the Web action of adding a combination of model and body to the clipboard, and of accessing clipboard contents. In our approach the reset Web action is performed before adding an item to the clipboard, and does not require repetition of navigation to the home page and to a page containing models and bodies combinations.

These numbers demonstrate that in the case of downloading all data from a Web site, our solution always generates a lower overhead than DWDI and is not worse that S2. However, the results are even more impressive in the case of specific queries, when the advantages of the early cutting of useless navigation paths becomes apparent.

In the case of all queries, the last column of the table shows the % reduction of the number of requests by using our approach instead of crawler-like systems, i.e. the reduction due to the fact of the early cutting of navigation paths. In the case of some queries applied in specific Web sites (e.g. Q5 issued to TS1 or Q4 issued to TS6), no reduction is observed. However, in the majority of cases a reduction by over 80% (with 99.6% being the best result) is obtained. The reduction is even higher (up to 99.9%) in the case of comparison with DWDI. In that case the result is better not only because path cutting is applied (TS1, TS3, TS6), but also because our approach to navigation state recreation is more selective (TS14).

System	TS2				
	Q1	Q2	Q3	Q4	Q5
A1	987	41	2	297	987
A2	511	22	2	168	511
Δ A1 to A2	48,2%	46,3%	0,0%	43,4%	48,2%
A1 / TS1	987	41	2	297	987

Table 6.11: Impact of Automated Introduction of Capture Request Nodes onto Performance of Our Solution

As we mentioned before, out of six used test Web sites TS2 is the only one where capture request optimization technique can be applied. At the same time, due to technical reasons (missing support for GUI-based Web actions), this Web site cannot be handled by S1 and S2. Thus, we present the results for this Web site separately in Table 6.11, and use them to compare the performance of the A1 and A2 variants of our algorithm.

In the case of the TS2 Web site, applying the request capture technique completely removes the issues related to client-side statefulness, and makes the content accessible in exactly the same number of requests as in the case of the querying of link-based Web site TS1 (results for TS1 are repeated in the last row for reference). As we can see in the table, this corresponds to an almost 50% reduction of the number of needed HTTP requests (with the exception of the Q3 query that requires both in the case of A1 and A2 only two HTTP requests). Therefore, while it concerns a specific subset of all complex data-intensive Web sites, it has a clearly positive effect on the data extraction overhead.

6.5 Future Research Directions

In this thesis we presented a novel approach to extracting information from complex data-intensive Web sites. While our solution is capable of handling many more challenges related to extraction from these Web sites than any previous solution, there are multiple Web information extraction issues that it currently does not address. In this section we present a few key research directions for our future work, and discuss the feasibility of addressing challenges that currently are not handled.

There are three crucial directions of planned future improvements of the approach presented in this thesis: extending the data extraction model, adding specialized variants of algorithms and providing support for learning data extraction graphs.

Data Extraction Model Extensions

The first of the extensions we plan to introduce to our model consists in adding recognizer nodes. We mentioned the basic type of recognizers, i.e. the data extraction rules used to identify the class of the current Web document, while discussing how our model deals with non-deterministic navigation patterns. However, we plan to support one more type of recognizer, called event detection recognizer. This type of recognizers will be responsible for detecting that some Web action has been completely done, e.g. by looking for some characteristic elements of Web page or characteristic request-response communication to the HTTP server. Introducing this type of recognizer will make it possible to deal with all kinds of system-triggered Web actions and will give even better support for all situations when content is downloaded or generated asynchronously.

The second significant extension of our model concerns adding an explicit description of the querying capabilities to all Web action templates that require external input. It concerns HTML forms and other multi-field user interface components, but also all Web actions that consist in the manual construction of HTTP requests with multiple input slots. Querying capabilities description should cover such aspects as the allowed and disallowed combinations of queried attributes, constraints on the format and scope of provided values, and specification of what is a default value of the omitted input slot assigned by the server (if any). While currently multiple allowed combinations of allowed input slots could be attained by creating a separate Web action template for each combination, adding querying capabilities description, after minor query planning algorithm adaptations, will make it possible to build concise data extraction graphs that adapt to the scope of the attributes specified by the user.

New Variants of Algorithms

The first modification of our algorithms that we plan to develop in the future concerns the parallel or distributed execution of data extraction graphs. While this modification will have a significant impact on the behavior and applicability of our solution in real-life situations, the scope of the required adaptations is limited, and concerns mostly queue management subroutines. The most important challenge to address concerns combining state-aware queue management with a separate modeling of the current navigation state for each of the data extraction agents or threads.

The key planned modification to the query planning algorithm focuses on using additional information, including the current query model and querying capabilities description, for developing better-performing query plans. As discussed in Section 4.5.7, using the current query model during query planning will make it possible to use in-built Web site data ordering techniques and to avoid data duplicates resulting from overlapping queries. At the same time, using the querying capabilities description will allow us to build query plans in cases when user input is not complete, but such incompleteness is accepted by the source being queried.

Using the current query model enables us to distinguish between paths in the graph that correspond to exactly identical query from the paths that correspond to different data with a shared schema. In the case of multiple sets of records that have only a shared schema, extraction output should contain the union of all these sets. By contrast, paths with identical queries will contain duplicates and should be normally avoided. However, such alternative paths to the same data can be also used to verify if the wrapper still functions well. Thus, we plan to use the current query model also as part of the failure detection mechanism.

Learning Support

As we discussed in Chapter 4, our current solution introduces the model of data extraction graphs, but does not provide any support for learning graphs for specific Web sites. As a result, the preparation and maintenance of wrappers for our system requires significant manual effort. Therefore an important part of our future effort will be focused on learning data extraction graphs for specific Web sites.

While our data extraction model presents an integrated approach to navigation and data extraction, the task of data extraction graphs learning can be easily decomposed into a few more specific tasks. Some of these tasks are well known problems in the area of Web information extraction. The first of them concerns the clustering of Web documents into groups characterized by similar templates and data schemas. The second task concerns learning data extraction rules from a set of similar Web documents. The third task consists in mapping individual attributes returned by extraction rules onto the attributes of some relational schema. Finally, the parsing of user interface (analyzing Web forms and other GUI element, discovering and clustering links), can be interpreted as a special case of extraction rules learning.

For all of these tasks there are many existing solutions that can be used as an inspiration for our future work. However, our model covers also the modeling of the navigation state and of the current query. To the best of our knowledge, the learning of statefulness and current query patterns have not been addressed by any existing Web information extraction system. We believe that the task of defining the navigation state parts existing in a Web site is infeasible in an automated way. However, detecting some of the statefulness patterns (i.e. dependency and impact on somehow observable parts of the navigation state) can be at least partially automated. Similarly, the majority of the current query constructs can be automatically discovered by heuristics and experimentation. These two topics are among our most interesting plans of future research.

6.6 Summary

In this chapter we evaluated the data extraction model proposed in Chapter 4 and its implementation described in Chapter 5. Our evaluation consisted of a few steps. Firstly, in Section 6.1 we performed a qualitative comparison of our approach with the three most similar previous systems. Secondly, in Section 6.2 we presented the data set of test Web sites used for detailed evaluation, and discussed how well our solution deals with wrapping these sources. Next, in Section 6.3 we assessed the generality of the proposed solution with respect to the research objectives set in Chapter 4, the Web information extraction challenges listed in Table C.1, the different types of data-intensive Web sites described in Section 1.2, the different business usage scenarios listed in Section 1.3.3 and various technical use cases introduced in Section 2.3.4. Then, in Section 6.4 we analyzed the performance of the proposed solution and individual optimization techniques. Finally, in Section 6.5 we described our key future research directions.

Conclusions

In this thesis we studied the problem of extracting structured information from complex data-intensive Web sites. Our research followed strictly the design-science information systems methodology proposed by Hevner and colleagues [152]. The non-original part of our research, covering "external" components of the used methodology (see: Figure 1), included a study of the organizational (Chapter 1) and technical (Chapter 2) environment of our research problem and a in depth review of the knowledge base of existing solutions (Chapter 3). The original part of our research included the development of our models and methods (Chapter 4) and their instantiation in a form of software prototype (Chapter 5), as well as an evaluation of the proposed approach (Section 6).

During our research we followed guidelines of applied methodology. The results of our research have a form of formalized models and methods (algorithms), together with their instantiation (guideline 1). The developed solution is technology-based, it concerns an important business problem of electronic commerce businesses, and provides an innovative solution to the unsolved problem of information extraction from complex data-intensive Web sites (guideline 2). We rigorously applied the techniques of simulation, scenarios analysis and informed argument in order to analyze the generality and performance of our solution (guideline 3), demonstrating that our approach both enables data extraction from some classes of previously unhandled data-intensive Web sites and minimizes data extraction overhead for a few other classes (guideline 4). During our study of the environment we combined the techniques of literature review, model-based analysis and data collection, and the presentation of our approach relied on the formalization of our models and methods (guideline 5). While in this thesis we present only the final models and methods, they were preceded by a few iterations of method design and evaluation (guideline 6). Some of them [112, 113, 9] were already presented to researchers and practitioners, and we plan to widely disseminate also the results of our research presented in this thesis (guideline 7).

The results of our research were presented in six chapters. We started by introducing key aspects of our research area. Firstly, we analyzed the different levels of information structure and the transformations between them (Section 1.1). Next, we discussed the abundance of different data-intensive Web sites, including complex data-intensive Web sites (Section 1.2). We concluded Chapter 1 by describing our direct motivation to undertake our research, i.e. the applicability of Web information in electronic commerce, illustrated by specific application scenarios, such as the monitoring of suppliers, competitors, distribution networks and foreign markets (Section 1.3).

We started Chapter 2 by a brief overview of different top-down and bottom-up efforts of structuring Web content, including Web content mining, Web structure mining and Web usage mining (Section 2.1). Next we introduced the research area of information extraction, the research problem of Web information extraction, and its specific variant addressed in this thesis, related to extracting information from complex data-intensive Web sites. We also defined a few useful terms, including extraction rules and wrappers (Section 2.2). In the following section, we presented a study in which we used four distinct information sources to compile a list of Web information extraction challenges. We started with an extensive literature review (Section 2.3.1). Next, we used our model of user interaction with Web content, consisting of nine components and 26 interaction stages to approach information extraction challenges in a more theoretical way (Section 2.3.2). As the third source of challenges, we performed an analysis of 41 real-life data-intensive Web sites out of 75 very diverse candidates in 24 categories (Section 2.3.3). Finally, we took the perspective of a few tasks that rely on Web information extraction, such as ad hoc querying, integrated data querying and Web monitoring (Section 2.3.4). As a result of this analysis we presented (Appendix C) and discussed (Section 2.3.5) the list of 336 challenges, grouped into 9 first-level groups, 25 second-level groups and 253 individual topics.

Chapter 3 was devoted to the description and comparison of existing Web information extraction solutions. We started by combining existing schemes for comparing different information extractions systems and extending them with a few new ideas. As a result, we proposed our own comparison scheme, which covered extraction output characteristics, extraction rules characteristics and the used approach to extraction rules management (Section 3.1). Next, we provided a detailed description of 42 information extraction systems, developed from 1994 to 2008, and we listed over 50 other systems not described in detail (Appendix D). Consecutively, we applied our comparison scheme, the list of identified challenges and a graph-based approach to representing ideas flow, in order to analyze 39 of these systems (Section 3.2). As part of this analysis we presented a feature-based, challenges-based and genetic analysis of previous work. We ended the chapter by identifying data extraction challenges not addressed by previous data Web information extraction systems, and by proposing key research directions to follow (Section 3.3).

In Chapter 4, which is the essential part of this thesis, we provided a detailed presentation of our approach to data extraction from complex data-intensive Web sites. Given our research problem defined in Chapter 2 and the information extraction challenges that were not addressed by previous solutions discussed in Section 3.3, we defined and discussed 25 detailed objectives of our research (Section 4.1). Seven of them were very strictly connected to mechanisms typical for complex data-intensive Web sites and were selected as our main research objectives.

We devoted the remaining part of the chapter to the step-by-step presentation of our data extraction approach. Whenever possible, we combined intuitive overview, specific real-life examples and formalization. The presentation of our solution started with a static view of our data extraction model, which covered the basic terms of navigation state, Web action and Web action template, data extraction rule, data extraction rules composition and data extraction rules templates, as well as data records, data records schema, and attribute set (Section 4.2).

Next, in Section 4.3, we demonstrated how these basic components can be generalized into data extraction nodes, instances and atoms, and how multiple data extraction nodes can be connected by input mappings to form a data extraction graph. Consecutively, we discussed how nodes with multiple input and output slots, as well as input mappings with several source and destination slots, can be used to model the complexities of data-intensive Web sites. We also provided the first elements of a dynamic view of data extraction graphs, by introducing the conditions for data extraction graph executability. Finally, we discussed how cycles in data extraction graphs can be used to model complex situations, including pagination and data hierarchies of unknown depth.

Consecutively, in Section 4.4, we presented a set of algorithms that enable execution of the data extraction graph, i.e. the actual data extraction from Web sites modeled with such graphs. A total of 12 algorithms were presented, including algorithms enabling the analysis of graph structure and its executability, the algorithm of actual graph execution, and its key subroutines related to queue management and data management. In the case of subroutines related to queue management, we presented two basic variants, corresponding to depth-first and breadth-first graph execution, and discussed the possibility of using more sophisticated algorithms in specific situations. Adaptations of these algorithms to different Web site statefulness scenarios, altogether with related extensions of the data extraction graph model were presented in Section 4.5.

In the final part of this chapter (Section 4.6) we focused on query planning, i.e. on the ability of our approach to transform a user query and data extraction graph corresponding to a Web site into an executable data extraction graph that corresponds to this query. We started by presenting basic query planning tasks, including the selection of the subgraph that is required to answer the query, the incorporation of user input (query constraints) into the query plan and the filtering of acquired data. Then, we presented a few query plan optimization techniques related to the early filtering of data and limiting the need of recreations of the navigation state. Finally, we presented a complete query planning algorithm with all required subroutines and optimization techniques.

While the data extraction graph model and related algorithms of graph execution and query planning are the essential results of our research, the role of prototype implementation is also significant. Firstly, it made possible the evaluation of our approach. Secondly, our implementation also included a few technical novelties unseen in previous solutions. For that reason we devoted the whole of Chapter 5 to describing the proof-of-concept implementation of our algorithms. We started by presenting the extensible architecture of our system and introducing our approach to Web server interaction that combines the use of an embedded Web browser and a pass-through proxy server (Section 5.1). In the remaining part of this chapter (Sections 5.2 and 5.3) we presented the individual data extraction graph nodes and algorithms implemented in our prototype.

In Chapter 6, which concludes our thesis, we presented the discussion and evaluation of the proposed solution. We started by comparing our approach with the three most similar previous solutions (Araneus, DWDI and Denodo Platform), both by applying our comparison scheme proposed in Chapter 3, and by discussing key similarities and differences concerning used data extraction and Web navigation methods (Section 6.1). Next, we presented the set of 40 test Web sites we developed to enable a repetitive evaluation of our approach and the identification of challenges that it does and does not handle (Section 6.2).

The following part of the final chapter is directly related to two key aspects of the thesis we defended in this dissertation. In Section 6.3 we demonstrated that our approach "enables information extraction from a significantly larger set of complex data-intensive Web sites than in the case of previous solutions". We documented the generality of the proposed solution in a few ways. Firstly, we showed that we had attained the majority of 25 detailed research objectives defined in Chapter 4. Secondly, we used the list of Web information extraction challenges described in Chapter 2 and the challenge-handling evaluation method presented in Chapter 3 to demonstrate that our solution deals with a significantly higher percentage of all identified challenges than any of the previous systems. Moreover, we showed that in five out of nine groups of challenges it is significantly more general than the best solution in a specific group, and there are only two groups of challenges where some previous systems address significantly more challenges. Thirdly, we discussed the ability of our solution to deal with different types of data-intensive Web sites (Section 6.3.3), its applicability in different business use cases (Section 6.3.4), and its relevance to various technical usage scenarios (Section 6.3.5).

In Section 6.4 we focused on the second essential aspect of the thesis defended in this dissertation, i.e. the ability to perform data extraction with low overhead. We compared experimentally the overhead of our approach (with and without capture request optimization techniques) with our implementations of the DWDI algorithm and crawler-like algorithms used e.g. by the Denodo Platform. Our experiment covered six diverse stateless and stateful Web sites (a subset of our test Web sites) and five different queries. Our experimentation demonstrated that in the case of downloading all data from a Web site, our approach assured an overhead that was not higher than in the case of previous solutions, while in the case of specific queries the query planning capabilities of the proposed solution led to up to a 99.9% reduction in the number of HTTP requests. We also demonstrated that in the case of some stateful Web sites, the request capture optimization technique can significantly decrease data extraction overhead.

While our solution is significantly more general than any previous solution, and assures low data extraction overhead, there are still some important situations that remain unhandled by our approach. In Section 6.5, which concludes Chapter 6, we discussed our plans for future research in three distinct areas: extending the data extraction model (mostly by adding recognizer nodes and the explicit description of querying capabilities), providing new variants of algorithms (including the parallel or distributed graph execution algorithm), and providing support for the automated learning of data extraction graphs.

Bibliography

[1] Cascading style sheets, level 2, CSS2 specification, 1998. [cited at p. 6]

[2] From semistructured data to XML: Migrating the lore data model and query language. In *2nd International Workshop on The Web and Databases*, 1999. [cited at p. 268]

[3] RSS 2.0 specification (version 2.0.10), 2007. [cited at p. 24]

[4] Rocio Abascal and J. Alfredo Sanchez. X-tract: Structure extraction from botanical textual descriptions. In *Symposium on String Processing and Information Retrieval Symposium*, pages 2–7, 1999. [cited at p. 268]

[5] Ahmed Abbasi, Hsinchun Chen, and Arab Salem. Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums. *ACM Transactions on Information Systems*, 26(3):12:1–12:34, 2008. [cited at p. 26]

[6] Ajith Abraham. Business intelligence from web usage mining. *Journal of Information & Knowledge Management*, 2(4):375–390, 2003. [cited at p. 27]

[7] Witold Abramowicz. *Filtrowanie informacji.* Wydawnictwo Akademii Ekonomicznej w Poznaniu, 2008. [cited at p. 3]

[8] Witold Abramowicz and Dominik Flejter, editors. *Business Information Systems Workshops, BIS 2009 International Workshops, Poznan, Poland, April 27-29, 2009. Revised Papers.* Springer, 2009. [cited at p. ix, 12]

[9] Witold Abramowicz, Dominik Flejter, Tomasz Kaczmarek, Monika Starzecka, and Adam Walczak. Semantically enhanced deep web. In *3rd International AST Workshop*, pages 675–680, 2008. [cited at p. 9, 82, 195, 269]

[10] Russell L. Ackoff. From data to wisdom. *Journal of Applied Systems Analysis*, 16:3–9, 1989. [cited at p. 2, 3]

[11] Brad Adelberg. NoDoSE – a tool for semi-automatically extracting structured and semistructured data from text documents. In *1998 ACM SIGMOD International Conference on Management of Data*, pages 283–294, 1998. [cited at p. 250]

[12] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and processing (W3C editorś draft 25 january 2008), 2008. [cited at p. 24]

[13] Eugene Agichtein, Chris Burges, and Eric Brill. Question answering over implicitly structured web content. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 18–25, 2007. [cited at p. 269]

[14] George A. Akerlof. The market for lemons: Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics*, 84(3):488–500, 1970. [cited at p. 18]

[15] Manuel Alvarez, Alberto Pan, Juan Raposo, and Justo Hidalgo. Crawling web pages with support for client-side dynamism. In *7th International Conference on Web-Age Information Management*, pages 252–262, 2006. [cited at p. 260]

[16] Manuel Alvarez, Alberto Pan, Juan Raposo, and Angel Vina. Client-side deep web data extraction. In *IEEE International Conference on E-Commerce Technology for Dynamic E-Business*, pages 158–161, 2004. [cited at p. 260]

[17] Manuel Alvarez, Juan Raposo, Fidel Cacheda, and Alberto Pan. A task-specific approach for crawling the deep web. *Engineering Letters*, 13(2), 2006. [cited at p. 260]

[18] Manuel Alvarez, Juan Raposo, Alberto Pan, Fidel Cacheda, Fernando Bellas, and Victor Carneiro. DeepBot: A focused crawler for accessing hidden web content. In *3rd international workshop on Data enginering issues in E-commerce and services*, pages 18–25, 2007. [cited at p. 7, 260]

[19] Manuel Alvarez, Juan Raposo, Angel Vina, and Alberto Pan. Automatic wrapper maintenance for semi-structured web sources using results from previous queries. In Hisham M. Haddad, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright, editors, *2005 ACM Symposium on Applied Computing*, pages 654–659, 2005. [cited at p. 260]

[20] Yoo Yung An, James Geller, Yi-Ta Wu, and Soon Ae Chun. Automatic generation of ontology from the deep web. In *18th International Conference on Database and Expert Systems Applications*, pages 470–474, 2007. [cited at p. 269]

[21] Vinod Anupam, Juliana Freire, Bharat Kumar, and Daniel Lieuwen. Automating web navigation with the WebVCR. In *9th International Conference on World Wide Web*, pages 503–517, 2000. [cited at p. 255]

[22] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. Technical report, 2002. [cited at p. 258]

[23] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *2003 ACM SIGMOD International Conference on Management of Data*, pages 337–348, 2003. [cited at p. 258]

[24] Norm Archer and Judith Gebauer. *B2B Applications to Support Business Transactions: Overview and Management Considerations*, pages 22–50. Idea Group Publishing, 2002. [cited at p. 16, 17]

[25] Liliana Ardissono, Cris Barbero, A. Goy, and Giovanna Petrone. An agent architecture for personalized web stores. In *3rd International Conference on Autonomous Agents*, pages 182–189, 1999. [cited at p. 9]

[26] Gustavo O. Arocena. WebOQL: Exploiting document structure in web queries. Master's thesis, University of Toronto, 1997. [cited at p. 249]

[27] Gustavo O. Arocena and Alberto O. Mendelzon. WebOQL: Restructuring documents, databases, and webs. In *14th International Conference on Data Ingeneering*, 1998. [cited at p. 249]

[28] Naveen Ashish and Craig A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *2nd International Conference on Cooperative Information Systems*, pages 160–169, 1997. [cited at p. 246]

[29] William Aspray and Paul E. Ceruzzi, editors. *Discovering a Role Online — Brick-and-Mortar Retailers and the Internet.* MIT Press, 2008. [cited at p. 16]

[30] Paolo Atzeni and Giansalvatore Mecca. Cut and paste. In *16th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 144–153, 1997. [cited at p. 248]

[31] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. Semistructured and structured data in the web: Going back and forth. *SIGMOD Record*, 26(4):16–23, 1997. [cited at p. 248]

[32] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. To weave the web. In *23rd International Conference on Very Large Data Bases*, pages 206–215, 1997. [cited at p. 248]

[33] Fabien Azavant and Arnaud Sahuguet. Bulding light-weight wrappers for legacy web data sources using W4F. In *25th International Conference on Very Large Data Bases*, 1999. [cited at p. 252]

[34] Ricardo Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, 1999. [cited at p. 24]

[35] Ranieri Baragila and Fabrizio Silvestri. Dynamic personalization of web sites without user intervention. *Communications of ACM*, 50(2):63–67, 2007. [cited at p. 9]

[36] Luciano Barbosa and Juliana Freire. Searching for hidden-web databases. In *8th International Workshop on the Web and Databases*, pages 1–6, 2005. [cited at p. 7]

[37] Luciano Barbosa and Juliana Freire. Combining classifiers to identify online databases. In *16th International Conference on World Wide Web*, 2007. [cited at p. 32]

[38] Luciano Baresi, Franca Garzotto, and Paolo Paolini. From web sites to web applications: New issues for conceptual modeling. In *Conceptual Modeling Approaches for E-Business*, pages 89–100, 2000. [cited at p. 11]

[39] Robert Baumgartner, Michal Ceresna, and Gerald Ledermüller. Deep web navigation in web data extraction. In *International Conference on Intelligent Agents, Web Technologies and Internet Commerce*, pages 698–703, 2005. [cited at p. 255]

[40] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Declarative information extraction, web crawling, and recursive wrapping with lixto. In *6th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 21–41, 2001. [cited at p. 255]

[41] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Supervised wrapper generation with lixto. In *27th International Conference on Very Large Data Bases*, pages 715–716, 2001. [cited at p. 255]

[42] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *27th International Conference on Very Large Data Bases*, pages 119–1128, 2001. [cited at p. 255]

[43] Dave Beckett and Brian McBride. RDF/XML syntax specification, 2004. [cited at p. 24]

[44] Thomas J. Beckman. *The current state of knowledge management*, pages 1.1–1.22. CRC Press, 1999. [cited at p. 3]

[45] Gene Bellinger, Durval Castro, and Anthony Mills. Data, information, knowledge, and wisdom, 2004. [cited at p. 3, 4]

[46] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Benaventano. Semantic integration of heterogeneous information sources. *Data & Knowledge Engineering*, 36(3):215–249, 2001. [cited at p. 268]

[47] Andre Bergholz and Boris Chidlovskii. Learning query languages of web interfaces. In *2004 ACM Symposium on Applied Computing*, pages 1114–1121, 2004. [cited at p. 9, 269]

[48] Michael K. Bergman. The deep web: Surfacing hidden value. *The Journal of Electronic Publishing*, 7(1), 2001. [cited at p. 1, 7]

[49] Gerd Beuster, Bernd Thomas, and Christian Wolff. MIA: An ubiquitous multi-agent web information system. In *International ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce*, pages 11–13, 2000. [cited at p. 268]

[50] Sourav S. Bhowmick, Sanjay K. Madria, and Wee Keong Ng. *Web Data Management.* Springer, 2004. [cited at p. 45]

[51] Jeffrey P. Bigham, Anna Cavender, Ryan S. Kaminsky, Craig Prince, and Tyler Robison. Transcendence: Enabling a personal view of the deep web. In *2008 International Conference on Intelligent User Interfaces*, pages 169–178, 2008. [cited at p. 9, 269]

[52] Michael Bloch, Yves Pigneur, and Arie Segev. Leveraging electronic commerce for competitive advantage: a business value framework. In *9th International EDI-IOS Conference*, pages 91–112, 1996. [cited at p. 14]

[53] Jose Borges and Mark Levene. Data mining of user navigation patterns. In *Workshop on Web Usage Analysis and User Profiling*, pages 31–36, 1999. [cited at p. 25]

[54] Vijay Boyapati, Kristie Chevrier, Avi Finkel, Natalie Glance, Tom Pierce, Robert Stockton, and Chip Whitmer. ChangeDetector: a site-level monitoring tool for the WWW. In *11th International Conference on World Wide Web*, pages 570–579, 2002. [cited at p. 268]

[55] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Francios Yergeau, and John Cowan. Extensible markup language (XML) 1.1 (second edition), 2006. [cited at p. 23]

[56] John Breslin, Andreas Harth, Uldis Bojars, and Stefan Decker. Towards semantically-interlinked online communities. In *2nd European Semantic Web Conference*, pages 500–514, 2005. [cited at p. 24]

[57] Dan Brickley and Libby Miller. FOAF vocabulary specification 0.91, 2007. [cited at p. 24]

[58] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2):87–129, 1996. [cited at p. 9]

[59] Alex G. Büchner and Maurice D. Mulvenna. Discovering internet marketing intelligence through online analytical web usage mining. *SIGMOD Record*, 27(4):54–61, 1998. [cited at p. 27]

[60] Peter Buneman, Mary Fernandez, and Dan Suciu. UnQL: a query language and algebra for semistructured data based on structural recursion. *The VLDB Journal*, 9(1):76–110, 2000. [cited at p. 268]

[61] David Buttler, Ling Liu, and Calton Pu. A fully automated object exraction system for the world wide web. In *21st International Conference on Distributed Computing Systems*, pages 361–370, 2001. [cited at p. 257]

[62] Deng Cai, Deng Cai, Shipeng Yu, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Extracting content structure for web pages based on visual representation. In *5th Asia Pacific Web Conference*, 2003. [cited at p. 262]

[63] Carlos Castillo. *Effective Web Crawling.* PhD thesis, Universidad de Chile, 2004. [cited at p. 68]

[64] Stefano Ceri, Piero Fraternali, Angela Maurino, and Stefano Paraboschi. One-to-one personalization of data-intensive web sites. In *2nd International Workshop on The Web and Databases*, pages 1–6, 1999. [cited at p. 9]

[65] Chia-Hui Chang, Chun-Nan Hsu, and Lui Shao-Chen. Automatic information extraction from semi-structured web pages by pattern discovery. *Decision Support Systems*, 35(1):129–147, 2003. [cited at p. 44, 59, 257]

[66] Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled Shaalan. A survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, 2006. [cited at p. 5, 32, 45, 233, 237, 238, 239, 240]

[67] Chia-Hui Chang and Shih-Chien Kuo. OLERA: Semisupervised web-data extraction with visual support. *IEEE Intelligent Systems*, 19(6):56–64, 2004. [cited at p. 45, 269]

[68] Chia-Hui Chang, Lui Shao-Chen, and Yen-Chin Wu. Applying pattern mining to web information extraction. In *5th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 4–16, 2001. [cited at p. 257]

[69] Sudarshan Chawathe, Yannis Papakonstantinou, Jeffrey D. Ullman, Hector Garcia-Molina, Kelly Ireland, Joachim Hammer, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *10th Meeting of the Information Processing Society of Japan*, pages 7–18, 1994. [cited at p. 243]

[70] Li-Qun Chen, Xing Xie, Wei-Ying Ma, Hong-Jiang Zhang, Heqin Zhou, and Huanqing Feng. DRESS: A slicing tree based web page representation for various display sizes. In *12th International Conference on World Wide Web*, 2003. [cited at p. 268]

[71] Liangyou Chen, Nan Wang, and Hassan M. Jamil. Automatic composite wrapper generation for semi-structured biological data based on table structure identification. *SIGMOD Record*, 33(2):58–64, 2004. [cited at p. 268]

[72] Boris Chidlovskii, Uwe M. Borghoff, and Pierre-Yves Chevalier. Towards sophisticated wrapping of web-based information repositories. In *5th International RIAO Conference*, pages 123–135, 1997. [cited at p. 5, 31, 43, 237, 239, 240]

[73] Freddy Y. Y. Choi. *Content-based Text Navigation.* PhD thesis, University of Manchester, 2002. [cited at p. 26]

[74] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. Collaborative wrapping: A turbo framework for web data extraction. In *23rd International Conference on Data Engineering*, pages 1261–1262, 2007. [cited at p. 265]

[75] Shui-Lung Chuang, Kevin Chen-Chuan Chang, and ChengXiang Zhai. Context-aware wrapping: Synchronized data extraction. In *33rd International Conference on Very Large Data Bases*, pages 699–710, 2007. [cited at p. 265]

[76] James Clark. XSL transformations (XSLT) version 1.0, 1999. [cited at p. 23]

[77] William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *1998 ACM SIGMOD International Conference on Management of Data*, pages 201–212, 1998. [cited at p. 252]

[78] William W. Cohen. Some practical observations on integration of web information. In *2nd International Workshop on The Web and Databases*, pages 55–60, 1999. [cited at p. 234, 239, 241, 242]

[79] William W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems*, 18(3):288–321, 2000. [cited at p. 252]

[80] William W. Cohen, Matthew Hurst, and Lee S. Jensen. A flexible learning system for wrapping tables and lists in HTML documents. In *11th International Conference on World Wide Web*, pages 232–241, 2002. [cited at p. 259]

[81] Jared Cope, David Hawking, and Nick Craswell. Automated discovery of search interfaces on the web. In *14th Australasian Conference on Database Technologies*, pages 181–189, 2003. [cited at p. 32, 268]

[82] Valter Crescenzi and Giansalvatore Mecca. Grammars have exceptions. *Information Systems*, 23(8):539–565, 1998. [cited at p. 248, 249]

[83] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: automatic data extraction from data-intensive web sites. In *2002 ACM SIGMOD International Conference on Management of Data*, pages 624–624, 2002. [cited at p. 259]

[84] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Handling irregularities in RoadRunner. In *Adaptive Text Extraction and Mining*, pages 33–38, 2004. [cited at p. 259]

[85] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *40th Anniversary Meeting of the Association for Computational Linguistics*, 2002. [cited at p. 169]

[86] Theodore Dalamagas, Panagiotis Bouros, Theodore Galanis, Magdalini Eirinaki, and Timos Sellis. Mining user navigation patterns for personalizing topic directories. In *9th International Workshop on Web Information and Data Management*, pages 81–88, 2007. [cited at p. 9]

[87] danah m. boyd and Nicole B. Ellison. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1):art. 11, 2007. [cited at p. 12]

[88] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: Scalable online collaborative filtering. In *16th International Conference on World Wide Web*, pages 271–280, 2007. [cited at p. 10, 27]

[89] Thomas H. Davenport. Competing on analytics. *Harvard Business Review*, (1), 2006. [cited at p. 18, 19]

[90] Sandip Debnath, Prasenjit Mitra, and C. Lee Giles. Automatic extraction of informative blocks from webpages. In Hisham M. Haddad, Lorie M. Liebrock, Andrea Omicini, and Roger L. Wainwright, editors, *2005 ACM Symposium on Applied Computing*, pages 1722–1726, 2005. [cited at p. 25, 269]

[91] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Halevy, and Dan Suciu. A query language for XML. In *8th International Conference on World Wide Web*, pages 1155–1169, 1999. [cited at p. 268]

[92] Xiaowen Ding, Bing Liu, and Philip S. Yu. A holistic lexicon-based approach to opinion mining. In *International Conference on Web Search and Web Data Mining*, pages 231–240, 2008. [cited at p. 26]

[93] Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A scalable comparison-shopping agent for the world-wide web. In *1st International Conference on Autonomous Agents*, pages 39–48, 1997. [cited at p. 246, 247]

[94] Robert B. Doorenbos, Daniel S. Weld, and Oren Etzioni. A scalable comparison-shopping agent for the world-wide-web. Technical report, 1996. [cited at p. 246]

[95] Witold Droździński, Hans-Urlich Krieger, Jakub Piskorski, Urlich Schafer, and Feiyu Xu. Shallow processing with unification and typed feature structures — foundations and applications. *Künstliche Intelligenz*, 1:17–23, 2004. [cited at p. 169]

[96] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *7th International Conference on Information and Knowledge Management*, pages 148–155, 1998. [cited at p. 25]

[97] Line Eikvil. Information extraction from world wide web - a survey. Technical report, 1999. [cited at p. 5, 28, 30, 31, 44, 233, 237, 238, 239]

[98] Magdalini Eirinaki and Michalis Vazirgiannis. Web mining for web personalization. *ACM Transactions on Internet Technology*, 3(1):1–27, 2003. [cited at p. 27]

[99] Ali I. El-Desouky, Hesham A. Ali, and Sally M. El-Ghamrawy. A new framework for domain-specific hidden web crawling based on data extraction techniques. In *4th International Conference on Information and Communications Technologies*, 2006. [cited at p. 269]

[100] Ramez El-Masri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Pearson Education, 2004. [cited at p. 4]

[101] David W. Embley, Cui Tao, and Stephen W. Liddle. Automating the extraction of data from HTML tables with unknown structure. *Data & Knowledge Engineering*, 54(1):3–28, 2005. [cited at p. 269]

[102] Oren Etzioni. The world-wide web: quagmire or gold mine? *Communications of ACM*, 39(11):65–68, 1996. [cited at p. 25]

[103] Bettina Fazzinga, Sergio Flesca, Andrea Tagaralli, Salvatore Garruzzo, and Elio Masciari. A wrapper generation system for PDF documents. In *2008 ACM Symposium on Applied Computing*, pages 442–446, 2008. [cited at p. 32, 269]

[104] Mary Fernandez, Alon Halevy, Daniela Florescu, and Dan Suciu. Declarative specification of web sites with strudel. *The VLDB Journal*, 9(1):38–55, 2000. [cited at p. 268]

[105] Thorsten Fiebig, Jurgen Weiss, and Guido Moerkotte. RAW: A relational algebra for the web. In *Workshop on Management of Semistructured Data*, 1997. [cited at p. 268]

[106] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2(2):115–150, 2002. [cited at p. 24]

[107] Aykut Firat. *Information Integration Using Contextual Knowledge and Ontology Merging*. PhD thesis, Massachusetts Institute of Technology, 2003. [cited at p. 31, 45, 234, 236, 261]

[108] Aykut Firat, Stuart Madnick, and Michael Siegel. The cameleon web wrapper engine. In *Workshop on Technologies for E-Services*, pages 14–15, 2000. [cited at p. 261]

[109] Dominik Flejter. Automatyczny podział dokumentów w kolekcji na tematyczne fragmenty (automatic topical segmentation of documents in text collections). Master's thesis, Poznan University of Economics, 2006. [cited at p. 25]

[110] Dominik Flejter, Monika Charydczak, and Wojciech Rutkowski. *Badanie efektywności komunikacji elektronicznej w podejmowaniu decyzji*, pages 91–104. Wydaw. AE, 2006. [cited at p. 19]

[111] Dominik Flejter, Slawomir Grzonkowski, Tomasz Kaczmarek, Marek Kowalkiewicz, and Tadhg Nagle, editors. *BIS 2008 Workshop Proceedings, Innsbruck, Austria, 6-7 May 2008*. Poznan University of Economics, 2008. [cited at p. ix, 12]

[112] Dominik Flejter and Roman Hryniewiecki. Bottom-up discovery of clusters of maximal ranges in HTML trees for search engines results extraction. In W. Abramowicz, editor, *10th International Conference on Business Information Systems*, pages 398–410, 2007. [cited at p. 195, 265]

[113] Dominik Flejter and Tomasz Kaczmarek. *Wybrane aspekty integracji informacji z głębokiego Internetu*, pages 97–110. Wydaw. AE, 2007. [cited at p. 7, 195]

[114] Dominik Flejter, Tomasz Kaczmarek, and Witold Abramowicz. Architectures for deep web data extraction and integration. In *Conference on Information Systems Architecture and Technology*, pages 93–100, 2007. [cited at p. 7]

[115] Dominik Flejter, Tomasz Kaczmarek, and Marek Kowalkiewicz. World wide web on the move. *Scalable Computing: Practice and Experience*, 9(4):219–241, 2008. [cited at p. 23]

[116] Dominik Flejter, Tomasz Kaczmarek, and Marek Kowalkiewicz. Proceedings of MEM 2009 workshop at WWW 2009 (http://www.integror.net/mem2009/), 2009. [cited at p. ix]

[117] Dominik Flejter, Tomasz Kaczmarek, Marek Kowalkiewicz, and Michiaki Tatsubori. Proceedings of MEM 2010 workshop at WWW 2010 (http://www.integror.net/mem2010/), 2010. [cited at p. ix]

[118] Dominik Flejter and Marek Kowalkiewicz, editors. *SAW 2007 : Social Aspects of the Web*. CEUR-WS.ORG, 2007. [cited at p. 12]

[119] Dominik Flejter, Marek Kowalkiewicz, and Tomasz Kaczmarek. MEM&LCW 2008 workshop PC chairs' message. In *WISE '08 Proceedings of the 2008 international workshops on Web Information Systems Engineering*, pages 142–143, 2008. [cited at p. ix]

[120] Dominik Flejter, Karol Wieloch, and Witold Abramowicz. Unsupervised methods of topical text segmentation for polish. In *Workshop on Balto-Slavonic Natural Language Processing*, pages 51–58, 2007. [cited at p. 25]

[121] Daniela Florescu, Daphne Koller, and Alon Halevy. Using probabilistic information in data integration. In *23rd International Conference on Very Large Data Bases*, pages 216–225, 1997. [cited at p. 245]

[122] Juliana Freire and Luciano Barbosa. An adaptive crawler for locating hidden web entry points. In *16th International Conference on World Wide Web*, 2007. [cited at p. 269]

[123] Dayne Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(1):169–202, 2000. [cited at p. 268]

[124] Dayne Freitag and Nicholas Kushmerick. Boosted wrapper induction. In *17th National Conference on Artificial Intelligence*, pages 577–583, 2000. [cited at p. 268]

[125] Johannes Fürnkranz. Exploiting structural information for text classification on the WWW. In *3rd International Symposium on Advances in Intelligent Data Analysis*, pages 487–498, 1999. [cited at p. 25, 27]

[126] John M. Gallaugher and Suresh C. Ramanathan. *Online Exchanges and Beyond: Issues and Challenges in Crafting Successful B2B Marketplaces*, pages 51–70. Idea Group Publishing, 2002. [cited at p. 14, 16]

[127] Dashan Gao, Yizhou Wang, Haitham Hindi, and Minh Do. Decompose document image using integer linear programming. pages 397–401, 2007. [cited at p. 269]

[128] Wolfgang Gatterbauer, Paul Bohunsky, Marcus Herzog, Bernhard Krüpl, and Bernhard Pollak. Towards domain independent information extraction from web tables. In *16th International Conference on World Wide Web*, pages 71–80, 2007. [cited at p. 269]

[129] Allan J. Godbout. Filtering knowledge: Changing information into knowledge assets. *Journal of Systemic Knowledge Management*, 1:art. 11, 1999. [cited at p. 3]

[130] Roy Goldman and Jennifer Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *23rd International Conference on Very Large Data Bases*, pages 436–445, 1997. [cited at p. 244]

[131] Paulo Braz Golgher, Alberto H. F Laender, Altigran Soares da Silva, and Berthier A. Ribeiro-Neto. An example-based environment for wrapper generation. In *Workshop on Conceptual Modeling Approaches for E-Business and The World Wide Web*, pages 152–164, 2000. [cited at p. 254]

[132] Andrzej Gontarz. Przez analizę do decyzji, 2008. [cited at p. 19]

[133] Darrell D. E. Long Gottfried Vossen and Jeffrey Xu Yu, editors. *Web Information Systems Engineering - WISE 2009, 10th International Conference, Poznan, Poland, October 5-7, 2009. Proceedings.* Springer, 2009. [cited at p. ix]

[134] Georg Gottlob and Christoph Koch. Logic-based web information extraction. *SIGMOD Record*, 33(2):87–94, 2004. [cited at p. 255]

[135] Luis Gravano, Mehran Sahami, and Panagiotis G. Ipeirotis. Automatic classification of text databases through query probing. In *3rd International Workshop on The Web and Databases*, pages 245–255, 2001. [cited at p. 33]

[136] Ralph Grishman. Information extraction: Techniques and challenges. In *Information Extraction (International Summer School SCIE-97)*, pages 10–27, 1997. [cited at p. 28]

[137] Martin Gudgin, Mark Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon. SOAP version 1.2 part 1: Messaging framework (second edition) (W3C recommendation 27 april 2007), 2007. [cited at p. 24]

[138] Pankaj Gulhane, Rajeev Rastogi, Srinivasan H Sengamedu, and Ashwin Tengli. Exploiting content redundancy for web information extraction. In *19th International Conference on World Wide Web*, pages 1105–1106, 2010. [cited at p. 269]

[139] Richard D. Hackathorn. *Web farming for the data warehouse.* Morgan Kaufmann Publishers Inc., 2001. [cited at p. 3, 4]

[140] Alon Halevy. The information manifold approach to data integration. *IEEE Intelligent Systems*, 13:12–16, 1998. [cited at p. 245]

[141] Alon Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001. [cited at p. 32, 186]

[142] Alon Halevy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *13th National Conference on Artificial Intelligence*, pages 40–47, 1996. [cited at p. 244]

[143] Alon Halevy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *22nd International Conference on Very Large Data Bases*, pages 251–262, 1996. [cited at p. 235, 244]

[144] Joachim Hammer, Hector Garcia-Molina, Junghoo Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the web. In *1997 ACM SIGMOD International Conference on Management of Data: Workshop on Management of Semistructured Data*, 1997. [cited at p. 244]

[145] Joachim Hammer, Jason McHugh, and Hector Garcia-Molina. Semistructured data: The TSIMMIS experience. In *1st East-European Symposium on Advances in Databases and Information Systems*, pages 1–8, 1997. [cited at p. 243]

[146] John R. Hauser, Glen L. Urban, Guilherme Liberali, and Michael Braun. Website morphing. Technical report, 2008. [cited at p. 10]

[147] Taher H. Haveliwala, Sepandar D. Kamvar, and Glen Jeh. An analytical comparison of approaches to personalizing PageRank. Technical report, 2003. [cited at p. 26]

[148] Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across web query interfaces. In *2003 ACM SIGMOD International Conference on Management of Data*, 2003. [cited at p. 7]

[149] Bin He and Kevin Chen-Chuan Chang. A holistic paradigm for large scale schema matching. *SIGMOD Record*, 33(4):20–25, 2004. [cited at p. 32]

[150] Bin He, Zhen Zhang, and Kevin Chen-Chuan Chang. MetaQuerier: querying structured web sources on-the-fly. In *2005 ACM SIGMOD International Conference on Management of Data*, pages 927–929, 2005. [cited at p. 33]

[151] Bin He, Zhen Zhang, and Kevin Chen-Chuan Chang. Toward large scale integration: Building a MetaQuerier over databases on the web. In *2nd Conference on Innovative Data Systems Research*, 2005. [cited at p. 269]

[152] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004. [cited at p. xii, xiii, 2, 23, 43, 67, 143, 159, 195]

[153] Rainer Himmeröder, Georg Lausen, Bertram LudÃd'scher, and Christian Schlepphorst. On a declarative semantics for web queries. In *5th International Conference on Deductive and Object-Oriented Databases*, pages 386–398, 1997. [cited at p. 249]

[154] Andrew Hogue and David R. Karger. Thresher: Automating the unwrapping of semantic content from the world wide web. In *14th International Conference on World Wide Web*, pages 86–95, 2005. [cited at p. 264]

[155] Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems*, 23(9):521–538, 1998. [cited at p. 31, 44, 233, 238, 239, 251]

[156] Gerald Huck, Peter Fankhauser, Karl Aberer, and Erich Neuhold. Jedi: Extracting and synthesizing information from the web. In *3rd IFCIS International Conference on Cooperative Information Systems*, pages 32–43, 1998. [cited at p. 250]

[157] Panagiotis G. Ipeirotis. *Classifying and Searching Hidden-Web Text Databases.* PhD thesis, Columbia University, 2004. [cited at p. 268]

[158] Utku Irmak and Torsten Suel. Interactive wrapper generation with minimal user effort. In *15th International Conference on World Wide Web*, 2006. [cited at p. 269]

[159] Alex Iskold. Top-down: A new approach to the semantic web, 2007. [cited at p. 23]

[160] Tawfik Jelassi and Albrecht Enders. *Strategies for e-business: Creating Value through Electronic and Mobile Commerce.* Pearson Education Limited, 2004. [cited at p. 16]

[161] Jane Yung jen Hsu and Wen tau Yih. Template-based information mining from HTML documents. In *14th National Conference on Artificial Intelligence*, pages 256–262, 1997. [cited at p. 268]

[162] Jose Joemon, Jana Urban, Ren Reede, Krishna Chandramouli, Divna Djordjevic, Pavel Praks, and Roland Mörzinger. KSpace project ID4.4.1: State of the art report on multimedia mining. Technical report, 2006. [cited at p. 26]

[163] Quentin Jones, Gilad Ravid, and Sheizaf Rafeali. Information overload and the message dynamics of online interaction spaces: A theoretical model and empirical exploration. *Information Systems Research*, 15(2):194–210, 2004. [cited at p. 18, 19]

[164] Govind Kabra, Zhen Zhang, and Kevin Chen-Chuan Chang. Dewex: An exploration facility for enabling the deep web integration. In *23rd International Conference on Data Engineering*, pages 1511–1512, 2007. [cited at p. 269]

[165] Tomasz Kaczmarek. *Deep Web data integration for company environment analysis (in Polish).* PhD thesis, Poznan University of Economics, 2006. [cited at p. 264]

[166] Ravi Kalakota and Andrew Whinston. *Frontiers of electronic commerce.* Addison-Wesley, 1996. [cited at p. 14]

[167] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Extrapolation methods for accelerating PageRank computations. In *12th International Conference on World Wide Web*, pages 261–270, 2003. [cited at p. 26]

[168] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999. [cited at p. 26]

[169] Craig A. Knoblock, Steven Minton, Jose Luis Ambite, Naveen Ashish, Ion Muslea, Andrew G. Philpot, and Sheila Tejada. The ariadne approach to web-based information integration. *International Journal of Cooperative Information Systems*, 10(1):145–169, 2001. [cited at p. 250, 268]

[170] Craig A. Knoblock, Ion Muslea, Kristina Lerman, and Steven Minton. *Accurately and reliably extracting data from the web: A machine learning approach*, pages 275–287. 2003. [cited at p. 250]

[171] Pranam Kolari and Anupam Joshi. Web mining: Research and practice. *Computing in Science and Engineering*, 6(4):49–53, 2004. [cited at p. 25, 26, 27]

[172] David Konopnicki and Oded Shmueli. W3QS: A query system for the world-wide web. In *21st International Conference on Very Large Data Bases*, 1995. [cited at p. 244]

[173] David Konopnicki and Oded Shmueli. Information gathering in the world wide web: The W3QL query language and the W3QS system. *ACM Transactions on Database Systems*, 23(4):369–410, 1998. [cited at p. 244]

[174] Raymond Kosala and Hendrik Blockeel. Web mining research: a survey. *ACM SIGKDD Explorations Newsletter*, 2(1):1–15, 2000. [cited at p. 25, 27]

[175] Marek Kowalkiewicz. *Ekstrakcja i agregacja informacji dla potrzeb podmiotów gospodarczych (Information extraction and aggregation for economic entities).* PhD thesis, Poznan University of Economics, 2006. [cited at p. 46, 79]

[176] Stefan Kulins and Ross Tredwell. Toolkits for generating wrappers. In *International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 184–198, 2002. [cited at p. 44]

[177] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. In *8th International Conference on World Wide Web*, pages 1481–1493, 1999. [cited at p. 26]

[178] Nicholas Kushmerick. *Wrapper induction for information extraction.* PhD thesis, University of Washington, 1997. [cited at p. 247]

[179] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 2000. [cited at p. 247]

[180] Nicholas Kushmerick and Bernd Thomas. *Adaptive Information Extraction: Core technologies for Information agents*, pages 79–103. Springer, 2003. [cited at p. 31, 45, 233, 238, 239]

[181] Alberto H. F Laender, Berthier A. Ribeiro-Neto, and Altigran Soares da Silva. DEByE - data extraction by example. *Data & Knowledge Engineering*, 40(2):121–154, 2002. [cited at p. 59, 253]

[182] Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. A declarative language for querying and restructuring the web. In *6th International Workshop on Research Issues in Data Engineering*, pages 12–21, 1996. [cited at p. 245]

[183] Georg Lausen and Kai Simon. ViPER: augmenting automatic information extraction with visual perceptions. *14th International Conference on Information and Knowledge Management*, 2005. [cited at p. 33]

[184] Ryan Levering and Michal Cutler. The portrait of a common HTML web page. In *2006 ACM symposium on Document engineering*, pages 198–204, 2006. [cited at p. 6, 23]

[185] Dirk Lewandowski and Philipp Mayr. Exploring the academic invisible web. *Library Hi Tech*, 24(4):529–539, 2006. [cited at p. 7]

[186] Longzhuang Li, Yi Shang, and Zhang Wei. Improvement of HITS-based algorithms on web documents. In *11th International Conference on World Wide Web*, pages 527–535, 2002. [cited at p. 26]

[187] King-Ip Lin and Hui Chen. Automatic information discovery from the 'invisible web'. In *2002 International Conference on Information Technology: Coding and Computing*, page 332, 2002. [cited at p. 7, 32, 268]

[188] Ling Lin and Lizhu Zhou. Leveraging webpage classification for data object recognition. In *IEEE/WIC/ACM International Conference on Web Intelligence*, pages 667–670, 2007. [cited at p. 269]

[189] Bing Liu. Web content mining. tutorial, 2005. [cited at p. 45]

[190] Bing Liu, Robert Grossman, and Yanhong Zhai. Mining data records in web pages. In *9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601–606, 2003. [cited at p. 238, 262]

[191] King-Lup Liu, Clement Yu, and Weiyi Meng. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002. [cited at p. 33]

[192] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-enabled wrapper construction system for web information sources. In *16th International Conference on Data Engineering*, pages 611–621, 2000. [cited at p. 254]

[193] Ling Liu, Calton Pu, Tang Wei, and Wei Han. CONQUER: A continual query system for update monitoring in the WWW. *International Journal of Computer Systems Science and Engineering*, 14(2):99–112, 1999. [cited at p. 268]

[194] Ling Liu, Wei Tang, David Buttler, and Calton Pu. Information monitoring on the web: A scalable solution. *World Wide Web*, 5(4):263–304, 2002. [cited at p. 254]

[195] Miao Liu. NetQL: A structured web query language. Master's thesis, University of Regina, 1998. [cited at p. 268]

[196] Yuting Liu, Bin Gao, Tie-Yan Liu, Ying Zhang, Zhiming Ma, Shuyuan He, and Hang Li. BrowseRank: Letting web users vote for page importance. In *31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 451–458, 2008. [cited at p. 27]

[197] Frederick H. Lochovsky and Jiying Wang. Data extraction and label assignment for web databases. In *12th International Conference on World Wide Web*, pages 187–196, 2003. [cited at p. 261]

[198] Bertram LudÃd'scher, Rainer Himmeröder, Georg Lausen, Wolfgang May, and Christian Schlepphorst. Managing semistructured data with FLORID: a deductive object-oriented perspective. *Information Systems*, 23(9):589–613, 1998. [cited at p. 249]

[199] Sanjay K. Madria, Sourav S. Bhowmick, Wee Keong Ng, and Ee-Peng Lim. Research issues in web data mining. In *1st International Conference on Data Warehousing and Knowledge Discovery*, pages 303–312, 1999. [cited at p. 25]

[200] Udi Manber, Ash Patel, and John Robinson. Experience with personalization of yahoo! *Communications of ACM*, 43(8):35–39, 2000. [cited at p. 9]

[201] Jennifer Marill, Andrew Boyko, and Michael Ashenfelder. Web harvesting survey (netpreserve.org), 2004. [cited at p. 32, 234, 235, 236, 237]

[202] Andrew McCallum and William W. Cohen. Information extraction from the world wide web. tutorial, 2002. [cited at p. 28, 45]

[203] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview (W3C recommendation 10 february 2004), 2004. [cited at p. 24]

[204] Giansalvatore Mecca, Paolo Atzeni, Alessandro Masci, Paolo Merialdo, and Giuseppe Sindoni. The ARANEUS web-base management system. In *1998 ACM SIGMOD International Conference on Management of Data*, pages 544–546, 1998. [cited at p. 248]

[205] Giansalvatore Mecca and Valter Crescenzi. Automatic information extraction from large websites. *Journal of the ACM*, 51(5):731–779, 2004. [cited at p. 259]

[206] Giansalvatore Mecca, Paolo Merialdo, and Paolo Atzeni. Araneus in the era of XML. *IEEE Data Engineering Bulletin*, 22(3):19–26, 1999. [cited at p. 249]

[207] Alberto O. Mendelzon, George A. Mihaila, and Tova Milo. Querying the world wide web. *International Journal on Digital Libraries*, 1(1):54–67, 1997. [cited at p. 245]

[208] Paolo Merialdo, Giansalvatore Mecca, and Valter Crescenzi. RoadRunner: Towards automatic data extraction from large web sites. In *27th International Conference on Very Large Data Bases*, pages 109–118, 2001. [cited at p. 259]

[209] Alistair Miles and Sean Bechhofer. SKOS simple knowledge organization system reference (W3C working draft 9 june 2008), 2008. [cited at p. 24]

[210] Steven Minton, Craig A. Knoblock, and Kristina Lerman. Automatic data extraction from lists and tables in web sources. In *Proceedings of the IJCAI 2001 Workshop on Adaptive Text Extraction and Mining*, 2001. [cited at p. 268]

[211] Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. Improving the effectiveness of collaborative filtering on anonymous web usage data. Technical report, 2001. [cited at p. 10, 27]

[212] Pragnesh Jay Modi, Craig A. Knoblock, Jose Luis Ambite, Naveen Ashish, Steven Minton, Ion Muslea, Sheila Tejada, and Andrew G. Philpot. Modeling web sources for information integration. 1998. [cited at p. 250]

[213] Ion Muslea. Extraction patterns for information extraction tasks: A survey. In *The AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999. [cited at p. 28, 29, 44]

[214] Ion Muslea, Steven Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In *3rd International Conference on Autonomous Agents*, pages 190–197, 1999. [cited at p. 250]

[215] Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Hierarchical Wrapper Induction for Semistructured Information Sources*, 4(1):93–114, 2001. [cited at p. 250]

[216] Kamal Nigam, John Lafferty, and Andrew McCallum. Using maximum entropy for text classification. In *Workshop on Machine Learning for Information Filtering*, pages 61–67, 1999. [cited at p. 25]

[217] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, 1998. [cited at p. 26]

[218] Alberto Pan, Paula Montoto, Anastasio Molano, Manuel Alvarez, Juan Raposo, and Angel Vina. A model for advanced query capability description in mediator systems. In *4th International Conference on Enterprise Information Systems*, pages 140–147, 2002. [cited at p. 260]

[219] Alberto Pan, Juan Raposo, Manuel Alvarez, Paula Montoto, Vincente Orjales, Justo Hidalgo, Lucia Ardao, Anastasio Molano, and Angel Vina. The denodo data integration platform. In *28th International Conference on Very Large Data Bases*, pages 986–989, 2002. [cited at p. 31, 234, 236, 239, 240, 260]

[220] Sandeep Pandey, Kedar Dhamdhere, and Christopher Olston. WIC: A general-purpose algorithm for monitoring web information sources. In *30th International Conference on Very Large Data Bases*, pages 360–371, 2004. [cited at p. 269]

[221] Yannis Papakonstantinou, Ashish Gupta, and Laura Haas. Capabilities-based query rewriting in mediator systems. In *4th International Conference on Parallel and Distributed Information Systems*, 1996. [cited at p. 32]

[222] Michaeal P. Papazoglou and Pieter M. Ribbers. *E-Business. Organizational and Technical Foundations.* John Wiley & Sons, 2006. [cited at p. 16]

[223] Justin Park and Denilson Barbosa. Adaptive record extraction from web pages. In *16th International Conference on World Wide Web*, 2007. [cited at p. 269]

[224] Mike Perkowitz and Oren Etzioni. Towards adaptive web sites: Conceptual framework and case study. *Artificial Intelligence*, 118:245–275, 2000. [cited at p. 9]

[225] Andrew G. Philpot, Craig A. Knoblock, Jose Luis Ambite, and Ion Muslea. Compiling source descriptions for efficient and flexible information integration. *Journal of Intelligent Information Systems*, 16:149–187, 2001. [cited at p. 32]

[226] Dimitrios Pierrakos, Georgios Paliouras, Christos Papatheodorou, and Constantine D. Spyropoulos. Web usage mining as a tool for personalization: A survey. *User Modeling and User-Adapted Interaction*, 6(2):311–372, 2003. [cited at p. 27]

[227] Michael E. Porter and Victor E. Millar. How information gives you competitive advantage. *Harvard Business Review*, 63(4):149–150, 1985. [cited at p. 17]

[228] S.T.S. Prasad and Anand Rajaraman. Virtual database technology, XML, and the evolution of the web. *IEEE Data Engineering Bulletin*, 21(2):48–52, 1998. [cited at p. 251]

[229] Calton Pu, Karsten Schwan, and Jonathan Walpole. Infosphere project: System support for information flow applications. *SIGMOD Record*, 30(1):25–34, 2001. [cited at p. 254]

[230] Dave Ragget, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 specification, 1999. [cited at p. 6, 7]

[231] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. Technical report, 2000. [cited at p. 256]

[232] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *27th International Conference on Very Large Data Bases*, pages 129–138, 2001. [cited at p. 7, 256]

[233] Vijay Raghavan, Zonghuan Wu, Hongkun Zhao, and Clement Yu. Fully automatic wrapper generation for search engines. In *14th International Conference on World Wide Web*, pages 66–75, 2005. [cited at p. 33, 263]

[234] Juan Raposo, Alberto Pan, Manuel Alvarez, and Justo Hidalgo. Automatically maintaining wrappers for semi-structured web sources. *Data & Knowledge Engineering*, 61:331–358, 2007. [cited at p. 260]

[235] Berthier A. Ribeiro-Neto, Alberto H. F Laender, and Altigran Soares da Silva. Extracting semi-structured data through examples. In *8th International Conference on Information and Knowledge Management*, pages 94–101, 1999. [cited at p. 253]

[236] Jerome Robinson. Data extraction from web data sources. In *15th International Workshop on Database and Expert Systems Applications (2)*, pages 282–288, 2004. [cited at p. 269]

[237] J. Edward Russo and Paul J. H. Schoemaker. *Winning Decisions: Getting it Right the First Time*. The Doubleday Publishing Group, 2002. [cited at p. 18]

[238] John Sacher, editor. *Electronic Commerce: Opportunities and Challenges for Government*. OECD, 1997. [cited at p. 14]

[239] Arnaud Sahuguet and Fabien Azavant. Web ecology recycling HTML pages as XML documents using W4F. In *2nd International Workshop on The Web and Databases*, pages 31–36, 1999. [cited at p. 252]

[240] Arnaud Sahuguet and Fabien Azavant. WysiWyg web wrapper factory, 1999. [cited at p. 252]

[241] Hiroo Saito, Masashi Toyoda, Masaru Kitsuregawa, and Kazuyuki Aihara. A large-scale study of link spam detection by graph algorithms. In *3rd International Workshop on Adversarial Information Retrieval on the Web*, pages 45–48, 2007. [cited at p. 27]

[242] Sunita Sarawagi. Automation in information extraction and integration. tutorial., 2002. [cited at p. 44]

[243] Pierre Senellart, Avin Mittal, Daniel Muschnick, Rémi Gilleron, and Marc Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *10th International Workshop on Web Information and Data Management*, pages 9–16, 2008. [cited at p. 266]

[244] Lui Shao-Chen and Chia-Hui Chang. IEPAD: Information extraction based on pattern discovery. In *10th International Conference on World Wide Web*, 2001. [cited at p. 257]

[245] Nikhil Sharma. The origin of the data information knowledge wisdom hierarchy, 2008. [cited at p. 2]

[246] Peter Sharp. MaKE first steps: a collaborative approach to defining knowledge in organisations. *The Electronic Journal of Knowledge Management*, 4(2):189–196, 2006. [cited at p. 2]

[247] Cynthia C. Shelly and George Young. Accessibility for simple to moderate-complexity DHTML web sites. In *2007 international cross-disciplinary conference on Web accessibility*, pages 65–73, 2007. [cited at p. 11]

[248] Denis Shestakov. *Search Interfaces on the Web: Querying and Characterizing*. PhD thesis, University of Turku, 2008. [cited at p. 267]

[249] Denis Shestakov, Sourav S. Bhowmick, and Ee-Peng Lim. DEQUE: querying the deep web. *Data & Knowledge Engineering*, 52(1):273–311, 2005. [cited at p. 267]

[250] Lawrence Kai Shih and David R. Karger. Using URLs and table layout for web classification tasks. In Stuart Feldman, Mike Uretsky, Marc Najork, and Craig Wills, editors, *13th International Conference on World Wide Web*, pages 193–202, 2004. [cited at p. 25]

[251] Luo Si and Jamie Callan. A semisupervised learning method to merge search engine results. *ACM Transactions on Information Systems*, 21(4):457–491, 2003. [cited at p. 268]

[252] Keith Smith. Simplifying ajax-style web development. *IEEE Computer*, 39(5):98–101, 2006. [cited at p. 11, 12]

[253] Stephen Soderland. Learning to extract text-based information from the world wide web. *3rd International Conference on Knowledge Discovery and Data Mining*, pages 251–254, 1997. [cited at p. 246]

[254] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1):233–272, 1999. [cited at p. 5, 268]

[255] Stephen Soderland, David Fisher, Jonathan Aseltine, and Wendy Lehnert. CRYSTAL: Inducing a conceptual dictionary. In *14th International Joint Conference on Artificial Intelligence*, pages 1314–1319, 1995. [cited at p. 246]

[256] Mikhail Sogrine and Ahmed Patel. Evaluating database selection algorithms for distributed search. In *2003 ACM Symposium on Applied Computing*, pages 817–822, 2003. [cited at p. 33]

[257] Myra Spiliopoulou. Web usage mining for web site evaluation. *Communications of ACM*, 43(8):127–134, 2000. [cited at p. 27]

[258] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter*, 1(2):12–23, 2000. [cited at p. 27]

[259] Michael Steinbach, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining 2000*, pages 1–2, 2000. [cited at p. 25]

[260] Tsuyoshi Sugibuchi and Yuzuru Tanaka. Interactive web-wrapper construction for extracting relational information from web documents. In *14th International Conference on World Wide Web (Posters)*, pages 968–969, 2005. [cited at p. 269]

[261] Marcin Sydow. Random surfer with back step. In Stuart Feldman, Mike Uretsky, Marc Najork, and Craig Wills, editors, *13th International Conference on World Wide Web*, pages 352–353, 2004. [cited at p. 26]

[262] Juliana S. Teixeira, Berthier A. Ribeiro-Neto, Alberto H. F Laender, and Altigran Soares da Silva. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002. [cited at p. 30, 44]

[263] Paul Timmers and Jorge Gasos. *Agent Technologies and Business Models for Electronic Commerce*, pages 189–206. Idea Group Publishing, 2002. [cited at p. 17]

[264] Takashi Tomokiyo and Matthew Hurst. A language model approach to keyphrase extraction. In *2003 Workshop on Multiword expressions*, pages 33–40, 2003. [cited at p. 25]

[265] Jordi Turmo, Alicia Ageno, and Neus CatalÃă. Adaptive information extraction. *ACM Computing Surveys*, 38(2):Art. 4, 2006. [cited at p. 28]

[266] Jeffrey D. Ullman. Information integration using logical views. In *6th International Conference on Database Theory*, pages 19–40, 1997. [cited at p. 245]

[267] Jeffrey D. Ullman, Ramana Yerneni, Vasilis Vassalos, Hector Garcia-Molina, Murty Valiveti, Chen Li, and Yannis Papakonstantinou. Capability based mediation in TSIMMIS. In *1998 ACM SIGMOD International Conference on Management of Data*, pages 564–566, 1998. [cited at p. 244]

[268] Sreelakshmi Vaddi, Zaiqing Nie, Ullas Nambiar, and Subbarao Kambhampati. Mining coverage statistics for websource selection in a mediator. In *11th International Conference on Information and Knowledge Management*, pages 678–680, 2002. [cited at p. 33]

[269] Iwan Vosloo and Derrick G. Kourie. Server-centric web frameworks: An overview. *ACM Computing Surveys*, 40(2):4:1–4:33, 2008. [cited at p. 11]

[270] Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *11th International Conference on World Wide Web*, pages 242–250, 2002. [cited at p. 6]

[271] Yang Wang and Thomas Hornung. Deep web navigation by example. In *1st Workshop on Advances in Accessing Deep Web*, pages 131–140, 2008. [cited at p. 8, 269]

[272] J. Christopher Westland and Theodore H. K. Clark. *Global Electronic Commerce: Theory and Case Studies.* The MIT Press, 1999. [cited at p. 14, 16]

[273] Jennifer Widom, Jeffrey D. Ullman, Anand Rajaraman, Dallan Quass, and Yehoshua Sagiv. Querying semistructured heterogeneous information. In *4th International Conference on Deductive and Object-Oriented Databases*, pages 319–344, 1995. [cited at p. 243]

[274] Dave Winer. XML-RPC specification (updated 6/30/03), 2003. [cited at p. 24]

[275] Marek Wiśniewski. *Ontology Learning from Text (Uczenie ontologii z tekstu).* PhD thesis, Poznan University of Economics, 2008. [cited at p. 26]

[276] Dawid Węckowski. Automatyzacja działań użytkownika w przeglądarkach internetowych (automating user activities in web browsers). Master's thesis, Poznan University of Economics, 2009. [cited at p. 143]

[277] Harris Wu, Mohammed Zubair, and Kurt Maly. Harvesting social knowledge from folksonomies. In Uffe K. Wiil, , Peter J. Nürnberg, and Jessica Rubart, editors, *17th conference on Hypertext and Hypermedia*, pages 111–114, 2006. [cited at p. 26]

[278] Li Xu, Yihong Ding, and David W. Embley. Automatic direct and indirect schema mapping: experiences and lessons learned. *SIGMOD Record*, 33(4):14–19, 2004. [cited at p. 32]

[279] Jaeyoung Yang, Joongmin Choi, Kyeongho Lee, Hosang Ham, and Joongbae Kim. A more scalable comparison shopping agent. In *Engineering of Intelligent Systems*, pages 766–772, 2000. [cited at p. 268]

[280] Xiaochun Yang, Bin Wang, Guoren Wang, and Ge Yu. A query rewriting system for enhancing the queriability of form-based interface. In *7th International Conference on Asian Digital Libraries*, page 462, 2004. [cited at p. 9, 269]

[281] Soon yong Choi, Andrew Whinston, and Dale Stahl. *Economics of Electronic Commerce.* Macmillan Computer Publishing, 1997. [cited at p. 16, 19]

[282] Clement Yu, Hai He, Weiyi Meng, and Zonghuan Wu. Automatic integration of web search interfaces with WISE-integrator. *The VLDB Journal*, 13(3):256–273, 2004. [cited at p. 269]

[283] Yanhong Zhai and Bing Liu. Extracting web data using instance-based learning. In *6th International Conference on Web Information Systems Engineering*, 2005. [cited at p. 239, 263]

[284] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *14th International Conference on World Wide Web*, pages 76–85, 2005. [cited at p. 33, 262]

[285] Bo Zhang, Ji-Rong Wen, Wei-Ying Ma, Zaiqing Nie, and Jun Zhu. 2d conditional random fields for web information extraction. In *22nd International Conference on Machine Learning*, pages 1044–1051, 2005. [cited at p. 269]

[286] Wei Zhang, Clement Yu, and Weiyi Meng. Opinion retrieval from blogs. In *16th International Conference on Information and Knowledge Management*, pages 831–840, 2007. [cited at p. 26]

[287] Zhen Zhang. Large-scale deep web integration: Exploring and querying structured data on the deep web, 2006. [cited at p. 1]

[288] Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Light-weight domain-based form assistant: querying web databases on the fly. In *31st International Conference on Very Large Data Bases*, pages 97–108, 2005. [cited at p. 9]

[289] Hongkun Zhao, Weiyi Meng, and Clement Yu. Automatic extraction of dynamic record sections from search engine result pages. In *32nd International Conference on Very Large Data Bases*, 2006. [cited at p. 263]

[290] Hongkun Zhao, Weiyi Meng, and Clement Yu. Mining templates from search result records of search engines. In *13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 884–893, 2007. [cited at p. 263]

[291] Tong Zhou, Bing Liu, and Calton Pu. TAM: a system for dynamic transactional activity management. In *1999 ACM SIGMOD International Conference on Management of Data*, pages 571–573, 1999. [cited at p. 254]

Appendices

Appendix A

User — Web Site Interaction

In the simplest interaction of a user with a Web site, there are at least four parties involved: Web browser (or any other application capable of interactions with Web servers), the user (or any agent that initiates specific actions in Web browser), Web server HTTP service, and Web server's file system that stores files returned to the user via the HTTP service and his/her Web browser. However, in the majority of Web sites, some additional components need to be considered: the Web site's data storage, server-side session storage, user profile, server-side and client-side Web application logic, and client-side information storage.

In Figure 2.1 we presented a schematic view of the data and control flows between these components during single user interaction with a Web site. In Table A.1 we characterize all involved components, and in the remainder of this appendix, we present a more detailed step-by-step analysis of the flows depicted in Figure 2.1.

Component	Role
User or Agent	A party that initiates (a series of) actions in Web browser (*Web actions*); it may be an user, a programmatic agent (e.g. data extraction wrapper, Web monitoring tool, search engine bot) or a Web page event (e.g. JavaScript refreshing data every minute).
Web browser	A component handling different Internet protocols (at least HTTP GET and POST methods), rendering different types of Web content for the end user (at least (X)HTML+CSS, XML+XSLT, Flash, graphical files), making them available via the Document Object Model, running client-side JavaScript, and user-issued events (such as clicks or form submission).
Client-side application logic	Scripts that are part of a Web site loaded into a Web browser; they interact with a user (agent), Web browser (displayed document), Web server HTTP service and client-side information storage, and enable the dynamic behavior of Web page content and advanced forms of responsiveness to a user's actions in the Web browser.
Client-side information storage	A component that stores information (*client-side navigation state*) at the Web browser between consecutive interactions with a user or Web server; implemented by using part of some document that is not replaced with consecutive HTTP requests (e.g. the HTML frame or the body of an AJAX Web page), parameters of HTTP requests (e.g. GET parameters in links and hidden fields in HTML forms), as well as HTTP or Flash Cookies.

Component	Role
Web server HTTP service	A component that returns browser-interpretable content (HTML, JavaScript, XML, graphical files, Flash files, any format accepted by client-side application logic) and HTTP headers (e.g. HTTP status codes, type and size of content, expiry date) in response to HTTP requests.
Server-side application logic	The program controlling behavior of a Web site at the HTTP server; interprets HTTP requests, interacts with other server-side components (Web site's database, session storage, user profile and server-side static files), and generates content for browser or client-side application logic.
Web site database	A component that stores the data needed by server-side Web application logic and provides low-latency responses to its queries.
Session storage	A component that stores information (*server-side navigation state*) shared by consecutive HTTP requests handled by server-side Web application logic; implementations depend on used server-side technologies.
User profile	The long-term storage of information on user interests, expertise, personal data, behavior patterns and relations with other users.
Static files	Files that exist at the server and may be served directly by HTTP service or used internally by server-side Web application logic.

Table A.1: Key Components of User Interaction with Web Site

Action Planning (1)

In normal scenarios, some user action in a browser is initiated as part of following a specific goal and usage scenario. The first step in performing any navigation corresponds to deciding that a given action needs to be performed in a specific Web site and why. From an information extraction perspective it covers all challenges that are external to actual request issuing, such as selection of the best sources for a specific user query, defining a source-specific query and building a query execution plan.

User Action (2)

Each navigation cycle starts with an action performed in a Web browser by a user or some programmatic agent. A user may interact with a Web browser's features (e.g. manually entering the URL, using favorites or the back button), or with Web site's user interface (e.g. filling in and submitting forms, clicking links, using some more complex Web controls). Finally, the action may be initiated via a programmed event (e.g. the reload of a Web page with a META Refresh field or JavaScript timer, or a request issued in an `onload` event handler of a document).

From an information extraction perspective this step is challenging because it requires understanding the user interface, understanding which actions are really required to obtain data, and the ability to translate information need (query) into specific actions in a Web application interface. More specifically, the challenges are:

- identification of objects that are actionable, understanding their impact on client-side and server-side Web application, and selection of actions to be performed,

- understanding of implicit and explicit user interface limitations (e.g. combinations of attributes that may be queried in a form),
- ability to work with different types of user interface controls (including e.g. open-domain fields, CAPTCHAs, standard Web interface components, such as form elements and links, non-standard Web interface components, such as e.g. dHTML and Flash components),
- identification of automatically-issued Web actions that are actually useful for navigation (because they provide some data or change server-side navigation state or set some Cookies) and ignoring other automatically-issued Web actions.

Handling user action (3a, 3b)

If it is a simple navigational action handled by Web browser mechanisms (link click, form submission) based on single a HTTP request, this request is sent directly to the server (3a).

In more complex situations user action is an event passed to the client-side Web application that interprets it with its business logic and decides on further actions (3b). Client-side Web application may initiate any combination of actions 4-6 discussed below, or may decide to ignore user action and perform no activity at all.

Interaction with client-side information storage (4, 5)

To decide on further actions or its parameters, client-side application logic may need to acquire some information from client-side information storage (4). E.g. the application logic may check if the user has previously accepted some Web site internal rules (e.g. privacy policy, copyright related issues) to decide if some operation can be performed or if the user should have been asked for rules acceptance before.

Depending on actual user/agent action, the client-side Web application may store some information on user action in client-side information storage for further reference (5). For example, in a complex grid of data, each setting of data filtering and ordering may be saved in client-side information storage to make it possible to return to previous settings.

Usage of client-side information storage makes information extraction more challenging, as a Web site's behavior (and data extracted) depends on previous activities. Therefore, all interactions with information storage (that may have a complex programmatic form) need to be reconstructed during data extraction. Moreover, good understanding of locally stored information is needed - sometimes to get data it may be necessary to perform a sequence of actions, and sometimes it may be required to avoid executing some actions or to clean client-side information storage at a specific moment.

Request construction (6)

If at a given moment of time some interaction with the HTTP server (retrieving or saving some information) is required by client-side Web application logic, an HTTP request is constructed and sent to the server (6). In modern Web browsers, client-side Web application may issue both GET and POST requests (via techniques such as changing the location property of a document or frame, including extra script or CSS components to a page, and special objects such as XmlHttpRequest). Moreover, there are a few mechanisms (frames, XmlHttpRequest, Flash controls) that allow client-side Web application logic to issue requests asynchronously, making it possible to issue multiple requests in parallel.

At this point the execution of a user action moves to the Web server.

There are three main challenges for information extraction related to emulating this step of user interaction. The first challenge, which is purely technical and not that hard to handle, is the need to deal with different types of HTTP methods (GET, POST, PUT).

The second challenge is the ability to construct HTTP requests. In its basic form, it requires emulating client-side application logic rules (that may be complex). However, in many cases the HTTP server is to respond also to some request that cannot be produced by client-side application logic (e.g. queries for other combinations of attributes or operators). In such cases, the challenge is to know the real limitations of server-side Web application logic with respect to handled HTTP requests, and to benefit from its complete flexibility. It is important to note that sending requests that are not construcable in a client-side application's GUI may be risky and lead to obtaining error messages or even be treated as an intrusion, causing black-listing of a specific IP address.

The third challenge is related to the process of request issuing in asynchronous scenarios. As modern browser and Web applications often send multiple requests in parallel, time alignment, and consequently the application behavior over time, may be partially unpredictable. As a result, both client-side Web application logic and information extraction tools need to handle different orders of HTTP responses.

HTTP response generation

If, according to HTTP service configuration, a received HTTP request corresponds to a static file, the HTTP service prepares its content to be returned to the Web browser (7a, 10a), as well as response HTTP headers such as document's MIME type (that allows the Web browser to decide what to do with the received content), the size of the sent content ("Content-Size" header), its modification date ("Last-modified" header), validity period ("Expires" header) and expected Web browser's behavior ("Pragma", "Content-Disposition"). If a user is trying to access a non-existent static file, the server will prepare headers corresponding to the well-known "404 Not Found" status code.

If, according to HTTP service configuration, a response to a received HTTP request should be generated dynamically, the control is passed to server-side Web application logic (7b) and a response is returned (10b) only after actual content is generated (in steps 8-9, as described below).

From an information extraction perspective, there is one key challenge related to this step, concerning the fact that even if we repeat a complete sequence of requests in a Web site, we may receive different responses. It may happen, for example, whenever a server-side database, static files or session storage were modified by other parties (e.g. Web site authors, other users' actions or some regular Web site's server-side logic behavior), whenever the server-side Web application logic is changed, or at any moment in the case of adaptive or personalized Web sites.

As a result it may happen that responses received from a Web site for two queries are generated from a different database or by different server-side algorithms. At worst, such changes may occur during a single navigation session (i.e. some part of response the is received from a somehow modified Web site).

In general, it may be very challenging or even impossible to detect and handle such situations; however, it may be feasible if the data change patterns in a Web site are known. For example, if a site using pagination only adds new items at the beginning of a list (e.g. the content is ordered chronologically from the newest items), at worst some items will repeat in more than one result page, if some item(s) are added during a navigation session and pagination is not suited for such situations.

Another issue of response generation is related to the possible limitations of a Web server's, Web applications or Internet connection throughput that may concern a number of parallel connections, a number of parallel threads, or the total transfer per unit of time for all users, a specific segment of users (e.g. users of free of charge services), a specific IP address or a specific user logged in. It poses two important challenges. The first is to limit the number of requests or the size of downloaded data in a way that avoids running past these limits. The second is to react properly to HTTP errors or Web application error pages informing about limits being reached.

Response generation in complex Web applications (8, 9, 10b)

If content is generated by a Web application, it may store information (8) on or from a received request in the Web site's database (e.g. navigation history, information on a user's order, a visitor's personal information), a Web site's file system (e.g. file upload), user session (e.g. navigation history or history of user queries) and user profile (e.g. user interest, language, acceptance of Web site's "terms of usage").

In the case of dynamically generated content, the actual content (or relevant error message) returned to the Web browser is generated by server-side Web application logic (9). Apart from its internal algorithms, server-side Web application logic may use for content generation a Web site's back-end database(s), information stored in the current user session and user profile, as well as some static files. For example, a contemporary social search engine, in order to provide the best fitted search results, may combine information on user query (stored in HTTP request or session), and her/his long-term preferences (stored in user profile), with a list of objects coming from a database (search engine index).

In both cases, the content and HTTP headers generated by server-side Web application logic are subsequently returned to the Web server HTTP service (10b). According to its configuration, the HTTP service can add some additional headers to the content prepared to be sent.

One specific challenge of receiving data from the database is dealing with situations when server-side application limits the total number of records that can be received from a database (e.g. for any query it returns no more than 100 results split into pages of 25 items), or even requires that a query is specific enough not to return more results than a limit, and when it is too general no result is returned and the user is asked to make her/his query more precise. Such situations often require the query to be rewritten onto multiple queries returning a lower number of results that are further combined.

Interaction with client-side information storage (12, 13)

Upon receiving am HTTP response, client-side Web application logic may combine its contents with some information stored in client-side information storage (12). For example, it is possible that information storage contains a previously downloaded list of categories, and the received data consist only of identifiers of categories to be displayed next to each product.

At the same time, the client-side Web application logic may use the received content to update the client-side information storage (13). E.g. some labels from data received from the server may be loaded into information storage, to enable a better suggestion of query terms in a query definition interface.

When the obtained content is an executable code, it can perform any combination of two actions. Firstly, it may use information from the HTTP response and from client-side storage to update the current Web document. Secondly, it may use the HTTP response to update client-side information storage.

Integrating received content into Web browser (11a, 11b, 14)

If the HTTP request was issued by a Web browser or was issued by a client-side Web application but by using basic browser mechanisms (frame or document reload, loading additional CSS or image file), and the returned content is not an executable code, it is returned directly to the Web browser and modifies the current client-side browsing state (11a).

The challenge that is present at this stage is that how received content is interpreted depends strictly on the type of content. It is rather simple in the case of loading HTML, text or multimedia content (presented directly to the user), as well as CSS files (modifying presentation of HTML content), but it can be complex in the case of executable code, such as JavaScript or binary files interpreted by browser plugins (such as Flash).

In the case where the HTTP request was issued via a code-oriented mechanism (such as XmlHttpRequest control), or returned content contains some executable code (e.g. it is an additional script file loaded to the current document, or it is an HTML document that contains some “onload” event code), the control may be passed to client-side Web application logic rather than directly to the Web browser (11b).

Alternatively, both 10a and 10b can happen in parallel, if some new content is loaded directly into the browser, but at the same time it is interpreted by client-side Web application logic when detected by some event (onload, onDOMChange) or continuous content watching procedure (e.g. an infinite loop detecting new content in a frame).

In the case when control is returned to the client-side Web application, the received content is typically somehow transformed and used to modify the Web browser's content (14). In a simple case it may mean loading some ready-to-use content (e.g. a fragment of HTML code) into a location in the current Web page. In more complex situations it may mean rendering pure data (e.g. sent as XML or JSON) as a fragment of a Web page (e.g. as a table). In most complex situations it may mean updating multiple fragments of a page (e.g. items in a table, lists of product) by modifying some attributes and/or adding attributes to existing data. It may be implemented both by interpreting pure data received from the server by client-side logic, and by receiving new fragments of executable code that perform the update themselves (e.g. a few jQuery instructions that add extra data to each of the existing data records).

In the case of information extraction, it is in this stage that a key decision regarding the content used in extraction needs to be taken. The first option (available to programs but not to end users) is to capture the encoded data and perform extraction on it. The second choice it to perform extraction on user contentm i.e. on content contained in the Web browser after all client-side Web application logic actions were run or emulated. Each of these situations has specific challenges.

Working with encoded content poses the following challenges:

- Diversity of content formats: Web applications are not limited with respect to the format of data transfered. While it is a good practice to use standard formats such as XML or JSON, any binary or textual format may be used.
- Content compression or ciphering: encoded content need some processing before it can be used for data extraction.
- Dealing with executable code: if encoded content is expressed in JavaScript, code execution or its emulation may be necessary to receive useful content.

Challenges inherent in working with user content are:

- Complexity of Web browser content updates: updates may concern a relatively small part of a Web page. They may update dispersed parts of a page, and they may even combine data received from the server with data already present at the Web page.
- Transformation into a less structured format: it is relatively frequent that encoded data are much better structured than user content that is produced out of it (e.g. because some attributes are merged into a single HTML tag).
- Detecting the moment when the Web browser is ready for extraction: i.e. dealing with many asynchronous or system-triggered events that are needed to get user content ready, as well as ignoring requests that are not related to the content update (such as tracking codes, or advertisements).
- Dealing with no-HTML Web applications, e.g. developed in Flash.

Web browser content interpretation (15)

The next step is the understanding of (new or modified) Web browser contents by the user or agent that initiated the action (15).

In a scenario of information extraction, this is the step where actual information extraction from Web a browser's content or encoded data is performed. It is also the stage when a few significant challenges occur:

- all difficulties of information extraction listed in the previous literature review,

- dealing with non-textual content, such as graphical files (including text encoded as graphic files, text present in video files, and handling such special multimedia content as Flash or SliverLight),
- filtering data (i.e. applying query conditions that could not be included in requests sent to the server).

Usage of acquired information (16)

Once the information was acquired from the Web site, it can be applied in a specific usage scenario, i.e. interpreted, connected with other pieces of information, aggregated or filtered out by complex rules.

In the case of information extraction this step corresponds to result rewriting or integration with other sources, posing such challenges as:

- rewriting extracted data into the required data model,
- translating individual values into a user-specific vocabulary,
- applying complex filters that were not possible to be applied during navigation or data extraction.

Appendix B

Analyzed Web Sites

Category	Web Site	API/export?
Statistical Agencies	Eurostat	No
	GUS	No
	data.gov	No
	data.gov.uk	No
Financial Services	Yahoo! Finance	Yes[1]
	gielda.onet.pl	No
	www.xe.com/ucc/	No
On-line shops	Amazon	Yes
	Merlin	No
Auctions Sites	ebay	Yes
	Allegro	Yes
Comparison Shopping Services	Kelkoo	Yes
	shopping.com	Yes
	pricequest.com	No
	shopzilla.com	Yes
	ceneo.pl	Yes
	skapiec.pl	No
Product Test & Review Sites	dpreview.com	No
	benchmark.pl	No
	chip.pl rankings	No
Product Opinions Services	epinions	No
	cokupic.pl	No
	Yelp	Yes
Yellow Pages	Panorama Firm	No
Job Seach Services	monster.com	Yes
	pracuj.pl	No
	gazetapraca.pl	No

[1]RSS feeds and YQL querying possibilities.

Category	Web Site	API/export?
CS Bibliography Web Sites	DBLP	Yes
	ACM DL Search	No
	CiteSeer	Yes
	Google Scholar	No
People Search Engine	people.yahoo.com	No
	pipl.com	No
	people123.com	No
	ktokogo.pl	No
General Search Engines	Google	Yes
	Bing	Yes
	Netsprint	No
Q&A Site	Stack Overflow	Yes
Web Page Directories	DMOZ	Yes
	Yahoo! Directory	No
	ONET Katalog	No
Weather Services	Yahoo! Weather	No
	pogodynka	No
	pogoda.gazeta.pl	No
	new.meteo.pl	No
Traveling	TripAdvisor	Yes
	rozklad PKP	No
	expedia.com	No
Maps	Google Maps	Yes
	zumi	No
Images / Photos	flickr	Yes
	Google Image Search	Yes
	Fotka.pl	Yes
	fotolia	Yes
Music Web Sites	Last.fm	Yes
Video / Movies Sites	YouTube	Yes
	imdb	Yes[2]
	netflix	Yes
	filmweb.pl	No
	film.ipla.tv	No
TV programme	Telemagazyn	No
Events-Oriented Services	Yahoo! Upcoming	Yes
	www.poznan.pl	No
	www.eventim.pl	No
Bookmarks Management	delicious.com	Yes
	digg.com	Yes
	wykop.pl	Yes
Social Network Sites	facebook	No
	LinkedIn	Yes
	Nasza klasa	No
	grono.net	No
	Goldenline	No

Table B.1: General Dataset - Candidate Web Sites Split Into Categories

[2]Data export available

Web Site	1	2	3	4	5	6	**Technologies**
Comments							
1. Eurostat	●	●	◐	○	●	●	AJAX, PDF, Excel, SVG
AJAX tree of datasets; pivot tables with footnotes and legend; SVG graphs							
2. GUS	●	●	●	○	●	●	HTML, AJAX, Excel, PDF
Diversity of data sets; most require some navigation and AJAX; some use JS tree controls; pivot tables and graphs used							
3. data.gov	○	◐	●	○	●	●	HTML, Excel, PDF, RSS
Very diverse datasets, mostly presented as exportable file (in multiple formats); some via various interactive tools at third-party Web sites							
4. data.gov.uk	○	◐	●	○	◐	●	HTML, Excel, PDF
Very diverse datasets, most in file format							
5. gielda.onet.pl	●	●	◐	○	●	●	Flash/JSON, AJAX, Java/CSV
Simple data in simple GUI; more complete and complex data in graphs (Java; Flash)							
6. www.xe.com/ucc/	●	●	○	○	●	○	Flash/encoded data
Complex interaction (Flash, AJAX) and data presentation (graphs) for historical data							
7. Merlin	○	○	○	◐	○	○	HTML, JS
Relatively simple site with multi-faced and similarity-based graph data model							
8. pricequest.com	○	○	●	○	○	○	HTML, JS
Dynamic query refinement in few steps needed for some types of queries							
9. skapiec.pl	○	○	●	○	○	○	HTML, JS, some AJAX
Dynamic query refinement in few steps needed for some types of queries							
10. dpreview.com	○	○	◐	○	○	○	HTML, JS, some AJAX
Pagination of longer articles; some pivot tables in articles							
11. benchmark.pl	○	○	◐	○	○	○	HTML
Simple Web site with minimal navigation needs							
12. chip.pl rankings	○	○	◐	○	○	○	HTML
Simple HTML, navigation needed only for detailed technical specification							
13. epinions	○	○	●	◐	○	○	HTML
Simple HTML, navigation-based query refinement, multi-faceted graph							
14. cokupic.pl	○	○	●	◐	○	○	HTML
Simple HTML, navigation-based query refinement, multi-faceted graph							
15. Panorama Firm	○	○	●	○	○	○	HTML
Pagination of query results							
16. pracuj.pl	○	○	●	○	○	○	HTML
Simple HTML Web site with navigation-based access to individual job offers and advanced search							
17. gazetapraca.pl	○	○	●	○	○	○	HTML
Simple HTML Web site with navigation-based access to individual job offers and advanced search							
18. ACM DL Search	○	●	●	●	○	○	HTML, JS, some AJAX
Multi-faced navigation in people and references graph; some AJAX used							
19. Google Scholar	○	○	●	●	○	○	HTML, JS
Graph of people and citations							
20. people.yahoo.com	○	○	◐	○	○	○	HTML
Pagination							
21. pipl.com	○	●	◐	○	○	○	HTML, JS, AJAX
Page structure depending on what data exist for a given person							
22. 123people.com	●	●	●	○	○	○	HTML, JS, AJAX
AJAX-based pagination of some data sets; JS dropdown selection controls							
23. ktokogo.pl	○	○	●	●	○	○	HTML

Web Site	1	2	3	4	5	6	**Technologies**
Comments							
Graph of people and companies							
24. Netsprint	○	○	◐	○	○	○	HTML
Pagination							
25. Yahoo! Directory	○	○	●	●	○	○	HTML
Hierarchy with no constant depth							
26. ONET Katalog	○	○	●	●	○	○	HTML
Hierarchy with no constant depth							
27. Yahoo! Weather	○	○	○	○	○	○	HTML
Simple Web site, just very general forecasts							
28. pogodynka	○	○	◐	○	○	○	HTML
Navigation for disambiguation of geographical names							
29. pogoda.gazeta.pl	●	●	○	○	●	○	HTML, Flash/XML, JS
Flash-based user interaction (map and tables)							
30. new.meteo.pl	○	○	○	○	●	○	HTML, some AJAX
All data communicated as graphs in image format							
31. rozklad PKP	○	○	●	○	○	○	HTML, JS
POST requests, session-based content provision							
32. expedia.com	○	●	●	○	●	○	HTML, JS, AJAX
Complex session-based navigation with AJAX							
33. zumi	●	●	●	○	○	○	HTML, JS, AJAX, Flash
Flash asynchronous communication (in AMF format); complex Flesh map control							
34. filmweb.pl	○	●	●	●	○	○	HTML, some Flash, AJAX
Complex data (graph), AJAX with explicit obfuscation							
35. film.ipla.tv	○	○	●	●	○	○	HTML
Few different approaches to navigation or search, but each limited							
36. Telemagazyn	○	●	○	●	●	○	HTML, AJAX
Graph of films and series, some data not linked (people); table data presentation; some AJAX							
37. www.poznan.pl	○	○	●	○	○	○	HTML
Simple organization with limited querying capabilities							
38. www.eventim.pl	●	●	◐	○	●	○	HTML, Java
Basic event information in simple HTML format; complex price and free seats preview in Java with asynchrounous communication							
39. facebook	◐	◐	●	●	○	○	HTML, AJAX
Complex graph-data model, some AJAX, need to be logged in to access some data							
40. grono.net	○	◐	●	●	○	○	HTML, AJAX
Complex graph-data model (people, intersts, forums, friends), some AJAX, need to be logged in to access most of data							
41. Goldenline	○	○	●	●	○	○	HTML, JavaScript
Complex graph of people, groups and forums							

Table B.2: General Dataset - Preliminary Analysis: 1 - complex UI elements; 2 - complex server interaction; 3 - need of navigation; 4 - complex data model; 5 - complex data presentation; 6 - multiple data sets

Appendix C

List of Information Extraction Challenges

Challenges Topics	L	IM	WS	Chall.
I **Different data models**				
I.1 **Complexities of data entities**				
Basic data types				
I.1.a Simple types (integer, float, text, dates)		15		D L
I.1.b Domain-specific types (codes, phones, addresses)		15	11, 15, 21, 22	D L
Record-like structures				
I.1.c Records with stable structure	[155, 180, 97, 66]	15		D L
I.1.d Optional attributes	[155, 180, 97, 66]	15		D L
I.1.e Data with unknown complete list of attributes (schema)		15		D
I.1.f Data with unknown data types of individual attributes		15		D
Composite types				
I.1.g Lists of embedded records with known structure and cardinality		15		D L
I.1.h Lists of embedded records with known structure and varying cardinality	[155, 180, 66]	15		D L
I.1.i Optional lists of embedded records		15		D L
I.1.j Embedded hierarchies of unknown depth	[180]	15		D L
I.1.k *Domain-specific data structures such as streams of data, time series, unordered sets, geographical data*		15	1, 5	D A
I.2 **Complexities of relations**				
Hierarchical relations				
I.2.a Tree-like structures		1, 15, 16	25, 26	D
I.2.b Star schemas		1, 15, 16	39, 40	D

Challenges Topics	L	IM	WS	Chall.
Graph relations				
I.2.c Directed relations		1, 15, 16	18, 23, 34, 39, 40, 41	D
I.2.d Undirected relations		1, 15, 16	7, 23	D
I.2.e Cyclic graphs		1, 15, 16	7, 18, 19, 23, 34, 36, 39, 40, 41	D
I.2.f Acyclic graph		1, 15, 16		D
Relations stability				
I.2.g Known entities types and depth		1, 15, 16		D A
I.2.h Known entities types and unknown depth		1, 15, 16		D
Relations labels				
I.2.II Unlabeled but explicit relations (e.g. see also references)		1, 15, 16		D L
I.2.j Text labeled relations (e.g. hyperlinks)		1, 15, 16	18, 19	D A
I.2.k Typed relations		1, 15, 16	18, 19	A
I.2.l *Implicit relations (no hyperlinks, just names or id identity)*		1, 15, 16	36	D
II **Server interaction challenges**				
II.1 **Single Request**				
Different Request Methods				
II.1.a GET	[107]	3a, 6		D
II.1.b POST	[107]	3a, 6	31	D
Different HTTP Headers				
II.1.c Redirects	[107]	3a, 6	32, 39	D
II.1.d Errors (e.g. 404, 500)	[107]	3a, 6		D
II.1.e Client-side content caching	[78]	3a, 6, 12, 13		D
II.1.f Cookies	[107, 201]	3a, 6, 12, 13	39, 40	D
HTTPS/SSL				
II.1.g HTTPS Requests	[219, 107]	3a, 6		D
II.1.h Certificates	[219, 107]	3a, 6		D
Data transport				
II.1.i Handling data compression		3a, 6		D
II.1.j Capturing and using sent content		3a, 6	2, 5, 33	D
HTTP / Web application session				
II.1.k Session ids	[219]	3a, 6		D L
II.1.l Session expiry and reinstantiation			31, 32	D L
II.1.m Different methods of user authentication	[201]	3a, 6		D
II.2 **Multiple requests**				
Request alignment				
II.2.a Synchronous requests		6		D
II.2.b Asynchronous requests		6		D
II.2.c Alignment of multiple synchronous and asynchronous requests and their results		6, 11b, 14		D
Being "polite" towards servers				
II.2.d Limiting number of requests	[78]	1		D
II.2.e Limiting data transfer	[78]	1		D
Per user / IP / session server limitations				
II.2.f of number of parallel requests		3a, 6		D L

Challenges Topics	L	IM	WS	Chall.
II.2.g of number of requests per unit of time	[201]	3a, 6		D L
Global server limitations				
II.2.h of number of parallel requests		3a, 6		D L
II.2.i of number of requests per unit of time		3a, 6		D L
Distributed request issuing challenges				
II.2.j Splitting some request between multiple agents/IPs		1		D
II.2.k Issuing some sequences of requests from single IP		1		D
II.2.l Managing temporal alignment of distributed requests	[201]	6		D
II.2.m Dealing with personalized, adaptive and context-aware Web sites		1	39, 40	D
II.3 **Detectability**				
Behaving similarily to normal user				
II.3.a Organizing request into logical data paths		1		D
II.3.b Avoiding too numberous or too regular request issuing		1		D
II.3.c Avoiding causing HTTP errors or issuing queries with empty results		3a, 6		D L
Reproducing browser behaviours				
II.3.d Downloading related files		3a, 6		D
II.3.e Handling Cookies and JavaScript		4, 5, 12, 13	39, 40	D
II.3.f *Limitting downloads from single IP address*		1		D
III **Server-side Web application logic challenges**				
III.1 **Keyhole effects**				
Querying capabilities - individual attributes				
III.1.a Attributes that cannot be directly queried	[143]	3a, 6	7, 15, 19, 35	D
III.1.b Limited attribute operators		3a, 6	7, 15, 19	D
III.1.c Range or category-based querying of attributes		2, 16		D
III.1.d Source-specific codes/names for attributes values		3a, 6		D L
III.1.e Required attributes	[143]	2, 3a, 6		D L
Querying capabilities - Multiple attributes				
III.1.f Limited combinations of querable attributes	[143]	2, 3a, 6		D L
III.1.g Limited Boolean operators between multiple conditions		2, 3a, 6		D L
III.1.h No attributes comparison facilities		1, 16		D
Additional Web site's query capabilities not exposed in user interface				
III.1.i Additional attributes		3a, 6		A L
III.1.j Additional operators		3a, 6		A L
III.1.k Additional attribute combinations allowed		3a, 6		A L

Challenges Topics	L	IM	WS	Chall.
III.1.l Additional Boolean operators allowed		3a, 6		A L
Limit of number of results				
III.1.m Constant for any query		1, 16		D L
III.1.n Depending on type of query		1, 16		D L
III.1.o Depending on query complexity		1, 16		D L
IV **Client-side Web application logic challenges**				
IV.1 **User Interface Interactions**				
IV.1.a *Hyperlinks*	[219]	2, 3a, 6		D
HTML form elements				
IV.1.b Closed-domain HTML controls (select, checkbox and radio)	[201]	2		D A
IV.1.c Open-domain text, textarea controls	[201]	2		D
IV.1.d Controls with “suggest” functionality	[201]	2, 11b, 14	31	A L
IV.1.e CAPTCHAs	[201]	2		D
Non-standard, JavaScript user interface controls				
IV.1.f Based on “on click” events	[219, 201]	2, 3	2, 5, 22	D
IV.1.f Based on “on mouse over” events	[201]	2, 3	22	D
IV.1.g Based on “drag and drop” events		2, 3	33	D
IV.1.h Based on keyboard events		2, 3		D
IV.1.i Based on page scrolling		2, 3		D
IV.1.j Based on time elapsed		2		D L
IV.1.k *HTML5 elements*		2		D
Embedded controls				
IV.1.l Java controls	[201]	2, 14	2, 38	D
IV.1.m Flash controls	[201]	2, 14	5, 29, 33	D
IV.1.n Other (ActiveX, SVG, SilverLight)		2	1	D
IV.2 **User interface elements recognition / interpretation**				
Forms parsing				
IV.2.a Detecting lables		2, 15		L
IV.2.b Mapping elements onto query attributes		2		L
Other elements recognition				
IV.2.c Detecting lables		2, 15		L
IV.2.d Mapping elements onto query attributes		2		L
IV.3 **Emulating basic Web browser mechanisms**				
Loading other external files				
IV.3.a Images		2		D
IV.3.b CSS files		2		D
IV.3.c JavaScript files	[219, 107]	2, 11b, 12, 13, 14		D
IV.3.d Frames and Iframes	[219]	2, 12, 13	21, 22, 32	D
Running scripts				
IV.3.e Body part of loaded JavaScript files	[219, 107]	2		D
IV.3.f Inline head and body scripts	[219, 107]	2		D
IV.3.g Functions run in “on load” event(s)	[219, 107]	2	22, 32, 34, 39	D
IV.3.h *HTML-level (HTTP-Equiv) redirects*		2		

Challenges Topics	L	IM	WS	Chall.
IV.4 **Request construction**				
From hyperlinks				
IV.4.a Absolute addresses		2, 3a, 6		D
IV.4.b Relative addresses with HREF BASE set		2, 3a, 6		D
IV.4.c Relative addresses without HREF BASE		2, 3a, 6		D
IV.4.d *From HTML forms*		2, 3a, 6		D
From other controls				
IV.4.e Controls constructing requests via forms hidden fields		6	32	A
IV.4.f Controls manually building requests		6		D
V **Extracting information from different types of content**				
V.1 **Types of content**				
Static				
V.1.a Text documents	[97, 66]	15		D
V.1.b HTML documents	[72, 97, 66]	15		D
V.1.c Graphical files		15	2, 4, 29, 30	D
V.1.d Specific binary formats (e.g. PDF, Excel, SVG)	[201]	15	1, 2, 3, 4	D
V.1.e Structured text formats (JSON, XML, AMF, CSV, ...)	[97, 66, 201]	11b, 15	2, 3, 4, 5, 29	D
Composite Web content				
V.1.f Multi-frames / iframes documents		15	4, 21, 22	D
Content updated on "on load" event, JavaScript timer, HTML-level redirect/refresh				
V.1.g client-side generation	[201]	14	39	D
V.1.h AJAX modifications		11b, 14	22, 39	D
Content updated on user events				
V.1.i client-side generation		2, 14		D
V.1.j AJAX data download		2, 11b, 14	1, 2, 6, 9, 10, 18, 22, 32, 34, 39	D
Continuously updated pages				
V.1.k Regular, on-line updates		11b, 14		D
V.1.l Server-event-driven updates (e.g. chat)		11b, 14	39	D
Embedded binary Web applications				
V.1.m Java applets	[201]	15	2, 5, 38	D
V.1.n Flash controls	[201]	15	5, 6, 29, 33	D
V.1.o Other embedded objects (Silverlight, SVG)	[201]	15	1	D
V.1.p *Purposefully obfuscated content*		15	34, 39	D
V.2 **Content capture**				
Capturing transfered content				
V.2.a Basic capture		11b	2, 5	D
V.2.b Capturing cyphered data		11b		D
V.2.c Capturing compressed data		11b	2, 5	D
V.2.d Capturing data in persistent connections		11b		D
Capturing complex Web content				

Challenges Topics	L	IM	WS	Chall.
V.2.e Identifying time when content is ready		14		D
V.2.f Capturing current Web browser state		14	34, 39	D
V.2.g Representing multiple frames as single document		15	4, 21, 22	D
V.2.h Identifying new data (for AJAX-like updates)		11b, 14		D
VI **Data extraction**				
VI.1 **Using different features**				
Mark-up				
VI.1.a Pure code	[180, 97]	15		D L
VI.1.b Document Object Model representation (or similar)		15		D L
VI.1.c Semantic annotation (microformats, RDFa)		15		A L
VI.1.d Explicit references (hyperlinks)		15		D L
Visual features				
VI.1.e Positioning: Size, position and spacing		15		D L
VI.1.f Value Colors, fonts, contrast		15		D L
Page text features				
VI.1.g Syntax		15		D L
VI.1.h Vocabulary patterns		15		D L
VI.1.i Attribute markers / delimiters	[180, 97]	15		D L
VI.1.j Attribute-specific data formats	[180, 66]	15		D L
VI.1.k Implicit references (footnotes, legends)		15		D L
Data model assumptions				
VI.1.l Attributes order	[155, 180, 97, 66]	15		A L
VI.1.m One-to-one relations		15, 16		A L
VI.1.n One-to-many relations		15, 16		D L
VI.1.o One-to-n relations (n known in advance)		15, 16		A L
VI.1.p One-to-n relations (n extractable from the source)		15, 16		A L
VI.1.r *Extraction rules combining many features*		15		A L
VI.2 **Extraction of composite data**				
VI.2.a *Object identification*		15		D
VI.2.b *Single value extraction*		15		D
VI.2.c *Value-to-attribute mapping*		15, 16		D
VI.2.d *Linking values to objects*		15, 16		D
VI.3 **Different data presentation models**				
Complex data presentation structures				
VI.3.a Hierarchical lists	[180]	15	1, 2	D
VI.3.b Non-contiguous (vertical) records	[190]	15		D
VI.3.c Pivot tables		15	1, 2, 10, 36	D
VI.3.d Dynamic grids		15	1, 2	D
VI.3.e Dynamic graphs		15	5, 6	D
References to other part of data				
VI.3.f Explicit (e.g. hyperlinks)		15, 16		D
VI.3.g Implicit (e.g. legend, footnotes, unique names)		15, 16	1, 2, 36	D L

Challenges Topics	L	IM	WS	Chall.
Number of objects in single page				
VI.3.h One	[66]	15		D L
VI.3.i Multiple (known number)	[66]	15		A L
VI.3.j Multiple (unknown number)	[66]	15		D L
Dynamic Web pages				
VI.3.k Extraction of just new data		11b, 14, 15	39	D L
VI.4 **Extraction robustness**				
Noise in extraction content				
VI.4.a Extra random elements in page	[72]	15	21	D L
VI.4.b Extra random elements between data records	[72]	15		D L
VI.4.c Extra random elements within data records	[72]	15		D L
Template and data-level instability				
VI.4.d Varying ways of presenting the same information		15	22	D L
VI.4.e Varying attributes order and cardinality	[155, 180, 97]	15		D L
VI.4.f Typographical errors	[155, 180]	15		D
VI.4.g Varying information patterns for unstructured text		15		D
I.1.h Untokenized attributes	[66]	15		D L
Page evolution over time				
VI.4.i Template (HTML code) evolution	[72, 78]	15		D L
VI.4.j Data presentation evolution		15		D L
VI.4.k Data model evolution		15		D L
VI.4.l Detection when wrappers stop to work	[283]	15		D
VI.4.m *Alternative extraction rules using different features*		15		A
VII **Navigation planning**				
VII.1 **Web site modeling**				
Modeling types of pages				
VII.1.a Data contained	[66]	15		D L
VII.1.b Template		15		D L
Modeling types of actions				
VII.1.c HTTP requests	[219]	3		D L
VII.1.d GUI actions		2, 3, 12, 13, 14		D L
VII.1.e Actions initiated by client-side application logic		2, 14	21, 32	D L
Modeling actions executed at specific pages				
VII.1.f Impact on contained content	[66]	1		D L
VII.1.g Impact on data constrains (actual query)		1	8, 13, 14, 16, 17	D L
VII.1.h Impact on client-side and server-side session		4, 5		D L
Modeling additional actions not exposed by Web site GUI				
VII.1.i Other requests patterns		3a, 6		D L
VII.1.j Acquisition of related data in other part of Web site		1		D L

Challenges Topics	L	IM	WS	Chall.
Modeling client-side and server-side state evolution (stateful and adaptive Web sites)				
VII.1.k Sequences needed to follow		4, 5, 12, 13	8, 13, 14, 16, 17, 26, 27	D L
VII.1.l Needed session resets		12, 13		D L
Modeling data locations within Web site				
VII.1.m Indexes usage				D L
VII.2 **User query rewriting into**				
VII.2.a *Navigation plan*	[219]	1, 3a		D
VII.2.b *Request patterns*		1, 3a, 6		D
VII.2.c *Extraction rules*		1, 15		D
VII.2.d *Post-extraction filters*		1, 16		D
VII.3 **Choosing right navigation scope**				
Including all needed pages				
VII.3.a Pages containing data	[219, 72, 66]	1, 15		D
VII.3.b Pages containing definitions of next actions	[219, 72]	1, 3		D
VII.3.c Pages needed for session management		1, 4		D
Filtering out unnecessary requests				
VII.3.d Cutting navigation plan branches		1, 3a, 6		D
VII.3.e Skipping unnecessary navigation steps		1, 3a, 6		D
VII.3.f Ignoring unnecessary extra requests (images, CSS, etc.)		1, 3a, 6		D
Taking into account source behavior				
VII.3.g Querying capabilities		3a, 6		D
VII.3.h Number of results limitations		7a, 8, 9, 10b		D
VII.3.i Dependence on IP or location		9		D
VII.3.j Personalization		8, 9	36, 39, 40	D
VII.3.k Adaptiveness		8, 9		D
Working with too large data sources strategies				
VII.3.l Prioritization of some part of data		1, 16	39	D
VII.3.m Some attribute for many objects first (many incomplete records)		1, 16		D
VII.3.n All attributes for some object first (few complete records)		1, 16		D
Choosing the best of multiple possible navigation plans				
VII.3.p Optimizing path length		1		D
VII.3.r Optimizing transfered content overhead		1		D
VII.3.s Optimizing usage of data ordering (if exposed)		1, 7a, 9, 10b		D
VII.3.t Downloading prioritary data first		1		D
VII.3.u Statistics creation and maintenance		1		D
VIII **Navigation execution**				
VIII.1 **Interpretation of current navigation state**				
Using different features				

Challenges Topics	L	IM	WS	Chall.
VIII.1.a Using request URLs patterns		1, 16		D
VIII.1.b Using history of previous requests		1, 16		D
VIII.1.c Using some data in previous pages		1, 16		D
VIII.1.d Using characteristic HTML elements		15		D
VIII.1.e Using in-depth page analysis (e.g. parsing, similarity analysis)		15		D
VIII.1.f Using specific client-side or server-side events		14		D
Data-level interpretation				
VIII.1.g Actual query of contained data		1	8, 13, 14, 16, 17	D
VIII.1.h Scope of contained objects attributes		1		D
VIII.1.i Data presentation applied		1, 15		D
VIII.1.j Readiness of data to be downloaded		14		D
Navigation-level interpretation				
VIII.1.k Possible next actions		1, 15	8, 13, 14, 16, 17	D
VIII.2 **During execution adaptation of navigation plan**				
Detailing navigation plan				
VIII.2.a Pagination		7a, 10b	7, 8, 9, 10, 13, 14, 15, 20	D
VIII.2.b Hierarchies with unknown depth		1, 15, 16	25, 26	D
Data moving between pages during navigation				
VIII.2.c Based on known ordering (e.g. sorting, temporal alignment)		7a, 9, 10b		D
VIII.2.d Based on server-side variables (e.g. number of user votes)		7a, 9, 10b		D
Non-deterministic patterns specification				
VIII.2.e Identifying specific page attained through an action from multiple possibilities		15	28, 31	D
Modifying navigation plan due to number of results limitations				
VIII.2.f More results than allowed by site		1, 7a, 9, 10b		D
VIII.2.g Less results than allowed by site		1		D
Unexpected situations				
VIII.2.h Technical problems (e.g. HTTP errors)		7a, 10b		D
VIII.2.i Passing site limits		7a, 9, 10b, 15		D L
VIII.2.j No results pages		7a, 10b, 15		D L
VIII.2.k Layout changes		7a, 10b, 15		D
VIII.2.l Navigation model changes		7a, 10b		D
VIII.2.m Source model incompleteness		7a, 10b		D
Duplicate data				
VIII.2.n Expected (e.g. resulting from overlapping categories)	[78]	16	18, 19	D L
VIII.2.o Unexpected		16		D
IX **Usage scenarios specificities**				

Challenges Topics	L	IM	WS	Chall.
IX.1 **Data materialization / content indexing from a source**				
Building set of queries returning complete data				
IX.1.a Navigation-only Web sites		2, 3a, 6		D A
IX.1.b Web sites with closed-domain query interfaces		2, 3a, 6		D A
IX.1.c Web sites with open-domain query interfaces		2, 3a, 6		D
IX.1.d Web sites with limited number of results		2, 3a, 6		D
IX.1.e Data overlapping between multiple queries		1, 16		D A
IX.2 **Issuing as hoc queries**				
Quick reception of most useful data				
IX.2.a Data prioritization		1, 16		D
IX.2.b Partial data reception and display		1, 16		D
IX.3 **Data monitoring**				
Handling queries for individual objects				
IX.3.a Unambiguous identification of objects		15		D
IX.3.b Data movements in the Web site		1, 16		D L
IX.3.c Indexes or cache for quicker access	[78]	1, 16		A L
IX.3.d *Calculating revisit frequency*		1		D
IX.3.e *Combining revisits of multiple objects to minimize overhead*		1		D
IX.4 **Integrated querying of multiple sources**				
Schema-level inconsistencies				
IX.4.a Mapping individual schema attributes	[78]	16		D
IX.4.b Transformation of multiple attributes between schemas		16		D
Data-level inconsistencies				
IX.4.c Matching of user query values to source query interface	[78]	3a, 6		D
IX.4.d Transforming result values to user view values		16		D
Working with multiple sources				
IX.4.e Source discovery and selection	[78]	1		D L
IX.4.f Query rewriting for multiple sources		1		D L
IX.4.g Source probing (scope of data analysis)	[78]	1		D

Table C.1: Complete List of Information Extraction Challenges Areas with Sources (L - literature review, IM - corresponding steps in user interaction model, WS - Web sites with specific challenges) of Evidence and Individual Challenges (column Chall.: D - "dealing with", A - "taking advantage", L - "learning"

Appendix D

Presentation of Existing WIE Systems

In this section we review 42 Web information extraction solutions developed from the early 1990s to the first decade of the 21st century. While, due to large body of research in this area in recent years, the list of described systems is surely not exhaustive, the covered solutions are representative for all major research directions. For the interested reader we provide a list of a number of other Web information systems not described in this thesis in Section D.43.

D.1 TSIMMIS

The Stanford-IBM Manager of Multiple Information Sources, more widely known as TSIMMIS, was developed in a joint project of Stanford University and the IBM Almaden Research Center in the years 1994–96. The focus of this project was on developing "tools that facilitate the rapid integration of heterogeneous information sources that may include both structured and unstructured data" [69], rather than on the automation of information extraction or integration.

The TSIMMIS project proposed a system architecture that shaped many future data integration systems. It was composed of:

- **common data model** enabling actual data integration,
- **storage facility** for data objects,
- **query languages** to enable data access,
- **user interface** for intuitive query construction,
- **mediators** enabling source selection and query rewriting, and
- **data wrappers** giving access to specific data sources.

A common data model proposed was the Object Exchange Model (OEM), a relatively simple data model supporting object identity, nesting, lists and sets. Storage of OEM objects was assured by the LORE repository [273]. Query languages proposed were LOREL, extending OQL [273, 145] and a SQL-like language called OEM-QL [69]. An innovative user interface called MOBIE was available through Mosaic Web browser, enabling the hypertextual browsing of OEM objects. The repository was presented to the user by applying DataGuides [130], a kind of data schema (capturing possible path expressions over nested data), dynamically created based on the actual contents of a specific OEM repository. Mediators supporting sources selection, result merging and extra domain-specific data processing logic, accept OEM-QL queries, rewrite them onto wrappers on other mediators and return results as OEM objects. Wrappers (also called translators), built specifically for individual sources, accept OEM-QL queries and return corresponding data from sources translated into OEM objects.

Apart from general modular architecture, TSIMMIS has two specific contributions in working with Web sources. The first was seminal work on capabilities-based query rewriting [267], allowing algorithms to decompose complex queries into a set of simpler ones that can be answered by Web sources with limited querying capabilities. The second was a data extraction formalism described in [144] transforming HTML pages into OEM objects. It was based on a procedural extraction language using named variables, extraction patterns (simple regular expressions applied to HTML code) and a number of in-built functions (such as `get` for accessing document at a given URL and `split` for constructing an array out of a string).

D.2 W3QS

W3QS [172, 173], developed in Israeli Technion, was the first (of many) SQL-like hypertext query languages constructed specifically for the World Wide Web. While its actual implementation was limited as compared with its model capabilities, it introduced a number of important ideas for other query languages and data extraction systems.

W3QS proposed a highly formalized query language covering two distinct types of queries: content queries (with predicates specified for Web page content) and structure-specifying queries (with predicates specified for navigation patterns). Its navigation support included automated, data-driven form filling with support for learning how to fill in new forms, as well as a graph-based modeling of navigation patterns with the support of “self-loop” (i.e. pages linking to pages of the same type, as in the case of pagination). W3QS authors introduced also the concept of automatically updated views over Web content (pioneering Web monitoring ideas, however, without any advanced implementation). Finally, at the architecture level, W3QS underline importance of using existing software packages and “pipelining” output into other systems.

D.3 Information Manifold (IM)

Information Manifold [143, 142] was another early data integration system capable of working with many diverse data sources, including data-intensive Web sites. In contrast with TSIMMIS, which makes use of specific mediators for query answering, Information Manifold relies heavily on the shared data model (set of relations), description of individual sources in terms of this general model and on general query rewriting algorithms based on views.

Source descriptions, being a central element of the IM model, consist of contents description and query capabilities. Contents description (source coverage) is defined using predicates met by all data contained in a source. It enables the quick assessment of the relevance of a given source to a user query. Query capabilities describe the in (queryable) and out (returned) attributes of a specific source, and a minimal and maximal number of queryable attributes that can be submitted to the query interface. Apart from logic-based description, the probabilistic extension of this model was further proposed [121]. Given a user query expressed by using the global data model, IM is able to choose relevant data sources (possibly containing data relevant to the user query and capable to answer this query through their limited query interfaces), building a query plan, and executing it.

As compared with TSIMMIS, IM may sometimes build better performing query plans, makes it easier to add new data sources (without rebuilding mediators), and provides a direct way of describing source query capabilities. However, at the same time it requires a stable shared data model and does not support semi-structured data with varying schemas [266].

It is to be noted that while Information Manifold enables users to build wrappers for individual Web sources, it does not analyze the problem of information extraction itself. It is due to Alon Halevy's conviction (proved wrong by today's reality) that with the advance of XML technology the need for extraction will quickly diminish or even disappear [140].

D.4 WebLog

WebLog [182] is a declarative logic-based Web querying and restructuring language with rich syntax and semantics. Like W3QS, WebLog enables complex queries combining both keyword-based content queries and hyperlinks-based navigation queries (with recursive support for structures with unknown depth such as pagination), and has support for integrating existing code libraries into query language (in a somehow more elegant way). In contrast with W3QS, it makes Web pages restructuring possible; however, it does not analyze Web monitoring issues addressed by W3QS. While the proposed language is sound and elegant, its implementation status is unknown, and a very general way of specifying data extraction queries make it potentially relatively computationally expensive.

D.5 WebSQL

Developed at the University of Toronto, WebSQL [207] is a Web query language, somehow similar to the language used in W3QS. It combines content-based conditions with conditions based on *path regular expressions*. Content-based conditions (based on keywords contained in documents) can be placed in two WebSQL constructs: in the FROM clause (resolved by using existing search engines) or in the WHERE clause (resolved at query time). Path regular expressions are based on the distinction between interior links (links within the same document), local links (links to local documents) and remote links (links to other Web pages). They allow users to define path patterns (limited, distinguishing only the three aforementioned classes of symbols) using regular expressions supporting Kleen star and alternatives. Moreover, WebSQL takes into consideration a simple cost model based on the same three classes of hyperlinks.

As compared with W3QS, WebSQL gives opportunities to use external indexes of documents as well as a relatively simple user interface for building queries, but ignores the subjects of dynamic views, integration with existing software and form filling, and provides limited navigation and extraction capabilities.

D.6 Ashish and Knoblock Work (AK)

Ashish and Knoblock in their work [28] proposed a solution for the semi-automated learning of data extraction wrappers based on a few HTML-level heuristics. Their proposed solution is well-suited for Web sites with a very rigid HTML structure for different data records, and handles two classes of Web sites: Web sites with multiple single-object documents of identical structure (called by the authors *multiple instance sources*), and Web sites providing a single page with multiple data records of the same structure (called *single instance sources*).

Wrapper learning is performed in three steps: structuring the source, building a parser, and providing wrapper communication with mediator. Structuring the source consists in applying heuristics to HTML code in order to identify important tokens and the document's hierarchical structure. It is done in an automated way, with the possibility of user post-corrections. In the second step, the parser is built automatically by using the YACC library over identified tokens. Finally, the third step consists in defining an URL mapping function that accepts some query parameters and builds a corresponding HTTP GET request. This proposed solution introduced an important direction of automated wrapper construction based on HTML code analysis. However, due to its sensitivity to even the slightest irregularities in HTML code, its applicability was limited.

D.7 Webfoot

Webfoot [253] is an information extraction system that combines Web page HTML code parsing with previously-developed NLP-based information extraction methods implemented in the CRYSTAL system [255]. Webfoot uses four levels of delimiters for discovering segment (record) and field boundaries. Any of these levels may combine domain-independent and domain-dependent delimiters. The delimiters used, all expressed in a very simple form of regular expressions, are both HTML tags and phrasal expressions, such as "line beginning with word(s) followed by :". The output of Webfoot processing is passed to CRYSTAL, which uses semantic lexicon and a POS parser for the supervised learning of extraction rules.

D.8 ShopBot/Jango

ShopBot, called also Jango [93, 94], was one of the first commercial information extraction tools. Developed at the University of Washington as a "domain-independent shopping agent", it handled basic Web site navigation (form parsing and filling in), and data extraction from product descriptions. It separated wrapper learning and usage phases, supported (to some extent) sources search and discovery, and handled a few problems related to "failure pages".

The proposed solution, which "relies on a combination of heuristic search, pattern matching, and inductive learning techniques" [93], is based on assumptions that Web sites of interest contain some search form with "navigation regularities", and that result pages are characterized by the "uniformity regularity" (consistent look and feel), and "vertical separation regularity" (consistent use of white space).

Wrapper learning in the ShopBot system starts with crawling the Web site in search of HTML forms and the vocabulary-based rejection of forms that are clearly not search forms (e.g. newsletter forms). Then, a dictionary of typical labels is used to recognize form fields. Next, some dummy non-existent query is posed in order to receive a "failure page" used in the phase of "failure page learning". The following step consists in the automated learning of naive rules for stripping the header and footer and for information extraction (rules are based on the identity of page code for multiple pages; thus, they work only for very regular templates).

Its full automation makes ShopBot one of the most interesting early information extraction systems. However, it has two key limitations. Firstly, it was based on some heuristics that worked well in a specific domain (on-line shops) with well-structured pages. Secondly, it had no support for multislot attributes, missing values or attribute permutations.

D.9 WIEN

Wrapper Induction ENvironment (WIEN) is a system implementing multiple wrapper induction algorithms developed at the University of Washington by Nicholas Kushmerick [178, 179]. It was the first system addressing the fully automated, inductive wrapping of Web sources, working with a variety of Web sites presenting data as records with a stable structure (i.e. with no optional attributes or attribute permutations).

It poses the problem of the supervised learning of data extraction wrappers for multiple single or multiple-object pages with the same structure, based on a few examples of such documents and associated data. WIEN treats HTML documents as text strings and looks for specific delimiters (any strings combining text and HTML tags) that identify the values of individual attributes, and (in more complex wrappers) relevant parts of pages.

The simplest class of wrappers (called "Left-Right" or shortly LR) is expressed as a separate pair of a left and right delimiter for each attribute. While intuitive and easily learned, this class of wrappers requires that identified delimiter pairs are not used in any other part of a Web page in any other situation than surrounding the specific attribute values of a specific record. This limitation of the LR class led to the proposal of a few other wrapper classes and corresponding learning algorithms. The Head-Tail-Left-Right (HTLR) class adds head and tail delimiters that mark the beginning and end of the actual relevant content of the page, making it possible to ignore the head and tail parts of the page. The Open-Close-Left-Right (OCLR) class adds open and close delimiters that mark the beginning and end of each individual record in the page, making it possible to handle multi-object pages. The HTOCLR class combines the delimiters of HTLR and OCLR. Finally, the two last classes of wrappers introduced a focus on handling hierarchically nested data: the N-LR class uses only left and right delimiters and N-HTLR has also support for head and tail delimiters.

D.10 Araneus

Araneus [204, 31, 32] is a Web data management system jointly developed by the Universita di Roma Tre and the Universita della Basilicata in late '90s. It proposes a holistic approach to Web data based on the notion of Web bases that support three main types of usage scenarios: posing queries, defining the view and updating Web data. The system supports round-trip interaction with Web data: it implemented the extraction and querying of relational data from the Web, but also enabled the creation of hypertext from relational data.

The Araneus system consists of four key components: the Araneus Data Model (ADM) for describing page-centric data schemas, Ulixes supporting defining views on data sources, Editor/Minerva for creating and using wrappers for specific Web sites, and Penelope for developing new Web sites out of data (so-called derived hypertexts).

ADM is the shared data model that helps other components of the Araneus system exchange information. It allows user to describe the data model of specific Web sites. Each page, identified by an URL, has a so-called page-scheme combining simple types, lists of multivalued tuples and optional attributes (without structures nesting support). HTML forms are modeled as virtual tuples (composed of all editable form elements and an URL address corresponding to the filled in form). As some URLs may return pages with slightly or significantly different schemas (e.g. depending on user query), ADM also supports heterogeneous union, allowing one page to have multiple page-schemas corresponding to different variants of a given page. The structure of data in the whole Web site forms the scheme which is a directed multigraph of page-schemes.

Ulixes is the component used to define views over the ADM representation of a Web site, which can be manipulated or integrated in a relational way. Ulixes views are based on navigational algebra (NALG) that operates on ADM objects and implements traditional relational operators (selection, projection, join), as well as Web specific operators: "unnest" (navigation within a page, marked with "`.`") and "follow link" operators (navigation between pages, marked with "`->`"). Apart from this, NALG supports two types of constraints. Link constraints ensure the consistency of the chosen attributes between multiple pages of navigation path. Inclusion constraints describe the relation between data in two locations, specifically that a list of tuples with the same scheme at a specific location is a subset of a list from other location (navigation path containment).

The navigational queries used in Ulixes are developed manually; however, choosing between possible query execution plans is to some extent automated thanks to using cost-based statistics (related more to page download time than to local CPU processing costs).

As the Web sources used in Araneus do not provide data in ADM representation, there is a need for the transformation of Web pages into filled in ADM schemas. This is performed using two data extraction components, i.e. Editor and Minerva. Editor [30, 82] is a procedural language based on operations such as search (based on Aho-Corasick regular expressions with

* wild card and positive look-aheads), replace, cut, copy, paste, as well as basic control structures such as loops and if-else. Minerva [82] is a formalism for wrapper generation over semi-structured and Web data sources combining declarative parsing (grammar supporting supports optional patterns, as well as "*" and "+" wildcards) with the procedural handling of exceptions using Editor language.

The whole Araneus project focused mostly on working with HTML sources. However, as illustrated in [206], it can be also somehow applicable to the plethora of XML information found on the Web.

D.11 WebOQL

WebOQL [26, 27] is an advanced Web query and data restructuring language developed at the University of Toronto. It combines the features of systems supporting Web queries (such as WebSQL, W3QS and WebLog), querying of semistructured data (such as Lorel or UnQL), Web site restructuring (such as Araneus or Strudel) and the querying of structured documents.

The WebOQL data model consists of hypertrees and Webs. Hypertrees are tree structures that model both document structure (through so-called 'internal arc") and external references (through "external arcs" labeled with URLs). Hypertree document representation is data-oriented and "subsumes collections, nesting and ordering". Webs are a higher-level abstraction that model URL-hypertree pairs, implicitly defining document graphs. It is to be noted that while the proposed data model is well-suited for Web resources, the authors aimed also at covering other types of documents and relations more implicit than hyperlinks.

Web queries in WebOQL follow the select-from-where form well-known from SQL, and are based on six key operators and a few control structures. Prime operator returns the first subtree of its argument. Peek operator extracts a field from an argument by its label. Hang operator constructs a labeled arc between its two arguments. Concatenate operator makes it possible to merge two trees. Head operator returns one-node or one-edge trees from its argument, and tail operator returns the part of its argument remaining after removing its head.

Control structures proposed include iterating all items in a collection ("in" construct), iterating through interconnected hypertrees with internal and/or external arcs following specific patterns (specified as regular expressions) ("in ... via" construct) and Webs merging ("pipe" pseudo-operator). Moreover, WebOQL queries implement some comparison operators (including regular expressions pattern matching).

D.12 FLORID

FLORID [153, 198], developed at the University of Freiburg, is another logics-based Web query language (specifically using F-logic). The primary focus of the FLORID system is on formalizing Web navigation by using path expressions in a labeled-edges graph representation of hypertext navigation, and path-based queries (that work well also with loops in a graph, as in the case of pagination). However, it provides also elementary information extraction support by using regular expressions with subpatterns. As compared with WebLog or W3QS, FLORID is the more mature system that clearly defines queries semantics and their execution algorithms.

D.13 Jedi

Java based Extraction and Dissemination of Information (Jedi) [156] is a wrapper development language developed at the German National Research Center for Information Technology, that focuses on irregularities in Web sources and other textual files, as well as on easy integration with other Java packages.

The proposed wrapper language combines declarative data extraction with the procedural reconstruction of extracted data into a desired output format. The declarative extraction language is a special form of context-free grammar that allows ambiguities between different alternative patterns, underlining the importance of extraction based on expected data format ("content") rather than on varying and unstable delimiters ("context"). Whenever a string is matched against an extraction pattern, an associated procedural script is called. As Jedi is strictly Java-oriented, it is capable of using any Java methods and its default output consists of Java objects passed directly to other packages. However, the procedural language used enables also the generation of output in other structural formats such as XML.

D.14 NoDoSE

The Northwestern Document Structure Extractor (NoDoSE) [11] is an information extraction system developed at Northwestern University. While it supports extraction from Web documents, its primary focus is on plain text documents with nested substructures. However, we include NoDoSE in this chapter as it proposed an unique interactive wrapper construction and data extraction technique that did not require technical expertise.

Wrapper construction in NoDoSE starts with loading a single document from a collection. Next, the program allows a user to visually build a hierarchy of embedded data records or lists, mapped onto the existing data model by mouse text selection. At any moment of time a user may run the "mining" procedure that attempts to generalize a user's selections and extract more data from the same document (if they exist). Once the mining is run, the user can modify its output, thus improving the learned wrapper. Consecutively, other documents are loaded one-by-one and the wrapper is applied, always making user corrections possible.

D.15 Stalker

Stalker [214, 215] is a wrapper induction tool developed as a part of the Ariadne system [212, 169, 170] developer at the University of Southern California. Given a few annotated examples, Stalker is able to learn the automata-based wrappers for pages with hierarchical data organization, varying the order of attributes as well as optional attributes. Web pages in

Stalker are represented in an hierarchical way using two basic abstractions (tuples and lists), and embedding mechanism.

As compared with WIEN, Stalker has a few advantages. First of all, thanks to the application of alternative delimiter rules (non-deterministic automata), Stalker is able to work with optional attributes and attributes permutations. Secondly, the handling of hierarchical data is much more natural in the case of Stalker than in the case of WIEN's N-LR and N-HTLR rules. Finally, as Stalker operates at the level of tokens (including HTML tags) rather than characters, the learning procedure is more effective, without losing much of its generality.

The key disadvantage of Stalker is the way that multiple records are extracted and connected. For each attribute a separate automaton is developed, and multiple attributes are connected thanks to the hierarchical relations between different automata. As a result, to extract information n passes are needed, where n corresponds to a number of attributes.

D.16 SoftMealy

SoftMealy [155] is a wrapper learning system developed at Academia Sinica and Arizona State University. It supports supervised wrapper learning for semi-structured text documents (including HTML files), record structures, attributes permutation and missing values.

SoftMealy wrappers are expressed as finite state transducers. Their input alphabet consists of so-called separators. Individual attributes (including a special "dummy attribute", meaning parts of the document that should be skipped) correspond to transducer states. Finally, transducer output is composed of sequences of tokens from the original document, representing attributes values. Separators are zero-with points in the text that separate two attributes (or an attribute from a "dummy attribute") and are defined by regular patterns on their left and right side (called separator's context).

As compared with WIEN, SoftMealy adds support for attribute permutations and optional attributes (thanks to allowing multiple alternative paths through states in FST), and a more flexible approach to defining attribute boundaries (for example, it supports concatenated attributes with no delimiters between them). However, it does not support nested data, and the proposed learning algorithm has a limited support to overlapping contexts for multiple attributes. As compared to Stalker, SoftMealy is able extract multiple attributes in one pass; however, it is unable to handle hierarchical data.

D.17 Junglee VDB

In [228] Prasad and Rajaraman presented a commercial information extraction, integration and querying system (shortly called Junglee) supporting the SQL-based querying of multiple external and internal Web sources. While the description of used technologies is very selective and mostly focused on the system's architecture (as it is often in the case of commercial systems), the authors are among the first to touch some important development directions.

Firstly, Junglee promises to address a few important technical issues of Web sources wrappers, including the support of multiple protocols (including HTTPS), authentication, HTML forms and Cookies. Secondly, the proposed system implements not only extraction and navigation rules, but also rules for data transformation (into a unified data model), and data validation rules (to ensure integrity). Thirdly, it provides a few business applications of data extraction (in on-line recruitment and e-commerce), and describes a technical scenario of importing Web data into a data warehouse to enable OLAP analyses. Finally, Junglee brings into the field the idea of ideally abstracting wrappers by providing access to individual sources or integrated views over multiple sources via unified data access layers such as JDBC or ODBC.

D.18 WHIRL

Word-based Heterogeneous Information Retrieval Logic (WHIRL) [77, 79], developed at AT&T Labs, is an information integration system based on the innovative idea of so-called *similarity joins*. While it does not consider the problem of information extraction, it has an impact on related aspects of information integration and query rewriting for heterogeneous Web sources.

The proposed solution is based on the assumption that in many real-life cases it is impossible to construct a shared domain model with normalized names of entities of interest. As a result, the need for a more “soft” identification of the same entities in heterogeneous databases arises. WHIRL answers this need by introducing the notion of “similarity joins”, i.e. joins between relational tables (as in relational algebra) based on word-level similarity rather than the identity of chosen attributes. A robust algorithm is proposed to enable effective execution of conjunctive queries using “similarity joins”.

D.19 W4F

World-Wide Web Wrapper Factory (W4F) [33, 239, 240], developed at the University of Pennsylvania, was the first wrapper development system that took advantage of the structure of HTML documents, providing a complete, “multi-granularity” extraction language. It also provided a novel architecture of wrappers, as well as some visual support for wrapper construction.

Wrappers in the W4F system are composed of three distinct layers: the retrieval layer responsible for accessing Web resources (parametrized POST and GET requests are supported), the extraction layer performing actual data extraction, and the mapping layer responsible for generating output in specific formats. For each of the layers a Java code is automatically developed, enabling the seamless integration of wrappers into other applications.

Data extraction in W4F is performed using the HTML Extraction Language (HEL), a language that combines DOM tree traversal with regular expressions. HEL DOM traversal expressions are similar to (basic) expressions in today’s XPath language. They enable the full specification of path in DOM trees, starting from the root `<html>` element, with specific indexes provided for each child element (assumed to be “zero index” if omitted), specification of the first-descendant tag of specific type expressions (e.g. find the first depth-first ancestor

`<a>` tag of the body tag), as well as working with collections of individual tags (selected by choosing multiple indexes). Special tree branches txt, src and getAttr(attribute name) exist to access text, source HTML code, and any attribute of selected tags, respectively.

At a lower level of granularity, HEL uses PERL-compatible regular expressions, enabling the parsing of text nodes (actual content) selected by DOM-tree expressions. HEL supports two regular expression operations: match (with subpatterns) and split (each occurrence of the pattern is used as a values separator). They may be chained, i.e. the output of one operation may be used as the input of the following one.

Apart from DOM and regular expressions rules, W4F supports the WHERE clause that can define constraints on any other objects defined through similar expression and connected to extracted data by the use of shared unbound variables. The authors demonstrate, for example, how these capabilities can be used to extract data from a specific cell in a pivot table.

Finally, HEL supports the hierarchical composition of extraction rules, and record constructor operators that can be embedded one into another. HEL makes data available in a flexible NSL data model including null values, string values and (potentially embedded) lists.

The third layer of W4F is responsible for mapping extracted NSL data onto any output data structures. W4F allows any output constructors to be implemented based on Java interfaces and provides in-built support for building XML files or fully-typed Java objects.

Another contribution of W4F is a prototypical user interface that supports the construction of DOM-based extraction rules. While the proposed interface is only capable of providing complete ("canonical") paths in DOM-trees (with no support for generalization, record constructors or WHERE clause specification), the W4F approach clearly inspired a few future systems.

D.20 DEByE

Data Extraction By Examples (DEByE) [235, 181], developed at the Federal University of Minas Gerais (Brazil), is an information extraction system enabling intuitive wrapper construction by a non-expert user. The key novelty introduced by the authors of the DEByE system is its approach to user interaction. To build a wrapper, a user starts by choosing any exemplary Web page (belonging to some class of similar pages). Then, (s)he chooses some or all data objects present in the sources, and marks the values of his/her interest in a visual way. Finally, the user organizes the selected values manually into hierarchical data structures represented as "nested tables", i.e. records with embedded lists of sub-records of the same structure. By doing this, the user defines the expected output for some exemplary objects contained in exemplary pages.

Wrappers in DEByE are specified by using three traditional constructs: single values ("atoms"), records ("tuples") and lists[1]. Individual values are specified based on the notion of their left and right context.

[1]The fourth construct covering alternative record structures ("variant") in introduced is [181].

User specifications are automatically transformed into wrappers by using two wrapper induction algorithms. The top-down algorithm learns the complete extraction rules for complete structures corresponding to user design. It is applicable in the case of very regular structures; however it does not handle permutations and other structure irregularities. The bottom-up algorithm builds rules for individual values that are *a posteriori* combined into complete data objects. It handles different types of structure irregularities. However, it introduces a new problem of identifying the right objects for each extracted value. To solve this problem DEByE uses heuristics based on the position of attributes in HTML code; each occurrence of an attribute that appears as the first in a given level of nesting is treated as the marker of an object (or sub-object) boundary.

In [131] a further extension of DEByE (called Agent Specification By Example, shortly ASByE) is proposed for the example-based specification of navigation patterns (including support for form-filling). The key novelty of the proposed solution is the ability to generalize similar documents based on examples and a few heuristics (such as URLs similarity and their distance in referring pages).

DEByE provides a new wrapper construction paradigm and is one of the first that touch on the problem of the construction of navigational wrappers. However, as compared with previous work, it provides a very basic (and not very expressive) data extraction language, and relies on heuristics that may fail in more complex Web sites.

D.21 XWrap

XWarp [192] is a wrapper construction system developed at the Georgia Institute of Technology. It implements the user-friendly generation of compiled wrappers using declarative extraction rules based on tag trees (similar to DOM). Apart from the wrapper creation environment, it provides also automated self-diagnosis and adaptation facilities, as well as the support of releasing wrappers as software packages. The system was used in a number of other research projects, including the WebCQ [194] data monitoring solution, TAM workflow management system [291] and InfoSphere [229].

The XWrap approach to wrappers generation consists of four steps. The first, called by the authors "syntactical structure normalization", consists in fetching the document at a specific URL (XWrap supports templated GET and POST HTTP requests), cleaning bad HTML and parsing the document into a DOM-like tree structure. The second step corresponds to the user-friendly defining of extraction rules (via tree expressions using parent-child and ancestor-descendant relations) and mapping individual attributes and whole substructures onto output templates (XML files templates). The third step consists in the automated generation of compilable code based on defined declarative rules. Finally, the fourth step covers the automated testing of created wrapper against multiple pages (and the user-assisted adaptation of rules if testing fails) as well as support for building releases (ready-to-use software packages with documentation).

D.22 WebVCR

Developed at Bell Laboratories, WebVCR [21] is not an information extraction system. However, it provides an interesting approach to repetitive navigation in data-intensive Web sites.

The authors observe that it is impossible to bookmark pages in Web sites that rely on navigation, are stateful, require authentication or have dynamically changing URLs. To enable this, WebVCR introduces a "record-and-replay" approach to Web navigation in complex Web sites. Using a modified Web browser, a user records some interactions with a Web site. For each performed action (such as form filling or clicking a link), multiple features are saved including the exact URL, anchor text and DOM location of a given element. During replay, all this information and some heuristics are used to deal with changes in the Web site.

D.23 Lixto

Lixto [41, 42, 40], developed at the Technical University of Wien, is the first information extraction system that combines an intuitive, visual wrapper construction by non-expert users with a highly expressive logic-based data extraction language, which uses DOM-based rules, string-based rules and support for some basic data formats and semantic classes. Moreover, Lixto was also further extended with the support of navigation and Deep Web sources [39].

User interaction with Lixto is somehow similar to that of DEByE: after loading a simple document selected by a user, the user marks fragments of a page text and maps them onto some arbitrarily created hierarchical information structure. Next, a candidate wrapper is generated and applied to other similar documents. Afterwards, the user is asked to validate the extraction result, and in case it is not satisfactory, to mark values that should not be extracted.

User choices are generalized to wrappers expressed in a logic-based language called Elog (based on monadic Datalog as discussed in [134]). Extraction rules are expressed in a declarative way, using a number of in-built predicates stating, for example, that a document is loaded from a specific URL (`document` predicate), that an element can be accessed by a specific DOM tree traversal (`subsq` and `subelem` predicates), that a value is a match of a regular expression within a specific element (`subtext` predicate), that a string is a value of a specific attribute of a specific element (`subatt` predicate) or that an element is located (not) before/after an other element (`before, after, notBefore, notAfter` predicates). Extracted values are mapped onto an output format by using predicates with names corresponding to the names of a specific tuple or a simple object. Moreover, Lixto implements some format validation predicates, such as `isCountry, isCurrency, isDate`.

It is to be noted that Elog programs supported recursive rules and link-based navigation. However, it was only later [39] that the initial real solution for this problem was presented. In the Lixto approach, Web documents are organized into classes. Each class shares the same extraction rules and similar actions (such as link following or form filling) that can be performed in the pages. Each navigation element is defined by an XPath expression (surprisingly not an Elog rule), thus connecting two classes of pages by a class of action. In the same paper a draft of a visually assisted method of defining navigation wrappers (with no support for recursive rules i.e. loops in navigation) was presented, and a few interesting directions for improving the technical quality of extraction (e.g. by wrapping a Web browser control in order to solve JavaScript, Cookies, and session management issues). However, the final results of these developments stayed unpublished - probably because Lixto has become a commercial system.

D.24 Hidden Web Exposer (HiWE)

Hidden Web Exposer [232], developed at Stanford University, focuses on crawling Deep Web sources, mostly for the purpose of their indexing. The central problem that this system addresses is related to filling in forms, aiming at wide (yet not exhaustive) access to Hidden Web documents. The system is composed of four main components: 1) a task-specific fuzzy database, 2) an internal form representation, 3) a matching function, and 4) a response analysis module. The task-specific database assigns to each label L a fuzzy set V of values with an associated membership function. It is initially filled with built-in entries such as dates or day of week values, as well as with data gathered from wrapped Web or database sources. Subsequently, the database strings and membership function values are updated using finite-domain form elements (such as grouped radio elements or select element options).

When a new HTML form is discovered, it is transformed to the internal form representation: firstly it is parsed, then fields labels are discovered based on visual distance and normalized by using simple lexical techniques. At the next step the matching function based on string distance metrics is used to match normalized form labels against database labels. Finally, 100 sets of values with the best overall fit (based on membership function) are found and used to fill in the form and gather the resulting pages.

HiWE has only a limited scope of data extraction, using a proprietary approach called the Layout-based Information Extraction Technique (LITE) [231]. Information extraction is used by HiWE in two contexts: firstly, it is used to identify form labels (based on their adjacency to form fields); secondly, the extraction of the so-called “significant part” of the page (the center-most frame or layout element) is performed to enable result analysis. Information extraction in the LITE approach combined a basic tags nesting (DOM) hierarchy and layout information (obtained from specific rendering engine) to identify significant information.

An important problem in indexing Deep Web contents concerns invalid or no-results queries. Pages that contain no significant information (e.g. an error message or empty resultset) should not be indexed. In HiWE such pages are detected using two methods, both based on the analysis of the significant part of a Web page (extracted using LITE) [231]. The first of them uses an explicit list of typical error messages returned by Web sites. The second one applies a hash function on the significant part of each Web page. The hashes obtained for different pages are compared, and repeating values are used to detect no-results pages.

D.25 Omini

Omini [61], developed at the Georgia Institute of Technology, is probably the first system ever developed that presented the wrapperless approach to Web information extraction. Its aim was to enable completely the automated extraction of objects (records) from unseen Web pages containing multiple data records.

Extraction in Omini is performed in three steps: preparing the document, locating objects of interest and extracting objects. The first of them covers acquiring and parsing the requested document, correcting HTML syntax errors and representing it as a DOM-like tree.

The second step covers two activities: discovery of the "object-rich subtree" in the tags tree, and extraction of the object separators. The object-rich subtree, i.e. a tag tree subtree that contains the actual content of a Web page, is extracted based on a set of heuristics, including content size and subtree size. The discovery of ordered lists of candidate separators is done by combining multiple methods, including the measurement of standard deviation between the pairs of occurrences of candidate separators, looking for repeating pairs of tags with no text between them, using a dictionary of common separator tags, counting the occurrences of all sibling pairs of tags, and finding often repeating tag tree path fragments.

Actual object extraction consists in breaking down the content of object-rich subtrees by the occurrences of chosen separator(s) and removing too small and too large candidate objects as well as objects that are structurally too different from other ones. Unfortunately, little information is provided on the exact method of eliminating these false-positive matches.

D.26 IEPAD

IEPAD (acronym for Information Extraction based on PAttern Discovery) [244, 68, 65] is a wrapper learning and data extraction system developed at the National Central University (Taiwan). It enables the fully automated creation of extraction rules for pages with multiple occurrences of similar patterns (such as search engine result pages) based on the discovery of repeating patterns.

In IEPAD documents are represented as sequences of tokens corresponding to opening and closing HTML tags as well as special "text" tokens corresponding to any string contained between HTML tags. Optionally, some HTML tags with similar meaning may be translated into the same token value corresponding to some semantic class of tags. A page represented as sequence of tokens is further encoded as a special index data structure called a PAT-tree (for Practical Algorithm to Retrieve Information Coded in Alphanumeric), which enables the quick discovery of maximal sequences of repeating patterns. Once repeating patterns are discovered, they can be transformed into extraction rules in a very simple way. The algorithm is further extended to handle approximate pattern matching that is able to discover patterns with optional attributes.

The key novelty of IEPAD, as compared with previous work, is its ability to completely automate wrapper generation for a specific class of problems. Like WIEN, Stalker or Soft-Mealy, IEPAD is a string-based algorithm; while it uses some knowledge of HTML, it does not exploit DOM tree structure. The main drawbacks of the proposed solution are its inability to extract data based on delimiters other than HTML tags, and the fact that it often produces multiple patterns (rules), making it hard to choose the right ones.

D.27 EXALG

EXALG is a data extraction algorithm developed at Stanford University by Arvind Arasu and Hector Garcia-Molina [23, 22]. It aims at the fully automated extraction of information based on template and schema deduction using multiple Web pages from the same Web site. As the method is automated, the schema elements are not named or interpreted. Moreover, while they correspond well to the page template, out of necessity they may be of different granularity than in the case of manually created schema (e.g. two logical attributes separated only by a space will be treated as one field).

One of the central points of the EXALG method is related to the proper representation of the schema and template. In EXALG, the schema, which is the deduced structure of extracted data, basically consists of simple type fields as well as of two constructs enabling hierarchical composition: sets (containing any number of elements of the same type) and lists (defining the order of specific sub-components). It is supposed that a data schema is transformed into a page by using a template, which is an infix notation imposed on each schema and subschema. Templates in EXALG implement two basic types of constructors, corresponding to two ways of schema hierarchical composition: tuple constructor and set constructor. The former, corresponding to schema lists, is called tuple and consists of separate constants for beginning (prefix), each space between two attributes (infixes) and the end (postfix). The latter, corresponding to schema sets, contains a single infix used between all elements of the data schema set. Two other useful constructors, which are optionals and disjunctions (i.e. alternative values), are emulated using basic constructors with cardinality constraints (0-1 in case of optionals, 0-1 for both alternatives in disjunctions, with the additional constraint that at least one of the constants must be present).

The ultimate objective of EXALG is the automatic discovery of schema from Web sites. The proposed learning algorithm requires multiple pages to find patterns. It is based on the idea of the equivalence classes of tokens, i.e. sets of tokens repeating given a number of times in a specific page (e.g. a set of all tokens that repeat exactly three times in a given page). The basic assumption adopted by the authors is that a set of tokens forms an equivalence class in multiple pages (probably with a different number of repetitions, e.g. ten repetitions in the first result page and three repetitions in the last result page) are probably constants of template constructors. After the equivalence classes are discovered, they are used (together with their nesting) to discover templates from multiple pages in an automated way. To ignore repetitions based on coincidence, only equivalence classes with consistent ordering and nesting are taken into consideration. As the same token may have two roles (belong to two separate token constructors), the extra notion of token-role is introduced. Token-roles are equal when both their value and DOM path are equal. Moreover, some range consistency is also checked (i.e. no two sets of occurrences of the same token-roles may overlap).

D.28 WL^2

WhizBang Labs' Wrapper Learner (WL^2) [80], developed by WhizBang Labs, is a modular wrapper learning system capable of combining different extraction features and languages corresponding to different representations of HTML documents.

The key contribution of WL^2 authors to the data extraction field is a proposal of a single generalized learning algorithm that can work with different types of extraction features. The only requirement for the algorithm to work is that for each extraction language two operations (specialized algorithms) are defined. One is a computation of the least general generalization, i.e. the most specific rule that covers all positive learning examples. The second operation is a refinement that finds a set of such rules that each of them covers all but a subset of positive learning examples.

The initial version of the proposed algorithm covers rules expressed in a single language and is demonstrated (but not limited to) using token-context rules for token-based document representation and XPath-like extraction rules for DOM-based representation. However, the proposal is further extended with ideas of the compositions and conjunctions of any languages. Moreover, an approach based on the geometrical representation of Web pages is proposed and its usefulness for complex tables embedded in Web pages (excluding only pivot tables) is demonstrated.

The proposed solution is based on FOIL logic rules learning, combined with a few pragmatic pre-processing steps, such as dirty HTML removal and formatting tags normalization.

In WL^2, rules are learned via user interaction. To minimize wrapper definition time, users are not required to mark all positive examples in a page, and can choose to mark only all positive examples within some selected page region.

D.29 RoadRunner

Developed at the Universita di Roma Tre and Universita della Basilicata, RoadRunner [208, 83, 205, 84] was a system focused on the automated generation of wrappers for HTML content. Given unlabeled, positive examples of pages from one class (i.e. "generated by the same script"), RoadRunner finds regular expressions wrappers capable of working with complex nested structures (lists, tuples, optional values).

The learning algorithm is based on the iterative, token-level comparison of the current wrapper and one additional sample page in order to generate a refined wrapper. The key method of the algorithm is analysis of the mismatches between wrapper and page, which are used to discover text fields (in the case of string mismatches) and optional or iterator substructures (in the case of tag mismatches). Depending on the type of page template, RoadRunner uses the Approximate Matching Technique (when tag trees are regular near their roots with low-level irregularities) or the Partial Matching Technique (when similar low-level tag sub-trees occur in rather unstable tag trees).

As RoadRunner treats documents as streams of tokens, including both tags and string values, it is able to capture both the tag-based and string-markers-based delimiters of individual attributes. The proposed algorithm is fully automated, domain-independent and able to capture different relatively complex structures. At the same time, it is relatively sensitive to HTML code irregularities, and provides data extraction results with no associated schema (data types, attribute names) and with possibly different granularity than expected.

D.30 Denodo Integration Platform

Developed at the University of A Coruna, and commercialized by Denodo Technologies, the Denodo Integration Platform [219] is a Web-oriented information integration system addressing the multiple challenges of complex data-intensive Web sites, including client-side dynamic [16], Deep Web crawling [17, 18] and wrapper maintenance [19, 234].

The platform, as compared with the previous work, has a few novelties. Firstly, it introduces a simple yet highly expressive method of specifying the limited query capabilities of Web sources query interfaces, making it possible to specify all combinations of fields that can be queried and/or that must be provided to any query by a conjunction of positive and negative predicates [218].

Secondly, Denodo Platform adopts a novel technical approach to Web navigation; instead of using HTTP-level libraries for issuing HTTP requests, it relies on an embedded Microsoft Internet Explorer library with navigation scripting possible using the procedural Denodo NSEQL navigation language [219, 15]. While basing navigation on an embedded Web browser may have efficiency drawbacks, it surely resolves many technical problems with HTTP redirects, Cookies, SSL and most cases of JavaScript use.

Thirdly, Denodo provides support for the parallel execution of multiple requests with multiple instances of their DeepBots [18]. While in the described scenario they are used for Web pages indexing, the presented infrastructure is the first step to providing large-scale information extraction from data-intensive Web sites.

The next component of the Denodo Platform deals with filling in Deep Web forms based on a domain model. Forms parsing uses a combination of visual distance between fields and labels that exist in HTML form, and text distance (using TFIDF and Jaro-Winkler metrics) between form labels and domain attributes' names [18].

Another contribution of researchers from A Coruna University is in the area of wrapper maintenance, i.e. the automated adaptation of wrappers to changing data sources [19, 234]. The basic idea behind the proposed algorithm is to track historically successful pairs of query and extracted data records, so that this information can be reused when the wrapper fails. Once the wrapper ceases to return results, a number of previously successful queries returning multiple results are selected and reissued to the sources. Next, the result pages are compared with previous result data, and good matches (i.e. data identical or similar to the previously acquired) are used as labeled positive examples for learning the wrapper for the new Web page template.

The final novelty of the Denodo Integration Platform is its explicit query planning respecting Web sites querying and navigation capabilities, and cost-centric query plan execution based on historical statistics (as in major database systems) [219].

D.31 DeLa

Data Extraction and Label Assignment (DeLa) [197] is a Deep Web data extraction system that focuses on the automated schema discovery of Deep Web sites. The proposed solution is composed of four components: a form crawler (using the previously discussed HiWe solution), a wrapper generator, a data aligner and a label assigner.

The wrapper generator in DeLa interprets each page as a long string, and learns regular expressions generalization of their content. Wrapper learning starts with "data-rich section extraction", i.e. with the removal of the non-data-rich parts of the page (such as menus or advertisements) by comparing two pages belonging to the same class. Then, repeating structures and their nesting are discovered and a few string-based and DOM-based heuristics are used during wrapper learning for pruning some candidate wrappers. The output of the wrapper generation phase are regular expressions with support for optional values, nested types (represented as nested subpatterns) and disjunctions (represented as alternative patterns).

In the data alignment phase, nested data structures are extracted by applying wrapper regular expressions; they are represented as star-schema-based data tables, corresponding to data nesting and disjunctions. Furthermore, a simple heuristic is applied to find boundaries between multiple attributes stored in a single HTML tag.

The final phase of schema discovery is related to assigning labels to the attributes of extracted data. To attain this objective, four heuristics are applied. The first is based on reusing form labels extracted by HiWe, if a specific attribute value corresponds to a value in a submitted query. The second is based on the analysis of table header (TH) tags of HTML code. The third looks for attributes names that precede attribute value in each of its occurrences. The final heuristic uses some predefined data formats (such as dates, e-mails or prices) to assign labels if previous methods failed.

D.32 Cameleon

Cameleon [108] [107], developed as a PhD project at the MIT Sloan School of Management, is a complete wrapper development system with wrappers based on regular expressions, queries implemented in the SQL language and output HTML and XML formats.

Wrappers in Cameleon are represented as a set of separate extraction rules, one per attribute. Each of them consists of markers of the beginning and end of the potentially interesting parts of a page, as well as of one or multiple alternative data extraction patterns (using regular expressions). While the simplicity of the proposed extraction language makes wrapper construction easier, it does not work well with any complex data structures, including optional attributes, nested structures, and even (when a Web page is not very regular) merging individual attributes into data records. It is also to be noted that an extra mechanism for rules disjunctions is redundant as alternatives are well handled by most regular expressions implementations.

One of Cameleon's novelties is its ability to represent navigational wrappers in such a way, that values extracted in a specific navigation step are further reused, enabling, for example, the dynamic construction of URLs that contain session identifiers. Cameleon supports also POST

HTTP method, HTTP authentication, Cookies, JavaScript (to some extent) and delayed extraction (a simple heuristic handling content generated in an `onload` event).

D.33 VIPS

While not being a pure data extraction algorithm, VIsion-based Page Segmentation (VIPS) [62], developed at Microsoft Research Asia, presents an innovative approach to Web page analysis that can be applicable also in data extraction.

VIPS performs hierarchical Web pages segmentation based on its DOM structure and visual features. It uses two key properties of content blocs: its internal degree of coherence (DoC), and external visual separators. VIPS uses features such as tag separators (e.g. HR tags), background color variations, text coherence and child element size standard deviation to identify candidate Web page blocks. Next, a set of heuristics is used to assign the 0 to 1 value of DoC to each block, measuring its internal "coherence". Afterwards, the page is analyzed for occurrences of separators, i.e. "horizontal or vertical lines in a web page that visually cross with no blocks" (white space between blocks). Each separator is assigned a weight based on its size, co-occurrence with HR tags, font and background color differences between separated blocks, as well as the structural similarity of separated blocks. Finally, DoCs and separators weight are combined to hierarchically partition the page into semantically coherent content blocks. While the VIPS solution may not be very elegant, it proved to perform well, and paved the way for some more complex vision-based Web page analysis methods.

D.34 DEPTA

Data Extraction based Partial Tree Alignment (DEPTA) [284] is a fully automated approach to the extraction of data records developed at the University of Illinois. In contrast with many previous works it does not aim at automated wrapper construction, but rather the fully automated extraction of data records from a single unseen HTML document based on DOM structure and the visual similarity between records.

Data extraction in DEPTA is performed in three phases: building a tree representation of a Web page, mining data regions, and identifying data records. The tree representation of a Web page is built in one of two possible ways: by reusing DOM tree structure [190] or by checking the nesting of Web browser boxes corresponding to individual HTML tags [284].

The mining data regions step consists in finding adjacent (sequences of) subtrees (called generalized nodes) of the same size that are structurally similar to one another. To limit the number of similarity measurements, a key assumption that consecutive data records are embedded in a shared parent node is adopted. Similarity between generalized nodes is calculated by applying Levenstein edit-distance on the source HTML code corresponding to them. Moreover, some false positives of complex repeating structures are removed by an assumption that visual gaps between multiple records should be larger than those within a record.

Identification of data records is itself composed of two steps. The first consists in rearranging generalized nodes into a set of trees with one tree corresponding to each data record.

It enables DEPTA to work with records that are not contiguous in terms of HTML source (as in the case of records organized into HTML table columns that are interwoven in an HTML source). The second step, which is much more complex, consists in applying a tree alignment technique in order to map corresponding values onto data attributes.

D.35 IDE

Instance-based Data Extraction (IDE) [283], another system developed by the same team that developed DEPTA at the Georgia Institute of Technology, approaches a significantly different subproblem of information extraction. While DEPTA focused on completely automated extraction from multi-record pages, IDE proposes an interactive data extraction cycle for pages containing single instances of data of a user's interest.

The key technique used in IDE, called instance based learning, consists in asking a user to label a single Web page only if it can be extracted by none of the previously learned rules. While working with a new collection of Web pages, when no extraction rule has been yet created a random page is chosen for labeling. Next, the system tries to extract data from consecutive pages by reusing previously identified extraction rules. If it is not possible, the user is asked to label another page and another extraction rule is added to the list.

Extraction rules in IDE are expressed as the shortest uniquely identifying token prefix and suffix of data to be extracted. A separate rule is created for each attribute and the problem of connecting attribute values to data objects is avoided, as the existence of a single object per page is assumed. Labeling is performed in an intuitive user interface.

Actual data extraction is performed by using prefixes and suffixes identified in labeled pages. However, as differences may exist between different page suffixes, IDE allows that the actual suffix or prefix in the page is incomplete if it still identifies a single value in the page. If none of the existing prefix-suffix combinations work, the user is asked to label the current page.

D.36 ViNTs

The Visual information aNd Tag structure based wrapper generator (ViNTs) [233] and its extensions [289, 290], developed at the State University of New York and University of Illinois, are an automated wrapper learning system for search engines result pages.

The solution consists of three major components responsible for: section extraction [289], record extraction [233] and record annotation [290]. Section extraction consists in finding one or more record-rich areas of the page (potentially corresponding to different types of results). Record extraction extracts HTML blocks corresponding to individual data records. Records annotation splits each of these blocks into fields. Both section extraction and record extraction combine a number of visual (geometrical) and tag-based heuristics, including the classification of types of content lines with respect to their tags, shape and size analysis of HTML blocks, the horizontal position of individual elements, a few separator detection methods, and the discovery of the common part of the tag tree path of different elements.

In record annotation, authors apply, instead of previously used alignment techniques, a set of heuristics removing "decorative tags" and detecting basic data types with hierarchical statistical clustering for connecting similar values and neural networks for discovery of the "template text decorators" (fragments of text that belong to the template rather than to the values to be extracted). Even if the wide set of used technologies makes the impression that the proposed solution is not entirely coherent, empirical experiments show that it is highly robust and effective.

D.37 Thresher

Thresher [154], developed at Google and MIT, is a browser-based wrapper creation and data extraction tool that maps extracted data onto Semantic Web (RDF) objects.

Wrappers in Thresher are learned based on a single positive example marked by a user, by using relatively simple DOM-base features, and can be extended by adding another object to the wrapper, if the wrapper does not properly recognize it.

What is new in Thresher's approach, is that extracted objects (and optionally some of its attributes) are mapped onto RDF ontology classes, thus becoming semantically interpretable. Another novelty is that the exemplary data are labeled during normal Web browsing activities by the user in his/her Web browser, and that wrappers are automatically applied to other visited pages, allowing user to perform various operations on identified objects. Thus, we can perceive Thresher developers as proponents of permanent information extraction during Web browsing, making it a hybrid information extraction and personal information management system.

D.38 DWDI

Deep Web Data Integration (DWDI) [165], developed at the Poznań University of Economics, is a navigational information extraction system focused on Deep Web information extraction.

The key component of the DWDI system is a navigation modeling language based on Finite State Machines, which covers the navigational paths in a Web site. In DWDI, automaton input characters correspond to generalizations of HTTP requests (represented as regular expressions), and automaton states correspond to generalizations of Web pages templates (with associated extraction rules). Whenever in some navigation state a link is followed or a form is filled in, it is interpreted as an input character of the automaton's input alphabet and automaton transition is performed, defining new navigation state and associated data extraction rules.

In the DWDI model, paginated results are represented by using loops from an automaton state to itself. In this setting the input symbol of such looped transition corresponds to the action of following a link to the next page of results.

During Web site navigation, at each automaton state two actions are performed. Firstly, if the definition of a given state has any associated extraction rules, they are performed and extracted values are stored in the current navigation path. Secondly, all links and forms consistent with symbols accepted in the given state of the automaton are extracted. Each of them forms a new navigation path consistent with the automaton definition, by extending the current path. All created paths are added to the queue of paths to be processed.

Thanks to using the finite state automata model, the complete path of previous HTTP requests (starting in the automaton start state) is used to interpret each sent request. As a result, DWDI works well if content served by a specific URL depends on previous requests. Moreover, each time a path is popped from the queue, DWDI reissues all HTTP requests corresponding to the path, starting at the automaton start state. While this causes a significant overhead (some requests are issued multiple times), it handles a significant part of stateful Web sites.

It is to be noted that DWDI navigation paths are used not only to enable interpretation of the current automaton state and to recreate the Web site's navigation state. It is also along the navigation paths that the extracted data are gathered. Thus, each path forms a separate data record by connecting values extracted in several related pages.

D.39 SSF

The Sub-Structures Finder (SSF) [112] developed by the author of this thesis focuses on automatically extracting similar structures from pages containing multiple data objects. Clusters of similar objects are built in a bottom-up way by an iterative expansion of initial one-tag objects to their sibling or parent nodes, performed as long as a cluster's internal similarity remains above some threshold. Document representation is based on a DOM tree with some tags filtered out by one of three filtering approaches, and with tags standardized via one of a few supported tags equivalence strategies. Inter-cluster incoherence is measured by string edit distance and tree edit distance.

The discovery of similar structures was performed in three steps: document parsing (dirty HTML removal, document parsing, tags filtering and tags equivalence strategy application), maximal ranges finding (finding the largest possible structures that still meet in-cluster coherence criteria) and aggregated clusters removal (the removal of clusters that overlap with other ones but contain larger items). While this approach yielded good results, the proposed solution did not extract individual attributes, nor did generalize extracted data into extraction rules.

D.40 Turbo

Turbo [75, 74], developed at the University of Illinois, is a completely new approach to a large scale induction of wrappers for multiple sources for a single domain (called context-aware or collaborative by the authors). In the Turbo approach multiple wrappers can be trained or improved iteratively by reusing the results of other wrappers or an existing data model (if provided). The authors demonstrate that using multiple data sources in an iterative improvement process can effectively split, merge or disambiguate attributes boundaries in multiple data sources.

The proposed solution is based on the Turbo codes from communication theory, and combines two probabilistic frameworks: Hidden Markov Models for modeling pairs of attribute labels and attribute values, and Expectation-Maximization for the iterative improvement of wrappers. The authors demonstrate that a few usage scenarios are possible: starting with an existing data model, starting with an existing data model and some initial wrappers, or starting with just an initial set of wrappers. They also show that even if RoadRunner, ExAlg and IEPAD-like automatic wrapper construction algorithms differ with respect to efficiency when applied to individual sources, they prove almost equally efficient when integrated into the Turbo framework. Moreover, the utility of even very imprecise initial wrappers (created by the heuristic of treating HTML blank line tags as fields separators) can work efficiently with Turbo.

D.41 Senellart et al. Approach

In [243] a complete system capable of learning the information extraction wrappers for Deep Web sources in a single domain is proposed. The proposed methods are completely automated and apply a few techniques (including Deep Web sources probing, and extraction rules learning that combines lexical and structural features in a probabilistic framework), by using some pre-existing domain knowledge (mined from some domain databases).

Wrapper learning in the proposed solution starts with pre-processing some domain database (the DBLP XML file is used as an example) to build a list of concepts and a statistical (word-frequency-based) instances model. The next step consists in structural analysis of Web form. In this step form fields and their potential labels are identified and probabilistic hypotheses relating these fields to domain model attributes are formulated.

These hypotheses are verified by the step of Deep Web source probing. Multiple domain model instances are fed into the Web site form according to the hypothesized field correspondence between model and form fields. The result pages are compared with results of non-sense queries (i.e. queries using non-existent words) by clustering with the use of TF/IDF vector space over all DOM tree paths found in documents. Result pages that end up in the same cluster as result pages to non-sense queries are considered failures. If many failures happen for some hypothesized attributes correspondence, other hypotheses are tested.

During the next phase, which is the learning of extraction rules, the domain knowledge is exploited again, even before Web page structure is analyzed. A gazetteer built from domain instances is used, together with a few heuristics for filtering out false positives, to mark examples of domain data in Web pages. Next, these examples, marked along multiple pages in the same cluster (i.e. result pages with similar structures), are used as positive examples for a supervised learning model based on conditional random fields over tag-tree paths.

In the final step, the data extraction model trained before is wrapped as a WSDL Web Service, so that it can be easily integrated into other applications.

D.42 DEQUE / I-Crawler

Shestakov, in his PhD thesis [248], presents a Deep Web access system composed of two key components. The first of them, DEQUE, is a Deep Web sources querying system. The second one, i-Crawler, is capable of discovering Deep Web sources.

The Deep Web Query System (DEQUE), first described in [249], is a system focused on querying complex Deep Web forms. It addresses a number of problems related to Deep Web sources, including Web form parsing, the handling of multi-page forms (among other things, forms that vary depending on previously selected values), automating querying a single form for multiple alternative values (i.e. unions over queries results) and posing queries to multiple Web sources (including the sequential composition of a few sources) with an SQL-alike query language.

Form parsing and label assignment to form fields in DEQUE is based on a set of heuristics that combine a few visual features. DEQUE searches are for JavaScript code that is connected to form events, so that it can be reused in form filling. Once a form is parsed, it is stored in a database for efficient use in query time.

DEQUE is capable of discovering pagination links in form search results. Two consecutive result pages are used for training data extraction wrappers using a variant of the RoadRunner algorithm. For convenience, the extracted attributes can be manually assigned names.

The querying of Web forms in DEQUE is done with the Deep Web Query Language (DEQUEL). Based on the preliminary SQL syntax, DEQUE allows user to query individual forms with two types of conditions: WHERE conditions that explicitly provide values for forms fields, and CONDITION conditions that impose constraints validated after data extraction is performed (in the post-filtering phase). An important feature of DEQUEL is the possibility of assigning multiple alternative values to a form field in the WHERE condition, so that a union of multiple form submissions can be obtained in a simple way. DEQUEL also supports using value from other data sources (including relational tables) as input of the possible values of a form field.

The second component of Shestakov's solution, I-Crawler, is a Deep Web source discovery tool. Given a set of Web pages, it performs three main tasks. First, it finds pages that contain HTML forms (including forms with the JavaScript submission mechanism).

In the second step, it uses some heuristics to reject some part of forms that are clearly not search Web forms, and then applies two classifiers to split forms into non-search forms, forms to unstructured, keyword based search facilities, and forms for structured Deep Web databases.

The final step consists in the classification of Deep Web sources into a predefined set of categories. To this end, a separate approach, based on probing of the Web source, is proposed for unstructured sources, and another approach, based on the analysis of form fields and SELECT control values, is applied in the case of structured sources.

D.43 Other Systems

In previous sections we described the most seminal works in the area of Web information extraction. In this section we gather for interested readers a few less important Web information extraction tools, as well as systems in the related domains of information extraction from text, Web information management, and Web query languages, that impact on or make use of Web information extraction technologies.

System	Affiliations	Comment
RAW [105]	U. Mannheim	Relational querying of Web pages
FAQ Miner [161]	National Taiwan U.	Template-based extraction from FAQ documents
Strudel [104]	AT&T Labs, INRIA Roquencourt, U. of Washington	Complex Web data restructuring
NetQL [195]	U. of Regina	Navigational Web query language
XML-QL [91]	U. of Pennsylvania, AT&T Labs, INRIA, U. of Washington	XML extraction and restructuring language
Lorel [2]	Stanford U.	Query language transforming TSIMMIS Lore objects into XML
X-tract [4]	U. de las Américas, Mexico	Extraction from botanical Web documents
WHISK [254]	U. of Washington	Learning extraction rules for free text and semi-structured documents
CONQUER [193]	Oregon Graduate Institute of Science and Technology	Web monitoring based on continuous queries over Web content
BWI [124]	Just Research, U. College Dublin	Induction of text-delimiter-based wrappers
MIA [49]	U. Koblenz	Multi-agent mobile system with information extraction support
UnQL [60]	U. of Pennsylvania, AT&T Labs	Web to XML query language based on recursion
SRV [123]	Just Research	Logic-based extraction from unstructured and semi-structured documents
[279]	Hanyang U., ETRI-Computer & Software Technology Laboratory	Joint information extraction and ontology building for comparison shopping
Ariadne [169]	U. of Southern California	Mediator and wrapper-based querying of the Web
[210]	U. of Southern California	Automated extraction of record-based data from HTML pages
MOMIS [46]	U. di Modena e Reggio Emilia, U. di Milano	Heterogeneous sources integration using conceptual (semantic) modeling
[187]	U. of Memphis	Deep Web sources selection for specific queries
ChangeDetector [54]	WhizBang! Labs	Web pages monitoring for changes based on entity recognition
DRESS [70]	MSRA, U. of Science & Technology of China	HTML-based visual segmentation of Web pages
[251]	Carnegie Mellon U.	Result merging and reranking from multiple sources in meta-search
[81]	Australian National U., CSIRO Mathematical and Information Sciences	Discovery of (Deep) Web search interfaces
QProber [157]	Columbia U.	Classification of Deep Web text databases
PickUp [71]	Mississippi State U., Wayne State U.	Wrapper generation based on tables analysis in biological domain

System	Affiliations	Comment
WISE-Integrator [282]	U. of Illinois, State U. of New York, U. of Louisiana	Matching multiple (Deep) Web query interfaces
[236]	U. of Essex	Automated wrapper generation based on repeating tagSets
[280]	Northeastern U.	Query rewriting for Web interfaces using previously extracted data
[47]	Xerox Research Centre Europe	Discovery of operators and syntactic constraints of Deep Web query interfaces
WIC [220]	Carnegie Mellon U.	Statistical models for Web pages monitoring for changes
OLERA [67]	National Central U., TrendMicro	Interactive, GUI-based Web data extraction
[101]	Brigham Young U.	Information extraction from Web tables using extraction ontologies
[285]	MSRA, Tshingua U.	Visual extraction, 2D conditional random fields
ContentExtractor [90]	Pennsylvania State U.	Feature-based automated extraction of informative Web page blocks
Meta-Querier [151]	U. of Illinois	Meta-search over Deep Web sources
[260]	Hokkaido U.	Interactive wrapper creation, on-the-fly preview
[99]	Mansoura U. (Egypt)	Deep Web crawling framework
[158]	Polytechnic U. at Brooklyn	Multi-feature, interactive, utility-driven wrapper learning
[188]	Thingua U., China	Classification-based approach to identify objects in Web pages
TQA [13]	Emory U., MSR	Web tables mining for question answering
WDE [223]	U. of Calgary	Automated data extraction based on weighted similarity of DOM subtrees
ACHE [122]	U. of Utah	Adaptive Deep Web sources discovery and classification framework
iMacros[2]	iOpus	Commercial solution with basic support of complex Web applications
[128]	Vienna U. of Technology	Visual Web information extraction
[20]	New Jersey Institute of Technology, U. of Michigan, City U. of New York	Automated ontology population from Deep Web sources
Dewex [164]	U. of Illinois	Deep Web sources database and source-centric query facility
PDFWrap [103]	U. della Calabria, Consiglio Nazionale delle Ricerche	Wrapper generation for PFD files
SE^3 [9]	Poznan U. of Economics	Semantically annotated navigation models for Deep Web sources
SNPMiner [127]	Ohio State U., Kent State U.	Domain-specific search tool for Deep Web
[271]	Albert-Ludwigs U. Freiburg	By-example Deep Web navigation with some support for AJAX content loading
Transcendence [51]	U. of Washington	Deep Web personalization
[138]	Yahoo Labs	Joint extraction from multiple Web sites

Table D.1: Other Work Related to Web Information Extraction Tasks

[2]See: `http://wiki.imacros.net/`, accessed on 18.07.2010.

Appendix E

Combining CsExWB and FiddlerCore Events

During any HTTP request a number of events are triggered both in the Web browser and in the Fiddler proxy. We list and describe these events in the following table.

Component	Event	(1)	(2)	(3)	(4)	(5)	(6)
Description							
CsExWB	`BeforeNavigate2`	●	●	○	○	○	○
(IE Window or frame event) Event triggered when navigation (loading new complete document) is about to happen top-level or in a frame. It gives access to the request URL, POST data (in the case of POST request), and identification of the target frame name and information if it is a top-level request. It also provides a reference to the Web browser corresponding to the affected frame or top-level window. Finally, it makes it possible to cancel navigation (e.g. in the case of capture request nodes).							
CsExWB	`ProtocolHandlerBeginTransaction`	●	●	●	●	○	○
(HTTP(S) protocol handler event) Event triggered in client-side HTTP/HTTPS protocol handler. It gives access to the request URL and some part of the request headers (including: HTTP method, version and URI and Accept, Accept-Encoding and Accept-Language headers). It makes it possible to cancel the request or to add extra headers that will be sent to the server.							
FiddlerCore	`BeforeRequest`	●	●	●	●	○	●
Event triggered at Fiddler proxy. It gives access to full request headers, including all headers previously available at ProtocolHandlerBeginTransaction, all extra headers set in that event, and the User-Agent header added by the Web browser.							
FiddlerCore	`ResponseHeadersAvailable`	●	●	●	●	○	●
Event triggered at the moment when Fiddler proxy receives the response headers from the HTTP server. It makes it possible to continue downloading content from the HTTP server, but also to modify response headers (e.g. to return an error code or enforce a redirect).							
FiddlerCore	`BeforeResponse`	●	●	●	●		
Event triggered when the response is ready to be sent from the proxy to the client. It makes it possible to modify the response content and headers.							
FiddlerCore	`AfterSessionComplete`	●	●	●	●	○	●
Called after the response was completely transfered to the client.							
CsExWB	`ProtocolHandlerOnResponse`	●	●	●	●	○	○

Component	Event	(1)	(2)	(3)	(4)	(5)	(6)
Description							
(HTTP(S) protocol handler event) Event triggered when the content of a request is interpretable by the browser. Apart from the URL and response headers corresponding to original requests, it also makes available the URL and headers of redirect if it was enforced by the server.							
CsExWB	`NavigateComplete2`	●	●	○	○	○	○
(IE window or frame event) Event fired when the main document of a specific navigation action was completely loaded (but possibly some external resources such as images or CSS files still need to be completed). It gives access to the current URL (possibly being the redirected URL) and to the reference to the Web browser corresponding to the affected frame or top-level window.							
CsExWB	`DocumentComplete2`	●	●	○	○	○	○
(IE window or frame event) Event triggered when the navigation action and all corresponding resources were fully loaded into the Web browser, but before the HTML BODY onload event is triggered (if it exists). Thus, even at this point there is no guarantee that the page is really completely loaded. It gives access to the current URL (possibly being the redirected URL) and to the reference to the Web browser corresponding to the affected frame or top-level window.							
CsExWB	`WBTopLevelOnLoad`	●	○	○	○	○	○
(HTMLEvents onload handler event) Event triggered in the case of top-level navigation, just before the BODY onload event handler defined for a given Web page is executed.							
CsExWB	`WBTopLevelAfterOnLoad`[1]	●	○	○	○	○	○
(HTMLEvents onload handler event) Event triggered in the case of top-level navigation, just after the BODY onload event handler defined for a given Web page is executed.							
CsExWB	`WBSecondaryOnLoad`	○	●	○	○	○	○
(HTMLEvents onload handler event) Analogous to WBTopLevelOnLoad, but triggered in the case of frame navigation. It gives access to the Web browser object corresponding to the target frame.							
CsExWB	`WBSecondaryAfterOnLoad`[2]	○	●	○	○	○	○
(HTMLEvents onload handler event) Analogous to WBTopLevelAfterOnLoad, but triggered in the case of frame navigation. It gives access to the Web browser object corresponding to the target frame.							
CsExWB	`WBOnDocumentChange`	○	○	○	◐	●	○
(IHTMLChangeSink::Notify event) Triggered whenever the DOM representation of the currently loaded document has been modified. In many cases it allows us to detect the moment when the result of an AJAX request or a pure JavaScript Web action was incorporated into the Web document.							

Table E.1: CsExWB and FiddlerCore Events Triggered For: (1) Top-level navigation, (2) frame navigation, (3) images, scripts, CSS files download, (4) AJAX requests, (5) pure JavaScript Web action, (6) Flash Web action

For single request, these events always happen in the order described in Table E.1. However, not all of them are triggered for all types of HTTP requests or Web actions, as marked in last six columns of the table.

All navigation actions[3] follow a similar sequence of triggered events, starting at `BeforeNavigate2` and ending at `WBTopLevelAfterOnLoad` (in the case of top-level navigation actions) or `WBSecondaryAfterOnLoad` (in the case of frame navigation actions).

[1]This event was not part of the original CsExWB library and was added by the author of this thesis.

[2]This event was not part of the original CsExWB library and was added by the author of this thesis.

[3]I.e. actions based on following links, form submissions, the manual entering of an URL address into the address bar, using a Web browser's back and forward buttons and triggering JavaScript Web actions that use `document.location` or `window.location` properties.

All other requests issued as part of Web actions (such as images, style files, scripts, Flash control or any AJAX-like data) trigger only protocol-level and proxy-level events. Moreover, in the case of AJAX-like requests that modify the current Web document after asynchronous content download, `WBOnDocumentChange` is also triggered. `WBOnDocumentChange` is also the only event that is triggered in the case of pure JavaScript Web actions, i.e. Web actions in which JavaScript modifies the current Web document without issuing any HTTP requests. Finally, in the case of Flash requests (i.e. requests issued by Flash controls in order to download extra data or graphical resources) only proxy-level events are triggered.

It is to be noted that while the presented order of triggered events is fixed, it concerns only a single HTTP request. However, in typical situations, for a single navigation action, after the `ProtocolHandlerOnResponse` event is trigger for the downloaded HTML file, multiple additional asynchronous requests corresponding to the download of additional page resources (images, styles, scripts) are generated and need to be finalized before `DocumentComplete2` can be triggered. Moreover, in the JavaScript `body onload` code additional data may be loaded through AJAX-like requests or additional IMG or SCRIPT elements may be generated, triggering the download of additional external resources. In such situations actual document completeness is deferred until all resources are downloaded. Finally, in Web pages with several frames, multiple navigation actions may happen in parallel, as in the case of some AJAX applications that may execute multiple asynchronous AJAX-like requests at once.

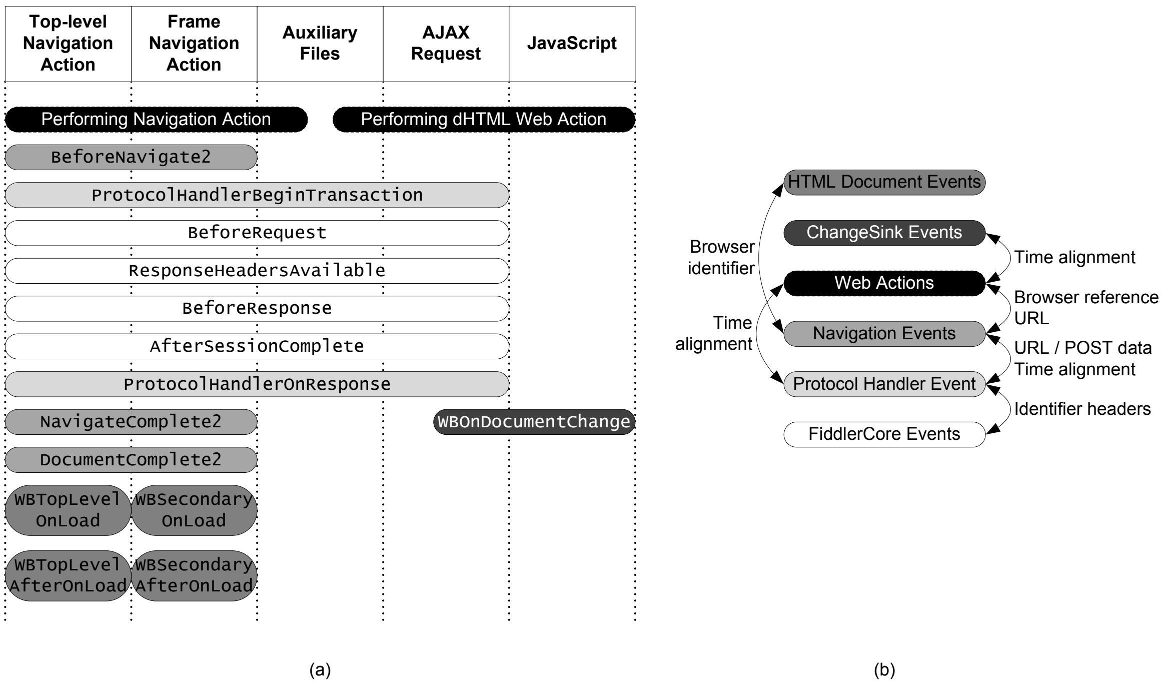


Figure E.1: Sequence and Methods of Connecting Events: (a) Time ordering of events in different scenarios. (b) methods used for connecting groups of events.

All these complex situations underline the importance of connecting multiple events related to the same Web action. It can be challenging, as events described in Table E.1 are triggered in different components that use different technologies and different request representations. In Figure E.1 we demonstrate a few aspects of the complexity of event sequences in different usage scenarios. In (a) we demonstrate the sequences of events in the case of top-level and frame navigation actions, auxiliary files (images, styles, scripts), AJAX requests and JavaScript. Events triggered by different components are marked by different background and text colors. In (b) we show methods that are used to logically connect events triggered by different components.

Connecting events triggered by the same component is relatively simple. All events that are triggered by FiddlerCore are easily connected thanks to their `oSession` argument that refers to the same object for all events. In the case of `BeforeNavigate2`, `NavigateComplete2` and `DocumentComplete2`, the browser document reference (i.e. Web browser instance corresponding to top-level document or specific frame) is always passed as argument, and is equal for all the events concerning the same navigation action. Browser identifier can be also used to connect the `DocumentComplete2` event with all HTML document `OnLoad` and `AfterOnLoad` events. However, connecting Fiddler events to navigation-related events to protocol-level events and to DOM changes is much more challenging. For connecting the events well, we combine a few methods, and assume that the relation of "being connected" is transitive.

The `BeforeNavigate2` event is connected to the corresponding `ProtocolHandlerBeginTransaction` event by the comparison of URL address, request headers and POST data (if applicable). If multiple requests are identical with respect to these elements (which is very improbable), we perform a temporal alignment, so that the first (chronologically) `BeforeNavigate2` event is connected to the first `ProtocolHandlerBeginTransaction` event.

To connect `ProtocolHandlerBeginTransaction` to corresponding Fiddler events, we use its capability to add extra request headers. The event handler adds an extra header containing an unique identifier of a given event. When this identifier is detected by the `BeforeRequest` event, the connection between protocol-level and proxy-level events is established. The `BeforeRequest` handler then removes this additional header from headers sent to the HTTP server, to avoid any confusion. The same identifier is added at the proxy-level to response headers in the handler of the `ResponseHeadersAvailable` event, so that the `ProtocolHandlerOnResponse` handler connects the received response with the previous request.

Another challenge is related to connecting Web actions to Web actions execution environment events. For most navigation actions it can be easily performed by comparing a Web action's URL and Web browser reference (corresponding to the frame or top-level Web browser) with those received by the `BeforeNavigate2` event. However, in the case of AJAX requests `BeforeNavigate2` is not triggered, thus there is a need of connecting Web action with protocol-level events.

In this case there are no natural identifiers that can be used for connecting events. Similar situations include the need of connecting pure JavaScript actions with their `WBOnDoucmentChange` events. In both these cases, we attempt to analyze the time of all events that could be possibly connected and use heuristics to choose the best possible connection. It works well for Web sites with a limited number of system-triggered events. For Web sites with a lot of unpredictable system-events we plan to use recognizers as part of our future work (see: Section 6.5).

Appendix F

List of Developed Test Web Sites

Web Site	Wrapped?
TS0. All data as single page. Basic Web site presenting all content as a single page.	●
TS1. Basic tree navigation. Web site with three types of pages (a list of makes, a list of models in a make, a list of cars in a model) connected by hyperlinks.	●
TS2. TS1 + JavaScript links. Identical to TS1 with links replaced by different types of JavaScript links (onclick, href="javascript:...").	●
TS3. Pagination. Similar to TS1 but instead of a single page with all models, each model has a separate page with navigation by "Next" and "Previous" buttons.	●
TS4. GET forms. Similar to TS1 but lists of makes and models contain HTML GET forms with single SELECT element instead of hyperlinks.	●
TS5. Overlapping categories. Similar to TS1 but the first page contains a list of segments (that overlap with respect to contained models) instead of makes.	◐
TS6. Lattice navigation. Web site with two ways of accessing exactly the same data: 1) by choosing make on home page and model + engine displacement on the next page, 2) by choosing engine displacement on home page and make + model on the second page.	◐
TS7. Repeated records. Identical to TS1 but some car records are repeated.	◐
TS8. Cookie-based content serving. As TS2 but each clicked link sets a Cookie that defines content to be presented and reloads page; Cookies are reset in onload event handler.	●
TS9. HTTPS with invalid certificate. Identical to TS1 but available through HTTPS with invalid certificate.	●
TS10. POST forms. Identical as TS4 but using HTML POST forms.	●
TS11. Open-domain POST forms. Similar to TS10 but make (specifically: any substring of make) needs to be entered into a text field.	●
TS12. Images, styles and JavaScript files. Similar to TS1, but with Web page templates that load several additional files such as JavaScript, CSS and images (that have no impact on data extraction).	●
TS13. HTTP errors. Identical to TS1 but some links cause 404 errors, and some links cause 500 error when accessed for the first time.	●
TS14. "Clipboard" for interesting cars. Individual cars for a model can be accessed only after adding a model to the "clipboard" of interesting cars; implements adding and removing models from a "interesting" list and preview of all "interesting" models with their cars.	●

Web Site	**Wrapped?**
TS15. Complete AJAX navigation. Navigation structure as in TS1 but all content loaded by AJAX actions.	●
TS16. Cumulative AJAX. Navigation structure as in TS1 but all content loaded by AJAX actions; new content added at the top of the page (without removing previous content).	●
TS17. Well structured encoded content and unstructured user content. Similar to TS16 but content presented in a very unstructured way (it is sent from HTTP server in a well-structured XML format and obfuscated by client-side JavaScript code).	●
TS18. Random-structured records. Similar to TS1 but all regular records are placed in an unstable template (e.g. in different tag-depth, with different preceding and following tags).	●
TS19. Merged attributes. Specification of each individual car as a single tag with multiple attributes, including untokenized attributes (engine power merged with engine type e.g. 80HDI).	●
TS20. Non-deterministic navigation. Similar to TS1 but before accessing some lists of individual cars an extra page with advertisement and “Next” button occurs.	●
TS21. Similar links to different types of pages. Similar to TS1+TS5, with home page containing a “tag cloud” containing links to different types of pages.	●
TS22. CAPTCHA. Similar to TS4 but accessing page containing a list of models requires not only choosing model but also doing CAPTCHA test (performing a math operation).	○
TS23. Pivot table. Individual cars are presented in a pivot table with rows corresponding to engine characteristics, columns corresponding to gearbox type and cells containing a list of versions for a given combination of engine and gearbox.	◐
TS24. Pivot table with grouping rows. Similar to TS23 but additional grouping rows corresponding to engine displacement are added, and engine displacement is removed from rows headers.	◐
TS25. Hierarchy of folders. All data in a single page consisting of embedded UL lists that form a hierarchy of unknown depth, based on analysis of shared attribute values.	●
TS26. Models similarity. Structure as in TS1 but extra links to similar models present at each page containing cars in a specific model.	●
TS27. Data ordering. All data in a single table with data ordering (clickable headers) and pagination.	●
TS28. GUI capabilities smaller than server capabilities. Similar as TS27 but gives also access to a form that allows to filter by model, engine displacement or version name; GUI enables search by a single criterion, but server accepts querying multiple criteria at once.	●
TS29. Iframes. Main pages with list of makes; one iframe for loading a list of models and the second one for loading cars.	●
TS30. Embedded framesets. Similar to TS29 but consists of frameset containing a list of makes and another frameset containing a list of models and a list of cars.	●
TS31. Web sites with onscroll Web actions. For each make a separate set of paginated result is provided with three models with all their versions per page; the link to the next three models appears only when the page is scrolled to the end.	●
TS32. JavaScript-based building of a form (e.g. defining price range). JavaScript components are used for scroll-based selection of minimal and maximal engine displacement; JavaScript code and an HTML form with hidden fields are used to send query to the server.	●
TS33. JavaScript-based content decoding. The HTML content downloaded from the server is obfuscated and decoded by JavaScript code in onload event.	●

Web Site	**Wrapped?**
TS34. Logical data alignment through absolute positioning. A Web site where individual items are well-structured; however, they are places in a source code in random order and positioned only after onload event by using JavaScript and CSS absolute positioning.	○
TS35. Timer-triggered event after on-load. Loads additional content via AJAX 60 seconds after onload event.	○
TS36. Pure timer-triggered navigation. Clicking a link displays a message that page will be loaded in five seconds, and redirect is performed based on JavaScript timer code.	○
TS37. Data for the same record in many locations in single page. Attributes about the same version contained in two separated lists, integrable thanks to record identifiers.	●
TS38. Web site with limit of a total number of requests. Web site that limits a number of consecutive Web page accesses from a single IP to 200 per 12h.	◐
TS39. Web site with limit of frequency of requests. Web site that limits the number of requests in every minute to 3 and in every consecutive ten minutes to 15.	●

Table F.1: List and Description of Test Web Sites Created for Evaluation of Our Solution That Are Completely Wrapped (●), Partially Wrapped (◐) and Not Wrapped (○) by Our Solution

List of Used Symbols

Symbol	Description
$\xrightarrow{w}$	Navigation state transition via Web action w (See: Definition 4.2.6)
$\bowtie$	Provenance-aware join (See: Section 4.4.6)
$\leq$	Refinement relation of input configurations (See: Definition 4.3.12)
$\perp$	Independence relation of input configurations (See: Definition 4.3.13)
$\alpha^{(n)}$	Data extraction instance with n input slots (See: Definition 4.3.1)
γ	Data extraction node (See: Definition 4.3.3)
Γ	Set of all data extraction nodes (See: Definition 4.3.3)
ϵ	Data extraction atom (See: Definition 4.3.2)
λ	Input mapping (See: Definition 4.3.7)
Λ	Set of all input mappings (See: Definition 4.3.7)
π	Attribute set projection operator (See: Definition 4.2.13)
σ	Attribute set selection operator (See: Definition 4.2.13)
A	Set of all data extraction atoms (See: Definition 4.3.2)
as	Attribute set (See: Definition 4.2.13)
C_x	All client-side navigation states existing in Web site x (See: Definition 4.2.2)
D	Data extraction graph (See: Definition 4.3.10)
$d(\lambda)$	Destination node of input mapping λ (See: Definition 4.3.7)
$d(isc)$	Destination node of input configuration isc (See: Definition 4.3.11)
dr	Data record (See: Definition 4.2.12)
drs	Data records schema (See: Definition 4.2.11)
$drs(dr)$	Data records schema of a record dr (See: Definition 4.2.12)
drs_γ	Data records schema of all outputs of data extraction node γ (See: Definition 4.3.3)
$ds(\lambda)$	Set of all destination slots of input mapping λ (See: Definition 4.3.9)
$ds(isc)$	Set of all destination slots of input configuration isc (See: Definition 4.3.11)
ert	Extraction rule template (See: Definition 4.2.9)
E_x	All extraction rules possible in a Web site x (See: Definition 4.2.8)
$I_\mathcal{L}$	Input objects accepted by extraction language $\mathcal{L}$ other than WO_x (See: Section 4.2.3)
$iIM(\gamma)$	Set of all input mappings incoming to node γ (See: Definition 4.3.7)
$is(\gamma)$	Input slots of data extraction node γ (See: Definition 4.3.4)

Symbol	Description
isc	Input configuration (See: Definition 4.3.11)
$\mathcal{L}$	Extraction language $\mathcal{L}$ (See: Section 4.2.3)
l_λ	Label of input mapping λ (See: Definition 4.3.7)
n	Navigation state (See: Definition 4.2.4)
$n^{(p)}$	Part of navigation state n named p (See: Definition 4.5.1)
N_x	All navigation states existing in Web site x (See: Definition 4.2.4)
$\mathbb{O}$	Set of all objects of any data type (See: Definition 4.3.1)
$\mathbb{O}_t$	Set of all objects of data type t (See: Definition 4.2.10)
$oIM(\gamma)$	Set of all input mappings outgoing from node γ (See: Definition 4.3.7)
$os(\gamma)$	Output slots of data extraction node γ (See: Definition 4.3.5)
qs	Query state (see: Definition 4.5.10)
$\mathbb{R}$	Set of all record values (See: Section 4.2.6)
$s(\lambda)$	Source node of input mapping λ (See: Definition 4.3.7)
$\mathbb{S}$	Set of all text (string) values (See: Definition 4.2.7)
$\mathbb{S}_t$	Set of all text (string) values convertible to data type t (See: Definition 4.2.10)
S_x	All server-side navigation states existing in Web site x (See: Definition 4.2.3)
sa_γ	Node's state annotation of node γ (See: Definition 4.5.4)
$sam_\gamma^\varnothing$	Set of all navigation state parts for which γ is a state-resetting node (See: Definition 4.5.5)
sam_γ^+	Set of all navigation state parts for which γ is a state-cumulative node (See: Definition 4.5.6)
sam_γ^*	Set of all navigation state parts for which γ is a state replacement node (See: Definition 4.5.7)
$sam_\gamma^\bigstar$	Set of all navigation state parts for which γ is a complete state replacement node (See: Definition 4.5.8)
$sam_\gamma^\blacklozenge$	Set of all navigation state parts for which γ is a partial state replacement node (See: Definition 4.5.8)
san	Semantic annotation of an attribute of data records schema (See: Definition 4.2.11)
$ss(\lambda)$	Set of all source slots of input mapping λ (See: Definition 4.3.8)
t	Data type (See: Definition 4.2.10)
$\mathbb{T}$	Set of all data types (See: Definition 4.2.10)
uq	User query (See: Definition 4.6.1)
uq_π	Projection operator of user query (See: Definition 4.6.1)
uq_σ	Selection operator of user query (See: Definition 4.6.1)
uq_i	User query input specification (See: Definition 4.6.1)
uq_o	User query ordering specification (See: Definition 4.6.1)
uq_{rs}	Output node of user query (See: Definition 4.6.1)
W_x	All Web actions possible in Web site x (See: Definition 4.2.5)
wat	Web action template (See: Definition 4.2.7)
WO_x	Set of all outputs of Web actions in Web site x (See: Definition 4.2.5)

List of Figures

1 Applied Methodology . . . xiii

1.1 Pyramid of Wisdom . . . 2
1.2 Transformation of Data Into Wisdom . . . 4
1.3 Different Types of Data-Intensive Web Sites . . . 13
1.4 Business Models and Value Networks / Dynamic Markets . . . 17
1.5 Information Overload and Decision Taking . . . 19

2.1 User's Interactions With Complex Web Site . . . 34

3.1 Idea Flow Between Existing Systems . . . 56

4.1 Example of Client-Side State Evolution . . . 71
4.2 Example of Server-Side State Evolution . . . 72
4.3 Screen Shots of Six pages of Demo Web Site . . . 89
4.4 An Example of Data Extraction Graph . . . 90
4.5 Basic Input Configurations . . . 91
4.6 Advanced Input Configurations . . . 92
4.7 Example of Graph Width Cycles . . . 98
4.8 Cyclic Graph for Extraction of Hierarchies of Unknown Depth . . . 98
4.9 Example of Navigation State Ambiguity and Unambiguity . . . 124
4.10 Example of Required Subgraph Selection . . . 128
4.11 Adding Data Filtering Attribute Sets . . . 131
4.12 Removing Nodes That Provide Not Used Attributes . . . 134

5.1 System Architecture . . . 144
5.2 UML Class Diagram of Data Model Component . . . 146
5.3 UML Class Diagram of Graph Execution Component . . . 147
5.4 Screen Shot of User Interface of Our Prototype . . . 151

6.1 Data Model of Test Web Sites . . . 166

E.1 Sequence and Methods of Connecting Events . . . 273

List of Tables

1.1 Value-Added of E-commerce Intermediaries . . . 16
1.2 Why Decision-Making Is Increasingly Challenging . . . 18

2.1 Transformations During User-Web Site Interactions . . . 33

3.1 Previous Solutions – Overview of Addressed Challenges . . . 49
3.2 Previous Solutions – Output Characteristics . . . 51
3.3 Previous Solutions – Rules Characteristics . . . 53
3.4 Previous Solutions – Rules Management . . . 55
3.5 Previous Solutions – Performance of Chosen Solutions (1) . . . 58
3.6 Recall of Some Systems on Three RISE Sources . . . 59
3.7 Previous Solutions – Performance of Chosen Solutions (2) . . . 59
3.8 Previous Solutions – Performance of Chosen Solutions (3) . . . 60

4.1 Detailed Objectives of Proposed Solution . . . 69
4.2 Nodes of Example Graph . . . 90
4.3 Example of Extracted Values With Their Provenance . . . 110
4.4 Description of Nodes of “Required Subgraph Selection” Example Graph . . . 129

6.1 Comparison With Previous Work – Output Characterisitcs . . . 160
6.2 Comparison With Previous Work – Extraction Rules Characterisitcs . . . 161
6.3 Comparison With Previous Work – Wrapper Creation and Management . . . 162
6.4 Challenges Addressed by Proposed Solution . . . 168
6.5 Addressed Challenges – Comparison with Previous Solutions . . . 178
6.6 Handled Types of Web Sites . . . 183
6.7 Handled Business Usage Scenarios . . . 185
6.8 Handled Technical Usage Scenarios . . . 187
6.9 Queries Used for Performance Assessment . . . 189
6.10 Performance – Comparison With Previous Systems . . . 190
6.11 Performance – Impact of Capture Request Optimizations . . . 192

A.1 User-Web Site Interactions Components . . . 222

B.1 General Dataset - Candidate Web Sites 230
B.2 General Dataset - Preliminary Analysis 232

C.1 Information Extraction Challenges 242

D.1 Other Related Work 269

E.1 CsExWB and FiddlerCore Events 272

F.1 List of Test Web Sites Used in Evaluation 277

List of Definitions

4.2.1 Web Document . . . 72
4.2.2 Client-Side Navigation State . . . 73
4.2.3 Server-Side Navigation State . . . 73
4.2.4 Navigation State . . . 73
4.2.5 Web Action . . . 74
4.2.6 Navigation State Transition . . . 74
4.2.7 Web Action Template . . . 76
4.2.8 Data Extraction Rule . . . 78
4.2.9 Extraction Rule Template . . . 80
4.2.10 Data Type . . . 81
4.2.11 Data Records Schema . . . 82
4.2.12 Data Record . . . 82
4.2.13 Attribute Set . . . 83
4.3.1 Data Extraction Instance . . . 86
4.3.2 Data Extraction Atom . . . 86
4.3.3 Data Extraction Node . . . 87
4.3.4 Node's Input Slots . . . 87
4.3.5 Node's Output Slots . . . 87
4.3.6 Trivial Data Extraction Node . . . 87
4.3.7 Input Mapping . . . 88
4.3.8 Input Mapping Source Slots . . . 89
4.3.9 Input Mapping Destination Slots . . . 89
4.3.10 Data Extraction Graph . . . 91
4.3.11 Input Configuration . . . 93
4.3.12 Refinement of Input Configuration . . . 94
4.3.13 Independent Input Configurations . . . 94
4.3.14 Equivalent Input Configurations . . . 94
4.3.15 Valid Input Mapping . . . 95
4.3.16 Executable Data Extraction Node . . . 95
4.3.17 Executable Data Extraction Graph . . . 96
4.3.18 Maximal Executable Subgraph of Data Extraction Graph . . . 96

4.3.19 Completely Executable Data Extraction Graph . . . 96
4.5.1 Part of Navigation State . . . 116
4.5.2 Node Modifying Part of Navigation State . . . 116
4.5.3 Node Depending on Part of Navigation State . . . 116
4.5.4 Node's State Annotation . . . 117
4.5.5 State Reseting Node . . . 120
4.5.6 State-Cumulative Node . . . 120
4.5.7 State Replacement Node . . . 120
4.5.8 Complete and Partial State Replacement Nodes . . . 120
4.5.9 Navigation State Ambiguity . . . 125
4.5.10 Query State . . . 126
4.6.1 User Query . . . 136

List of Examples

4.2.1 Changes of Client-Side Navigation State . . . 71
4.2.2 Changes of Server-Side Navigation State . . . 71
4.2.3 Generalization of Similar URLs . . . 75
4.2.4 Generalization of Similar User Interface Web Actions . . . 75
4.2.5 Regular Expression Extraction Rule . . . 76
4.2.6 XPath Extraction Rule . . . 77
4.2.7 Extraction Rules Composition . . . 78
4.2.8 XPath Extraction Rule Template With Hook . . . 79
4.2.9 Different Attributes . . . 81
4.2.10 Data Types of Extracted Data . . . 81
4.2.11 Simple Attribute Set . . . 83
4.2.12 Attribute Set Selection and Projection Operators . . . 83
4.3.1 Nodes, Instances and Atoms (WAT) . . . 85
4.3.2 Nodes, Instances and Atoms (ERT) . . . 85
4.3.3 Nodes, Instances and Atoms (AS) . . . 86
4.3.4 Complete Data Extraction Graph . . . 89
4.3.5 Basic Input Configurations . . . 91
4.3.6 Complex Input Configurations . . . 92
4.3.7 Relation Between Multiple Input Configurations . . . 94
4.3.8 Equivalent Input Configurations . . . 95
4.3.9 Using a Cyclic Graph for Pagination Modeling . . . 98
4.3.10 Using a Cyclic Graph to Extract Folders Tree of Unknown Depth . . . 98
4.4.1 Execution of Data Extraction Graph . . . 100
4.4.2 Provenance-Aware Join . . . 108
4.5.1 Shopping Cart Part of Navigation State . . . 114
4.5.2 Current Web Document Part of Navigation State . . . 115
4.5.3 Navigation State Recreation Need . . . 116
4.5.4 Transforming Partial to Complete State Replacement . . . 121
4.5.5 Transforming Partial State Replacement Node to State-Cumulative and Complete State Replacement Node . . . 121
4.5.6 Navigation State Ambiguity . . . 123

4.5.7 Complex Navigation State Ambiguity 124
4.6.1 Required Subgraph Selection 129
4.6.2 Data Scope Modification 130
4.6.3 Data Filtering Attribute Sets 131
4.6.4 Incorporating Data Filtering Into Extraction Rule Template 132
4.6.5 Data Filtering Incorporation in Case of Regular Expressions 132
4.6.6 Data Filtering Incorporation in Case of XPath Extraction Rule 133
4.6.7 Removing Useless Input Mappings 133
4.6.8 Removing Useless Destination Slots From Input Mappings 134
4.6.9 Removing Useless Input Mappings and Nodes 135
4.6.10 Node Decomposition Limiting Navigation State Recreations 135

List of Algorithms

4.4.1 ANALYZENODERECURSIVELY($\gamma, IM^{\gamma}, startIndex, candidate$) . . . 101
4.4.2 ANALYZEGRAPH(D) . . . 102
4.4.3 FINDEXECUTABLESUBGRAPH(D) . . . 103
4.4.4 EXECUTEGRAPH(D) . . . 105
4.4.5 ENQUEUEBFS(a) . . . 107
4.4.6 ENQUEUEDFS(a) . . . 108
4.4.7 POPQUEUEBASIC() . . . 108
4.4.8 QUEUEEMPTY() . . . 108
4.4.9 PROVENANCEAWAREJOIN($ic, data$) . . . 111
4.4.10 STOREDATA($\lambda, data$) . . . 112
4.4.11 NODEREADY(n) . . . 113
4.4.12 NEWINPUTRECORDS(n) . . . 113
4.5.1 EXECUTE($atom$) . . . 118
4.5.2 CREATEATOM($instance, r$) . . . 119
4.5.3 ENQUEUESTATEAWARE(a) . . . 122
4.5.4 POPQUEUESTATEAWARE() . . . 123
4.5.5 QUEUEEMPTYSTATEAWARE() . . . 123
4.5.6 ANALYZEGRAPHSTATEAWARE(D) . . . 125
4.6.1 PLANQUERY(D, uq) . . . 138
4.6.2 STATEAWARENODEDECOMPOSITION(D) . . . 139
4.6.3 MERGENODES(D) . . . 140
4.6.4 LINEARIZE(D) . . . 141