Chapter 6:
Web Data Extraction for Service Creation

Robert Baumgartner[1], Alessandro Campi[2],
Georg Gottlob[3], and Marcus Herzog[1]

[1] Lixto Software GmbH, Favoritenstrasse 9-11, 1040 Wien, Austria
{robert.baumgartner,marcus.herzog}@lixto.com
[2] Politecnico di Milano, DEI, Piazza Leonardo da Vinci 32, 20133 Milano, Italy
campi@elet.polimi.it
[3] Computing Laboratory, Oxford University, U.K.
gottlob@comlab.ox.ac.uk

Abstract. Web data extraction is an enabling technique in the search computing scenario. In this chapter, we first review the state of the art in wrapper technologies focusing on how wrapper generators can be used to create unified services that integrate data from Web Applications and Web services in various domains. Next, we describe the Lixto approach and we present the Lixto Suite as one example of Web Process Integration. Finally, application areas and future challenges and the usage of wrapper technologies in the search computing context is discussed.

1 Introduction

Although in today's Web much data is available via APIs, light-weight and heavy-weight Web service techniques, the larger amount of data is still only available in semi-structured formats such as HTML. In the recent years, Web pages became more complex and turned into Web Applications, using a lot of Web 2.0 and Rich Internet Application technologies. As a consequence, new research and technical challenges emerged, related to automated Web navigation and data extraction.

To use Web data in Enterprise Applications and service-oriented architectures, it is crucial to provide means for automatically turning Web Applications and Web sites into Web Services, allowing structured and unified access to heterogeneous sources. This includes to understand the logic of the Web application, to fill out form values, and to grab relevant data – all these aspects need to be reflected accordingly in the generated Web Service.

In a number of business areas, Web applications are predominant among business partners for communication and business processes. Various types of processes are carried out on Web portals, covering activities such as purchase, sales, or quality management, by manually interacting with Web sites.

Wrapper Generators enable the automation of processes and operations of Web Applications. They pave the way for *Web Process Integration*, i.e. the seamless integration of Web applications into a corporate infrastructure or service oriented landscape by generating Web services from given Web sites. Web

S. Ceri and M. Brambilla (Eds.): Search Computing, LNCS 5950, pp. 94–113, 2010.

process integration can be understood as front-end integration: integrate cooperative and non-cooperative sources without the need for information provider to change their backend. Furthermore, regarding light-weight mashup techniques, wrapper generators offer to extend the range of sources under consideration from structured Web feeds to include legacy Web applications. In this sense, Web process integration and Wrapper Technologies are essential enablers of the *Web of Services*.

The rest of the chapter is structured as follows. In Section 2, an overview of the state-of-the-art in Web data extraction methods and techniques is given. In Section 3 we give an overview of Lixto and its architecture as an example of wrapping technology used for search computing and for generating Web Process Integration scenarios. Section 4 is dedicated to survey the process of turning Web 2.0 Applications into Web Services, illustrated by describing the Lixto components and examples. Section 5 gives an overview on sample application areas as well as a summary of future research issues and the usage of Web data extraction in search computing context. Finally, some brief concluding remarks are given in Section 6.

2 Web Data Extraction

Web data extraction is a research field rooted in information extraction from text, in screen scrapers invented for extracting screen formatted data from mainframe applications for terminals such as VT100 or IBM 3270, and in ETL (Extract, Transform, Load) methods defined to extract information from various business processes and feed it into databases [8].

One of the first attempts to extract information from unstructured sources is [39] that presents AutoSlog, a system that automatically builds a domain-specific dictionary of concepts for extracting information from text. A significant step forward was Crystal [45], a system which allows one to automatically build a dictionary of entities from a text.

[32] presents a trial to classify wrappers considering in particular their expressiveness. Laender [34] proposed a taxonomy for data extraction tools based on the main technique used by each tool to generate a wrapper. [30] is a more recent survey on wrapper technology.

Wrapper Generation Systems can be classified according to different properties. One main such distinctive criteria is the mode of wrapper generation. This spans from manual wrapper writing (using e.g. some special-purpose APIs) to visual and interactive approaches where the user is guided through the wrapper generation and fully automated approaches. Fully automated approaches include on the one hand inductive learning based on positive and negative examples, and on the other hand unsupervised learning of similar patterns, usually restricted to a particular domain such as digital cameras. Automatic approaches tend to be limited in expressive power and robustness, but on the other hand are essential for large scale extraction scenarios.

One further differentiating criteria is the wrapper language and the objects a wrapper operates on. This ranges from perceiving wrappers as mapping

functions from node sets to node sets, various logical and automata theoretic representations, textual pattern matching on string representations of Web pages, and usage of natural language processing techniques.

The first studies dedicated to Web extraction [1,18,20,44] led the development of semi-automated systems, capable of extracting information in an automatic manner only after a training phase, performed with user intervention. TSIMMIS [23] proposes a framework for the manual construction of Web wrappers. In TSIMMIS a wrapper takes as input a specification made by a sequence of commands given by programmers describing the pages and how the data should be transformed into objects. Commands take the form (*variables*, *source*, *pattern*), where *source* specifies the input text to be considered, *pattern* specifies how to find the text of interest within the source, and *variables* are a list of variables that hold the extracted results. The generated outputs are represented using the *Object Exchange Model*. The output is composed by the target data and by additional information about the result. NoDoSE (Northwestern Document Structure Extractor) [1] is an interactive tool for semi-automatically determining the structure of such documents and then extracting their data. The user hierarchically outlines the interesting regions of files and describes their semantics. A mining component attempts to infer the *grammar* of the file from the information taken from the user. WebOQL [3] synthesizes ideas taken from query languages for the Web, from query languages for semistructured data and from languages for website restructuring. WebOQL is based on the usage of *hypertrees*, i.e., labeled ordered trees suitable to support collections, nesting, and ordering.

XWRAP [37] is a wrapper generation framework. XWRAP uses a common library to provide basic building blocks for wrapper programs. In this way, tasks of building wrappers specific to a Web source are separated from repetitive tasks for multiple sources. The wrapper building process is divided into two steps: the encoding of the source-specific metadata knowledge and the combination of the information extraction rules generated at the first phase. W4F (Wysiwyg Web Wrapper Factory) [40] is a Java toolkit to generate Web wrappers. The process is done in three steps: *retrieval*, *extraction*, and *mapping*. The first step retrieves a document from the Web and builds a DOM using an HTML parser. The next two steps apply a set of rules expressed in HEL (HTML Extraction Language) on the parse tree to extract information. Extracted information is stored using a proprietary format called NSL (Nested String List). Iepad [28] discovers extraction rules from Web pages. The system defines a data structure, called *PAT tree*, useful for the search of repeated patterns. Exploiting repeated pattern mining the system automatically identifies record boundaries.

RoadRunner [17] is based on a grammar inference techniques. It is based on an algorithm, called *match*, that exploits similarities and differences among a set of sample pages in order to infer a common grammar, which is then used as a wrapper. Previous results were obtained in Minerva [15], an attempt to exploit declarative grammar-based approaches and procedural programming in order to handle heterogeneities and exceptions. The idea is to allow the insertion of exception-handling mechanism in grammars using a special language called

editor. [16] defines a formal theoretical framework in which it is proved that Match runs in Ptime, whenever pages are compliant with a class of languages called Prefix Mark-up Languages. As real-life Web pages seldomly fall in this class of Languages, some studies have recently tackled the problem of improving Match in order to automatically infer a wrapper for a wider class of languages.

DEByE (Data Extraction By Example) [33] uses a small set of examples specified by the user that interacts with a tool using *nested tables* as the visual paradigm. The user defined examples are used to generate patterns which allow extracting data from new documents. For the extraction DEByE adopts a bottom-up procedure very effective with many different types of Web sources.

WARGO [38] is a system developed to allow non-technical users to generate complete wrappers for Web sources. Access to the pages containing required data is described by means of complex *Web flows* built by simply navigating with a Web browser. The *parsing* is made using interactive tool that allow users to generate complex extraction patterns by simply highlighting relevant data from very few example pages, and answering some simple questions. The system internally relies on NSEQL (Navigation SEQuence Language) for specifying navigation sequences and DEXTL (Data EXTraction Language) for specifying extraction patterns.

EXALG [2] is an algorithm capable of extracting structured data from a collection of Web pages generated by encoding data from a database into a common *template*. To discover the underlying *template* that generated the pages, EXALG uses so called Large and Frequently occurring EQuivalent classes (LFEQ), i.e. sets of words that have similar occurrence pattern in the input pages. The MGS framework [24] is based on the intuition that, on the Web, the set of attributes composing an underlying schema is limited and that there is a strong overlapping between the sources. Most of the selection of the sources and part of the extraction is done by hand. The work in [31,36] describes wrapper generation with particular emphasis on their robustness. [35] proposes an approach for the automatic extraction and segmentation of records from Web tables. The proposed approach relies on a specific pattern that occurs on many Web pages for presenting lists of items: a index page containing a list of short summaries, one for each item, which include a link leading to a page about details of the specific item. Their approach leverages on the redundant information of this pattern and is based on constraint satisfaction problems and on probabilistic inference techniques.

[14] describes a system capable to populate a probabilistic database with data extracted from the Web. Data extraction is performed by TextRunner [19], an information extraction system. The massive extraction of data from the Web is the subject of *WebTables* [13,29]. However, they just concentrate on data that is published in HTML tables, and do not perform any integration of the extracted data. The work in [43] is an attempt do demonstrate that developing information extraction programs using Datalog with embedded procedural extraction predicates is a good way to proceed. Datalog provides a cleaner and more powerful

way to compose small extraction modules into larger programs. Second, query optimization can be applied to Datalog programs.

Cimple [41,42] is a system based on the interaction of an expert to provide a set of relevant sources, to design an entity relationship model describing the domain of interest, and to compose the operators for the extraction of the data from the pages. MetaQuerier [25] supports exploration and integration of databases on the Web and concentrates its contribution on exploration of the deep Web. It exploits the regularities of web forms and automatically matches interfaces.

Flint [12] automatically searches, collects and indexes Web pages publishing data representing an instance of a certain conceptual entity. Flint takes as input a small set of labeled sample pages: it automatically infers a description of the underlying conceptual entity and then searches the Web for other pages containing data representing the same entity. Flint automatically extracts data from the collected pages and stores them into a semi-structured self-describing database.

Finally, as of today, a number of commercial systems emerged, mostly in the area of interactive wrapper generation. This includes the *Denodo* ITPilot, *WebQL* (using a SQL-like query language for the Web) and *KapowTech*'s Mashup Server. Commercial frameworks applying machine learning techniques include the Dapp Factory from *Dapper* and the *Fetch* Agent Plattform.

3 The Lixto Approach

Lixto offers state-of-the-art products for Web data extraction and integration and services for SOA-Enablement, Mashup Enablement, Market Monitoring, and Vertical Search. In this setting, we look at Lixto technology from the perspective of an enabling technology for the creation of Web process integration and search computing scenarios.

With the Lixto Visual Developer (VD), wrappers are created in an entirely visual and interactive fashion. Figure 1 sketches the architecture of VD and its runtime components.

The VD is an Eclipse-based visual integrated development environment (IDE). It embeds the *Mozilla* browser and interacts with it on various levels, e.g. for highlighting Web objects, interpreting mouse clicks, or interacting with the document object model (DOM). Usually, the application designer creates or imports a *data model* as a first step. The data model is an XML schema-based representation of the application domain.

Figure 2 gives a screenshot of the GUI of the Visual Developer. On the left-hand side, the project overview and the outline view of the currently active wrapper are illustrated. In the center, the embedded browser is shown. At the bottom, in the Property View, navigation and extraction actions can be inspected and configured (as shown in Figure 3).

During *wrapper creation*, the application designer visually creates deep Web navigations (e.g., form filling), logical elements (e.g., click if exists), and extraction rules. The system supports this process with automatic recording, immediate feedback mechanisms, and generalization heuristics. The application designer

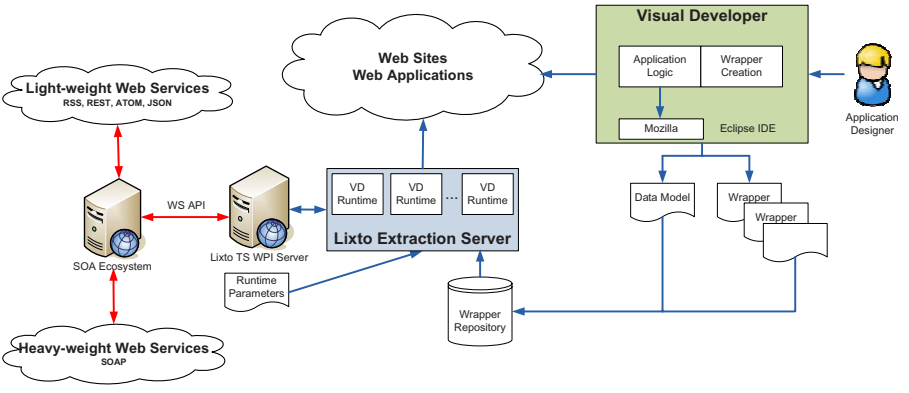

Fig. 1. Environment and Architecture

Fig. 2. Lixto Visual Developer

creates the wrapper based on samples, both in the case of navigation steps, and in the case of extraction steps. Finally, the designer parameterizes search, filtering and extraction steps of the wrapper. These parameters form the input values for the exhibited Web Service methods.

The internal extraction language *Elog* [5,22], the Web object detection based on XPath2, token grammars, and regular expressions are part of the *application logic*. Moreover, the application logic comprises deep Web navigation and workflow elements for understanding Web processes.

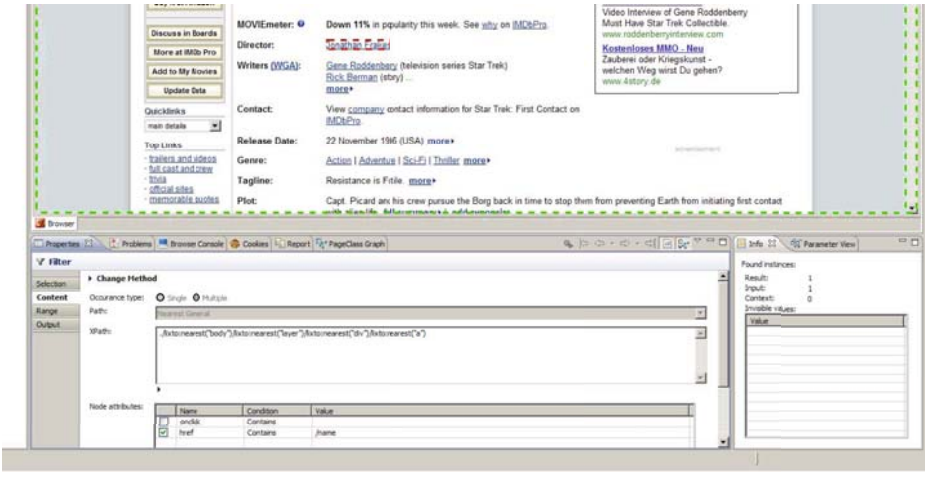

Fig. 3. Visual Filter and Condition Creation

Wrappers and data models are uploaded to the server. In the Web Process integration scenario, the WPI Edition of the Lixto Transformation Server (TS) is used (refer to Figure 1 again). In the SOA-oriented architecture of Lixto, servers such as the TS access the VD Runtimes via Web Service or Java RMI. Lixto TS exposes a query interface for ad-hoc and scheduled Web queries, and a Web Service entry-point where each request provides information about the wrapper to be executed and the runtime parameters (e.g. values for filling forms).

At wrapper execution time, each VD runtime, a.k.a. *VD head*, runs as independent process, using its own browser instance (during such executions the browser GUI is suppressed). *Lixto Extraction Server* spawns a number of VD heads and communicates results back to the server. Additionally, since Web applications can act unreliably, Extraction Server is capable of terminating and creating new heads to retry the wrapper if necessary. This architecture leverages Web extraction to a very stable and reliable process – browser instances of parallel executions do not interfere with each other, and in case of any problems with Web sites, parts of wrapper executions are retried in a new head. Due to the extraction process, Web data and Web applications can be consumed via the Lixto WPI Server as conveniently as usual light-weight and heavy-weight Web services.

4 Transforming Web Pages and Deep Web Sources into Web Services

In the following we exemplify the usage of Lixto components for turning a Web application into a Web service. As example we consider the IMDB site (International Movie Database `http://www.imdb.com`). The site offers information and

news about movies, tv shows, and actors. Although some information can be accessed as structured RSS feed, the majority of the data is primarily intended for manual browsing. In the following example we will extract information on particular movies, extract information about the characters and actors in the movies, and additionally extract available images about the actors. Information is extracted based on particular parameters, such as giving a movie title, specifying whether to return movies with the exact title only, return more than one movie, how many of the main characters to extract, and how many photos to include.

After defining how to extract and clean the information, which parameters can be specified and publishing it on the WPI Server, the service can be conveniently consumed by service-oriented applications.

4.1 Wrapper Generation with Lixto Visual Developer

Deep Web and Web Application Traversal. A wrapper project in the Visual Developer comprises a number of *actions*. Actions include mouse and key events occurring during a Web navigation [4]. Such actions are e.g. link traversing, filling out textboxes, selection from lists, or opening menus. One special action is the "Data Extractor" action. Inside of this action a declarative Elog program [6] resides. In the Elog program, exit points to different pages such as "next" pages, detail pages, or dynamic changes on a page are provided – this way one can conveniently iterate over entries in selection boxes. Further actions include procedural flow controls such as if conditions, while conditions, and call actions to other page classes.

An example of a simple click action is clicking on a link to traverse to a new Web page. The corresponding action stores a generalized XPath to the corresponding link element and the information that a single mouse click is performed on this particular element. The XPath is made as robust as possible by the system to ensure a stable navigation replay even in case of changes on the Web page.

Actions are embedded in declarative templates, so-called *page classes*. In Figure 2, the wrapper outline view is shown on the left-hand side (the flow in the first page class is enlarged in Figure 4), illustrating the procedural action flow and the Elog extractors in the declarative page class templates. At the bottom, the actual page class dependencies are given.

Consider the wrapper for IMDB in Figure 2 and especially the page class dependency graph illustrated in higher resolution in Figure 5: In the "start" page class the search form is filled out and results are extracted. Moreover, based on given parameters, it is decided which elements are clicked to reach the movie detail page. For each of these, the page class "movie" is called. In this page class, the details such as director and year are extracted, as well as the most important characters and their actors. Since character and cast information is on different pages with a different structure, different data extractors are used in

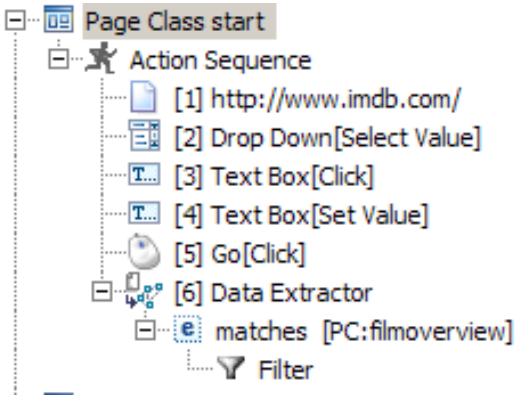

Fig. 4. A sample Page Class

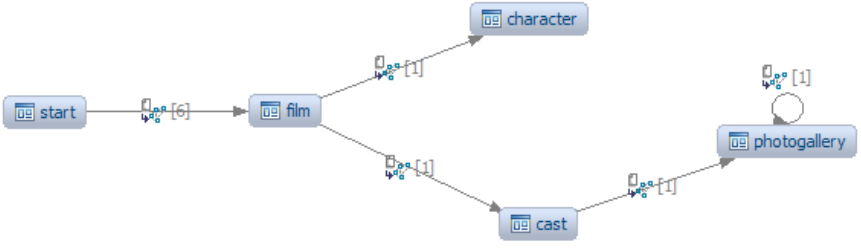

Fig. 5. IMDB Application Flow

the page class "cast" and "character", respectively. Finally, photos of the actors can be reached from the actor page. 48 photos per page are shown, hence the page class "photo" is iteratively called until there is no next link or the given limit is reached.

Due to branching and iteration capabilities of the navigation language, complex process flows can be modeled on top of the page class concept such as e.g. a flight booking process.

Web Data Extraction Language. The internal data extraction language, Elog, is a datalog-like language especially designed for wrapper generation. The Elog language operates on Web objects, that are HTML elements, lists of HTML elements, and strings. Elog rules can be specified fully visually without knowledge of the Elog language. Web objects can be identified based on internal, contextual, and range conditions and are extracted as so-called "pattern instances".

A typical Elog rule in the IMDB wrapper to extract the director of the movie looks like:

```
director(X0, X1) :-
    root(_, X0), subelem(X0,
      (./lixto:nearest("body")/lixto:nearest("layer")
       /lixto:nearest("div")/lixto:nearest("a"),
      [("href", "name", substring)]), X1),
    before(X1, ../h5, [("text",
      "^Director.*", regexp)]), 0, 1, X2, X3).
```

The "director" predicate used in the head of the rule evaluates to true for all assignments X_1 where the body holds true. In the "subelem" predicate, for each assignment of X_0 (matches of the "root" pattern) assignments for the result of the XPath generation are stored in X_1. The "before" predicate refers to instances of X_1, its results could be referenced by further predicates. The numerical values reflect distance settings (based on the node level), in this case immediately before.

Elog uses different kind of expressions to identify Web objects – this includes XPath2 statements (and extension functions) for tree nodes and regular expressions or predefined ontology concepts for textual data, and is open to be extended to e.g. extract based on the visual representation in the browser. Figure 3 illustrates how this rule is presented to wrapper designers.

Among the evaluation criteria of a wrapping language, expressiveness and robustness are the most important ones. Robustness grants that information on frequently changing Web pages are correctly discovered, even if e.g. a banner or a new page fragment is introduced. Visual Developer offers robust mechanisms of data extraction based on the two paradigms of tree and string extraction. Verification alerts can be imposed that give warnings in case user-defined criteria are no longer satisfied on a page. [22] shows a kernel fragment of *Elog* that captures monadic second order logic, hence if is very expressive while at the same time easy to use due to visual specification.

Visual Wrapper Generation. The usage of both Elog and of the internal Web interaction language is completely invisible to the average wrapper designer and all operations are carried out by visual means. In simple scenarios this is basically comprised of four steps:

1. First, the *modeling phase*, where the application designer defines an XML Schema-based data model to map Web data instances into or imports an existing one such as RSS.
2. As a second step, the application designer *visually records* a Web macro filling forms and traversing to the desired result page. The system protocols the actions on an action-based level, i.e. it does not rely on the server request/response, but identifies XPath elements based on user clicks in the GUI and is capable of replaying all kind of user interactions, even for highly dynamic pages.
3. Finally, the application designer designs the *data extractor* for the result page where usually hierarchically defines the elements of interests. *Filters* are

created visually by choosing example instances and then refining the selection based on system generalizations. Internally, filters are mapped to Elog rules. Result instances are mapped to the defined data model and verified for their consistency.
4. Additionally, every action and filter can be *parameterized* to individual search and restriction values, which are provided as method parameters to the Web service requests.

In real-life scenarios such as the IMDB example these steps are close by intermingled, especially when extracting data from various interlinked pages. The IMDB wrapper comprises a number of data extractors on different kind of pages, and a complex navigation describing when to apply which extractor and action. After finishing the example-based wrapper generation, certain actions and steps are manually parameterized by the designer. First, the value that is inserted into the search form, and next if one or more movie titles shall be returned based on a particular query, and how many of its actors and how many photos. In this way, similar to the output model, an input model is defined comprising all parameters that can be adjusted in an instance of an IMDB wrapping process by a request. As a next step, the wrapper is deployed to the WPI server.

4.2 Lixto WPI Server

Transformation Server. Heterogeneous environments such as integration and mediation systems require a conceptual information flow model. The usual setting for the creation of services based on Web wrappers is that information is obtained from multiple wrapped sources and has to be integrated; often source sites have to be monitored for changes, and changed information has to be automatically extracted and processed. Thus, push-based information systems architectures in which wrappers are connected to pipelines of post-processors and integration engines which process streams of data are a natural scenario, which is supported by the Lixto Transformation Server [26,7]. The overall task of information processing is composed into stages that can be used as building blocks for assembling an information processing pipeline. The stages are to

- acquire the required content from the Web applications
- integrate and transform content from a number of input channels and tasks such as finding differences,
- interact with external processes,
- format and deliver results in various formats and channels and connectivity to other systems.

The actual data flow within the Transformation Server is realized by handing over XML documents. Each stage within the Transformation Server accepts XML documents (except for the wrapper component, which accepts HTML), performs its specific task (most components support visual generation of mappings), and produces an XML document as result. This result is put to the successor components. Boundary components have the ability to activate themselves

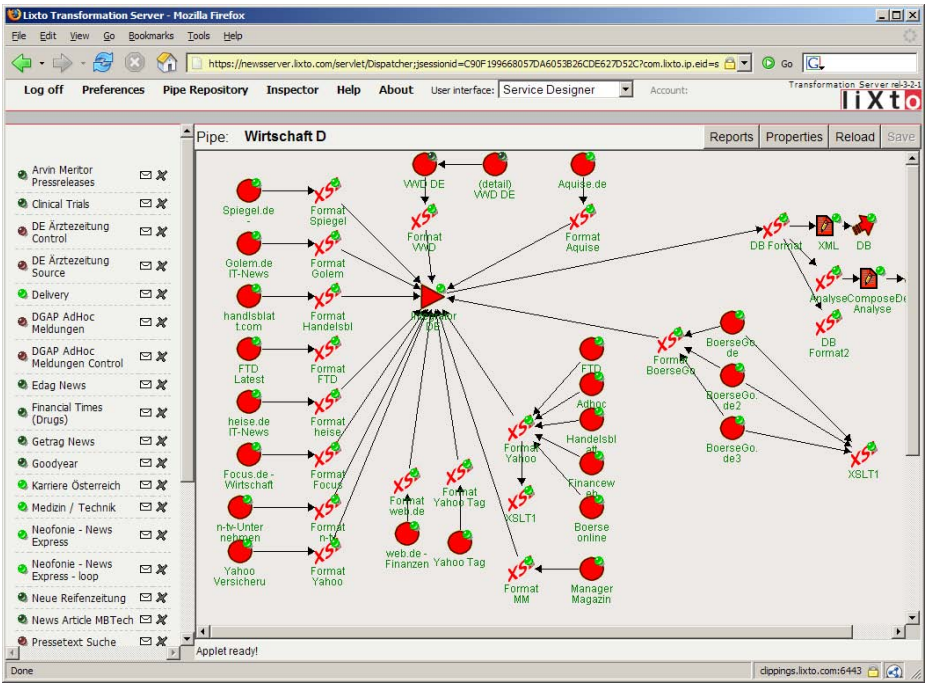

Fig. 6. Lixto Transformation Server

according to a user-specified strategy and trigger the information processing on behalf of the user. Figure 6 illustrates a complex example in the news domain.

From an architectural point of view, Lixto Transformation Server may be conceived as a container-like environment of information processing or as visually configured agent server. This "service flow" can model very complex unidirectional information flows. The usage of components also modularizes the information processing, so the service can be maintained and updated smoothly. Moreover, information services can be controlled and customized from outside of the server environment by various types of communication media such as Web services.

Extraction Server/Cluster. In simple scenarios, the Lixto WPI Server uses a single *Extraction Server*, where a number of extraction jobs can run in parallel. However, WPI scenarios with large number of services and users require a scalable extraction environment. It is crucial to be on the one hand very performant to support ad-hoc requests, and on the other hand to provide means for extreme scalability, especially in cases with a high peak load at certain times. Hence, data extractions can be executed via the *Extraction Cluster*. The WPI Server uses the Extraction Cluster [9] as directory service, asking for a free VD runtime head to be used in the next execution. The Extraction Cluster queues the request and assigns the best suited head, based on given weights. Furthermore, for ad

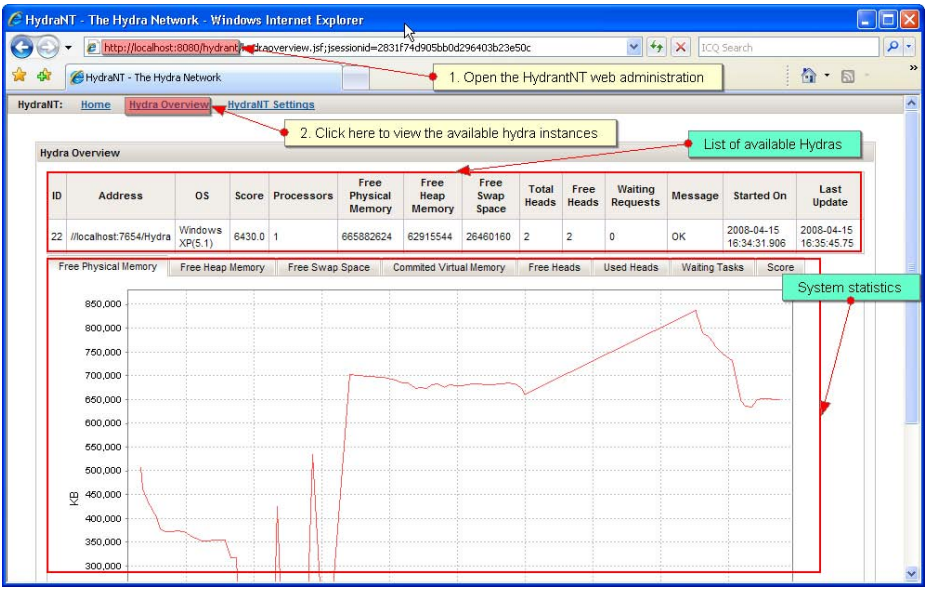

Fig. 7. Lixto Extraction Cluster

hoc requests, a priority queue is supported. Machines can be registered on the Extraction Cluster and inform it about the number of running VD heads and the machine parameters.

The Extraction Cluster distributes the load and can invoke Extraction Servers from Cloud Services such as the Amazon Elastic Cloud if the load gets too high. A screenshot of a simple status inspection is shown in Figure 7.

Lixto WPI Server Users and Registry. In Web Process Integration scenarios, we mainly distinguish two cases:

- *Scheduled Push Approach:* A service is configured to regularly push data to a particular component. The WPI Server handles the schedule and delivers results e.g. to a database or an e-mail address.
- *Ad-Hoc Pull Approach:* A service is configured to return data on demand. A Web Service interface is exposed that drives the service and executes it based on a given request. Data is returned e.g. as SOAP response or as REST.

Lixto WPI server distinguishes different user roles, the most prominent being the service designer and the service user. The service designer composes a service, including the definition of a wrapper, specification of transformation rules, how to integrate results if multiple wrappers are used, and how to deliver information. Service Designers publish services that are allowed to be consumed by Service Users.

Service users use the MyLixto GUI to browse the service registry and pick interesting services. A service user can choose subscribe to a service, which

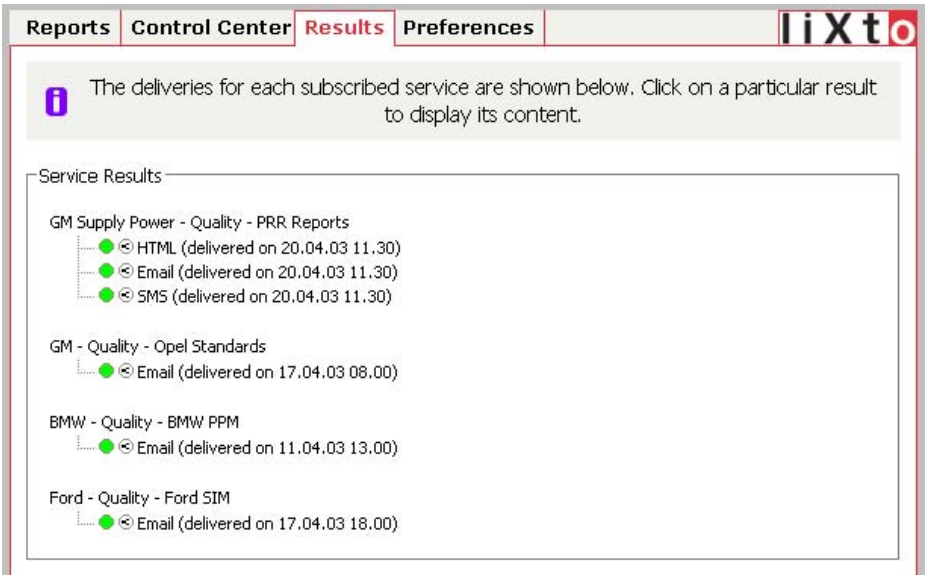

Fig. 8. Consuming the WPI Service Registry with MyLixto

regularly runs in her name and with her given parameters, and provides the information e.g. through e-mail. Please refer to Figure 8 as an example. Alternatively, users can choose to receive the data immediately, triggering an execution on the WPI server. The first approach is primarily used in corporate scenarios where employees need to be informed regularly, whereas the second is usually used in meta-search scenarios and on-demand mashup applications (refer to Section 5).

4.3 Web Service Delivery

Figure 9 illustrates the usage of the WPI server service registry. The service registry shows all available services (company-internal and public, respectively). During service creation, the service designer chooses which query methods for a service will be available and how to map wrapper and service parameters to methods and method parameters [10].

After picking a service, the service user is shown all available methods to a service. E.g., in the IMDB case, the following methods can be exposed:

- `getSingleMovieDescription(String title)`
- `getAllMovieTitles(String searchtext)`
- `getActorDataForMovie(String title, boolean photos)`
- `getActorCharacterRelationForMovie(String title)`

As Figure 9 illustrates, there are different ways to use the service registry. Users can either use the Service User GUI (MyLixto) for triggering or subscribing to a

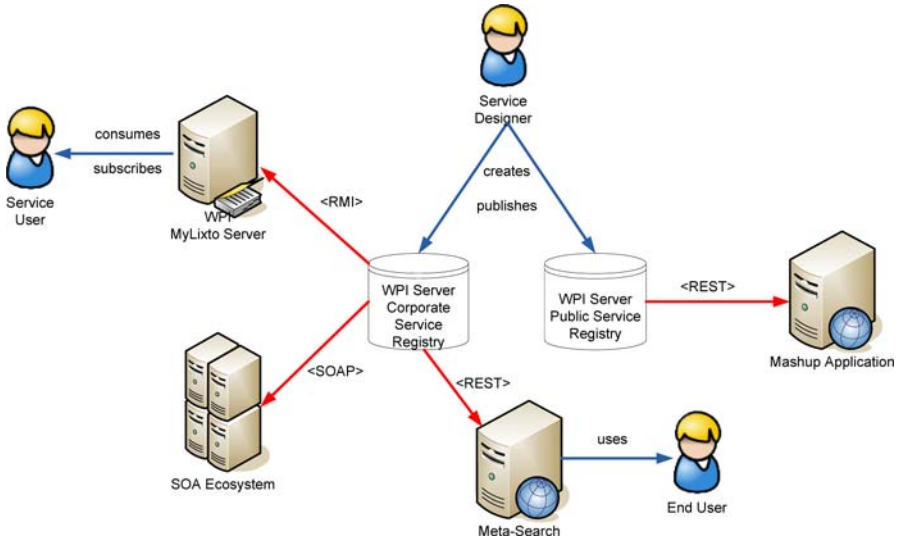

Fig. 9. Consuming the WPI Service Registry

service, or connect with their favorite Web Service client asking for the WSDL and sending a SOAP request (usually happening when the request is embedded into a larger SOA ecosystem), or if the service has been made available to the public, use a simple REST request specifying the parameters in the URL.

5 Application Areas and Future Research Issues

5.1 Sample Application Areas

Web Process Integration in the Automotive Industry. Many business processes in the automotive industry are carried out by means of Web portal interaction. Business critical data from various divisions such as quality management, marketing and sales, engineering, procurement, supply chain management, and competitive intelligence has to be manually gathered from Web portals and Websites. By automation, automotive part suppliers can dramatically reduce the cost associated with these processes while at the same time improving the speed and reliability with which these processes are carried out. The Automotive business case is described in more detail in [11]. In this scenario, wrapper technologies act as enabling technology for Service Oriented Architectures and are one crucial puzzle piece in Enterprise Application Integration and B2B process integration.

End User Mashups. Today, leading software vendors start to provide mashup platforms (such as Yahoo! Pipes or Lotus Mashups). A mashup is a Website or Web application that combines a number of Web sites into an integrated

view. Usually, the content is taken via APIs, embedding RSS or Atom Feeds in a REST-like way. Wrapper technology leverages legacy Web applications to light-weight APIs such as REST that can be integrated in mashups in the same fashion. Web Mashup Solutions no longer need to rely on APIs offered by the providers of sites, but can extend the scope to the whole Web. In particular, the deep Web gets accessible by encapsulating complex form queries and application logic steps into the methods of a Web Service. In this scenario, wrapper technologies help enable the Web of Services, built on legacy Web sites. End users are put in charge to create their own views of the Web and embed data into other applications, usually in a light-weight way. This results in "situational applications", possibly unreliable and unsecure applications that however help to solve an urgent problem immediately.

Vertical Flight Search and Booking. Vertical Search is a special-purpose meta-search scenario for integrating deep Web data behind complex query interfaces and providing intelligent services to customers. Typical application scenarios are domain-specific searches with complex Web query interfaces (refer to [27] for a description how Web forms can be formally modeled), such as finding the cheapest flight over several airlines within a specific date range or the cheapest computer on various channels. Meta-Search applications have an inherent workflow logic, due to the need of querying a number of different portals and understanding dependencies when to query which Web site; e.g. querying a weather site for a particular city in a multi-hop flight scenario where first the multi-hop stops have to be extracted and understood, and next additional data for such cities is queried. Furthermore, since users do not like to wait more than a couple of seconds for results, there is the absolute need to provide results as soon as they are extracted – this logic is encapsulated in a set of Web service requests and responses. A meta-search process comprises the workflow which Web sources to query and providing input paramters to them, as well as the understanding and modeling of the Web application logic. This includes complex bi-directional processes, e.g. in cases where a booking process is re-packaged in a Meta-Search application. In such cases, interception points are required during the wrapping process.

5.2 Future Challenges

Turning Web Applications and Web Sites to Web Services is an important contribution to the search computing paradigm. Due to understanding of deep Web applications and parameterizing the data extraction, focused search in the Deep Web can be realized.

Deep Web and Workflow Capabilities. In B2B application areas, key factors are workflow capabilities for the whole process of data extraction, transformation and delivery, capabilities to treat all kinds of special cases occurring in Web interactions, and excellent support of the latest Web standards used during secure transactions.

As Web pages are becoming increasingly dynamic and interactive, efficient wrapping languages have to make it possible to record, execute and generalize macros of Web interactions and, hence, model the whole process of workflow integration. An example of such a Web interaction is a complicated booking transaction. Future research issues also include the different approach of targeted deep Web crawling as an alternative to Web application flow modelling.

To query deep Web forms, wrappers have to learn the process of filling out complex Web search forms and the usage of query interfaces. Such systems have to learn abstract representation for each search form and map them to a unified meta form and vice versa, taking into account different form element types, contents and labels.

Extraction Capabilities. Whereas Web wrappers today dominantly focus on either the flat HTML code or the DOM tree representation of Web pages, recent approaches aim at extracting data from the CSS box model and the visual representation of Web pages [21]. This method can be particularly useful in recent times where the DOM tree does not accurately reflect how the user perceives a Web page.

One other challenge is Generic Web Wrapping. On the one hand this includes to evolve from site-specific wrappers to domain-specific wrappers by using semantic knowledge in addition to the structural and presentational information available. On the other hand, however, it is essential that wrappers still are sufficiently robust to provide meaningful data. Hence, techniques for making wrappers more robust and automatically adapt wrappers to new situations will contribute to this challenge.

Key factors in the area of mashup scenarios include efficient real-time extraction capabilities for a large number of concurrent queries and detailed understanding of how to map queries to particular Web forms.

6 Conclusions

In this paper we reviewed techniques and tools for Web data extraction. We first discussed a number of tools and then focused on one particular example, the Lixto tool which is able to overcome most of these obstacles. We presented the two main components of Lixto: (1) The Lixto Visual Developer, which allows a wrapper designer to visually and interactively develop a wrapper for a Website; and (2) the Lixto Web process Integration Server (WPI Server) that enables one to quickly design an interface between complex Web processes and corporate software. In particular, we showed how Lixto can be used to transform Web pages and deep Web sources into Web services, and how massive amounts of data can be delivered into applications by means of Web process integration. The latter aspect of Web data extraction is of particular relevance to the achievements of service marts, as elaborated in Chapter 9 of this book.

We showed, based on the example of Lixto, how software of a new type can fill an important gap in information technology. While most current obstacles

are addressed and satisfactorily solved by Lixto, the Web is moving on, and new challenges emerge. Some of these challenges were described in Section 5. Other important challenges regard the intelligent and efficient querying of Web services, and the fully automatic generation of wrappers for restricted domains such as real estate, and so on. The first challenge is currently being tackled by the SeCo project. The second challenge is tackled by the DIADEM at Oxford University.

References

1. Adelberg, B.: Nodose - a tool for semi-automatically extracting structured and semistructured data from text documents. In: SIGMOD Record, pp. 283–294 (1998)
2. Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 337–348. ACM, New York (2003)
3. Arocena, G.O., Mendelzon, A.O.: Weboql: restructuring documents, databases, and webs. Theor. Pract. Object Syst. 5(3), 127–141 (1999)
4. Baumgartner, R., Ceresna, M., Ledermüller, G.: Deep web navigation in web data extraction. In: Proc. of IAWTIC (2005)
5. Baumgartner, R., Flesca, S., Gottlob, G.: Declarative Information Extraction, Web Crawling and Recursive Wrapping with Lixto. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, p. 21. Springer, Heidelberg (2001)
6. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: Proc. of VLDB (2001)
7. Baumgartner, R., Herzog, M., Gottlob, G.: Visual programming of web data aggregation applications. In: Proc. of IIWeb 2003 (2003)
8. Baumgartner, R., Gatterbauer, W., Gottlob, G.: Web data extraction system. In: Encyclopedia of Database Systems (2009)
9. Baumgartner, R., Gottlob, G., Herzog, M.: Scalable web data extraction for online market intelligence, vol. 2, pp. 1512–1523 (2009)
10. Baumgartner, R., Gottlob, G., Herzog, M., Slany, W.: Interactively Adding Web Service Interfaces to Existing Web Applications. In: Proc. of SAINT (2004)
11. Baumgartner, R., Herzog, M.: Using Lixto for automating portal-based b2b processes in the automotive industry. International Journal of Electronic Business 2(5), 519–530 (2004)
12. Blanco, L., Crescenzi, V., Merialdo, P., Papotti, P.: Flint: Google-basing the web. In: EDBT 2008: Proceedings of the 11th international conference on Extending database technology, pp. 720–724. ACM, New York (2008)
13. Cafarella, M.J., Halevy, A., Wang, D.Z., Wu, E., Zhang, Y.: Webtables: exploring the power of tables on the web. Proc. VLDB Endow. 1(1), 538–549 (2008)
14. Cafarella, M.J., Ré, C., Suciu, D., Etzioni, O., Banko, M.: Structured querying of web text: A technical challenge. In: CIDR (2007)
15. Crescenzi, V., Mecca, G.: Grammars have exceptions. Inf. Syst. 23(9), 539–565 (1998)
16. Crescenzi, V., Mecca, G.: Automatic information extraction from large websites. J. ACM 51(5), 731–779 (2004)

17. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: Towards automatic data extraction from large web sites. In: VLDB 2001: Proceedings of the 27th International Conference on Very Large Data Bases, pp. 109–118. Morgan Kaufmann Publishers Inc., San Francisco (2001)
18. Embley, D.W., Campbell, D.M., Jiang, Y.S., Liddle, S.W., Lonsdale, D.W., Ng, Y.k., Smith, R.D.: Conceptual-model-based data extraction from multiple-record web pages. Data and Knowledge Engineering 31, 227–251 (1999)
19. Etzioni, O., Banko, M., Soderland, S., Weld, D.S.: Open information extraction from the web. Commun. ACM 51(12), 68–74 (2008)
20. Freitag, D.: Information extraction from html: Application of a general machine learning approach. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 517–523 (1998)
21. Gatterbauer, W., Bohunsky, P., Herzog, M., Krüpl, B., Pollak, B.: Towards domain-independent information extraction from web tables. In: Proc. of WWW, May 8-12 (2007)
22. Gottlob, G., Koch, C.: Monadic Datalog and the Expressive Power of Web Information Extraction Languages. Journal of the ACM 51(1) (2004)
23. Hammer, J., McHugh, J., Garcia-Molina, H.: Semistructured data: The tsimmis experience. In: Proceedings of the First East-European Workshop on Advances in Databases and Information Systems, ADBIS 1997, pp. 1–8 (1997)
24. He, B., Chang, K.C.-C.: Statistical schema matching across web query interfaces. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 217–228. ACM, New York (2003)
25. He, B., Zhang, Z., Chang, K.C.-C.: Towards building a metaquerier: Extracting and matching web query interfaces. In: International Conference on Data Engineering, pp. 1098–1099 (2005)
26. Herzog, M., Gottlob, G.: InfoPipes: A flexible framework for M-Commerce applications. In: Proc. of TES workshop at VLDB (2001)
27. Holzinger, W., Krüpl, B., Baumgartner, R.: Automated ontology-driven metasearch generation with metamorph. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) WISE 2009. LNCS, vol. 5802, pp. 473–480. Springer, Heidelberg (2009)
28. Chang, C.h., Lui, S.-C.: Iepad: Information extraction based on pattern discovery, pp. 681–688 (2001)
29. Jurić, D., Banek, M., Skočir, Z.: Uncovering the deep web: Transferring relational database content and metadata to OWL ontologies. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part I. LNCS (LNAI), vol. 5177, pp. 456–463. Springer, Heidelberg (2008)
30. Kayed, M., Shaalan, K.F.: A survey of web information extraction systems. IEEE Trans. on Knowl. and Data Eng. 18(10), 1411–1428 (2006); Member-Chang, Chia-Hui and Member-Girgis, Moheb Ramzy
31. Knoblock, C.A., Lerman, K., Minton, S., Muslea, I.: Accurately and reliably extracting data from the web: a machine learning approach, pp. 275–287 (2003)
32. Kushmerick, N.: Wrapper induction: Efficiency and expressiveness. Artificial Intelligence 118, 2000 (2000)
33. Laender, A.H.F., Ribeiro-Neto, B., da Silva, A.S.: Debye - date extraction by example. Data Knowl. Eng. 40(2), 121–154 (2002)
34. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S., Teixeira, J.S.: A brief survey of web data extraction tools. SIGMOD Rec. 31(2), 84–93 (2002)

35. Lerman, K., Getoor, L., Minton, S., Knoblock, C.: Using the structure of web sites for automatic segmentation of tables. In: SIGMOD 2004: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 119–130. ACM, New York (2004)
36. Lerman, K., Minton, S.N., Knoblock, C.A.: Wrapper maintenance: a machine learning approach. J. Artif. Int. Res. 18(1), 149–181 (2003)
37. Liu, L., Pu, C., Han, W.: Xwrap: An xml-enabled wrapper construction system for web information sources. In: ICDE, pp. 611–621 (2000)
38. Raposo, J., Pan, A., Alvarez, M., Hidalgo, J., Vina, A.: The Wargo System: Semi-Automatic Wrapper Generation in Presence of Complex Data Access Modes. In: Proceedings of DEXA 2002, Aix-en-Provence, France (2002)
39. Riloff, E.: Automatically constructing a dictionary for information extraction tasks. In: Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 811–816. MIT Press, Cambridge (1993)
40. Sahuguet, A., Azavant, F.: Building intelligent web applications using lightweight wrappers. Data Knowl. Eng. 36(3), 283–316 (2001)
41. Shen, W., Derose, P., Vu, L., Doan, A., Ramakrishnan, R.: Source-aware entity matching: A compositional approach. In: IEEE 23rd International Conference on Data Engineering, ICDE 2007, pp. 196–205 (2007)
42. Shen, W., DeRose, P., McCann, R., Doan, A., Ramakrishnan, R.: Toward best-effort information extraction. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1031–1042. ACM, New York (2008)
43. Shen, W., Doan, A., Naughton, J.F., Ramakrishnan, R.: Declarative information extraction using datalog with embedded extraction predicates. In: VLDB 2007: Proceedings of the 33rd international conference on Very large data bases, pp. 1033–1044. VLDB Endowment (2007)
44. Soderland, S., Cardie, C., Mooney, R.: Learning information extraction rules for semi-structured and free text. Machine Learning, 233–272 (1999)
45. Soderland, S., Fisher, D., Aseltine, J., Lehnert, W.: Crystal: Inducing a conceptual dictionary. In: Mellish, C. (ed.) Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 1314–1319. Morgan Kaufmann, San Francisco (1995)