Scalable Web Data Extraction for Online Market Intelligence

Robert Baumgartner
Lixto Software GmbH
Vienna, Austria
baumgartner@lixto.com

Georg Gottlob[*]
Oxford University
Oxford, UK
gottlob@comlab.ox.ac.uk

Marcus Herzog
Lixto Software GmbH
Vienna, Austria
herzog@lixto.com

ABSTRACT

Online market intelligence (OMI), in particular competitive intelligence for product pricing, is a very important application area for Web data extraction. However, OMI presents non-trivial challenges to data extraction technology. Sophisticated and highly parameterized navigation and extraction tasks are required. On-the-fly data cleansing is necessary in order two identify identical products from different suppliers. It must be possible to smoothly define data flow scenarios that merge and filter streams of extracted data stemming from several Web sites and store the resulting data into a data warehouse, where the data is subjected to market intelligence analytics. Finally, the system must be highly scalable, in order to be able to extract and process massive amounts of data in a short time. Lixto (www.lixto.com), a company offering data extraction tools and services, has been providing OMI solutions for several customers. In this paper we show how Lixto has tackled each of the above challenges by improving and extending its original data extraction software. Most importantly, we show how high scalability is achieved through cloud computing. This paper also features a case study from the computers and electronics market.

1. INTRODUCTION

This paper deals with Web data extraction technology applied to online market intelligence (OMI). More specifically, we report about how the Lixto Web data extraction technology [4] has been improved over the last few years in response to scalability and efficiency challenges posed by customers for which the Lixto company (lixto.com) has been carrying out advanced online market intelligence projects.

Online market intelligence. According to [21], "*Market Intelligence (MI)* is the information relevant to a company's markets, gathered and analyzed specifically for the purpose of accurate and confident decision-making in determining market opportunity, market penetration strategy, and market development metrics." Market intelligence comprises as a special case competitive intelligence. *Online market intelligence (OMI)* covers all aspects of MI that are related to online information sources, predominantly, to the Web. Given that most information on pricing, product availability, store locations, and so on, is available on the Web, OMI is becoming the most important form of market intelligence. Currently, almost every large retail company has OMI needs for marketing and pricing. Most, but not all applications of OMI are in the area of competitive intelligence. Here are five examples of typical OMI needs. The first three are in the area of competitive intelligence, while the last two are not.

- An electronics retailer would like to get a comprehensive overview of the market in the form of a dashboard displaying daily information on price developments including shipping costs, pricing trends, and product mix changes by segment, product, geographical region, or competitor.
- A supermarket chain wishes to be continually informed about their competitors' product prices. Moreover, they want to be immediately informed in case a competing supermarket chain issues a special offer or promotion. They need to react very quickly to price changes or new special discounts in order to maintain their competitive position. They also want to be informed as soon as new products show up on the market.
- An online travel agency offering a best price guarantee needs to know at which prices the packages they offer are sold over the Web by competing travel agencies. Moreover, they wish to be informed about the average market price of each travel product they feature.
- A road construction corporation would like to be informed about new public tenders in a dozen of different countries or states.
- Once a week, the house price index (reflecting the average prices for various property types) of a country is updated on the Web site of the country's national statistics agency. Variations of this house price index immediately trigger changes in the value of shares of various industries related to the real estate market. A hedge fund wishes to anticipate the house price variations by one day. To achieve this, the hedge fund

[*]Computing Laboratory and Oxford Man Institute of Quantitative Finance.

needs to obtain online market intelligence by aggregating data such as asking price, number of listings and average time on market, from the web pages of real-estate agents.

What all these applications have in common is that they require massive amounts of Web data from a relatively limited number of sources and that they are rather time critical. Moreover, in several cases aggregate data rather than raw data it needed.

OMI and Web data extraction. It is quite obvious that the main approach for solving OMI problems such as those mentioned is the use of *Web data extraction technology* [5]. Moreover, given that the relevant sources are usually known, and that the OMI applications require a complete coverage of the sources and a very high level of data precision, manual or semi-automatic wrapper generators for the relevant sources are preferable to generic Web harvesting tools. The latter may be used as additional tools, for example, in order to discover new sources. In the present paper we focus on the former type of technology. The principles of (manual and semi-automatic) Web data extraction technology and wrapper generation are described in [5]. A survey of various wrapper generation tools, that still covers most existing tools can be found in [15, 20].

However, standard data extraction technology alone does not suffice for addressing advanced OMI needs such as: (i) the possibility of defining sophisticated and highly parameterized navigation and extraction tasks; (ii) the possibility of performing data cleansing tasks, e.g. to identify identical products from different suppliers; (iii) features for smoothly defining data flow scenarios that merge and filter streams of extracted data stemming from several Web sites and store the resulting data into a data warehouse; (iv) means for carrying out market intelligence analytics and for presenting the results in form of appropriate dashboards to the users; (v) features enabling the highly scalable extraction and further processing of massive amounts of data in a short time.

Lixto (www.lixto.com), a company offering data extraction tools and services, has been providing OMI solutions for several customers. In this paper we show how Lixto has tackled each of the above challenges by improving and extending its data extraction software whose first version was presented at VLDB 2001. The resulting new software product *Lixto OMI* is based upon the company's web data extraction technology that has been designed to access, augment and deliver content and data from web applications that utilize client-side processing techniques such as JavaScript, AJAX and dynamic HTML. This allows for fast reactions to changes in the online channels. On top of the data store, Lixto uses enterprise-class reporting infrastructure to provide all necessary reports and analytics to enable an efficient identification of market opportunities that are most relevant for day-to-day business. Important market events are highlighted and reports customized to show exactly the data items that are of most interest to individual users. This latest version of the Lixto OMI solution utilizes the power of Internet Cloud Computing to improve the scalability and processing power of the companys SaaS and on-premise web intelligence solution that will allow Lixto's customers to extract even more data from the Internet for analysis in real-time.

To achieve this virtually infinite scaling of its solution, Lixto has introduced a new software architecture into the latest Version 5 solution for its server-side products that supports a grid-based scaling (or Cloud Computing) approach. This allows for linear scaling in relation to the number of processing units available in a cluster of servers. The software supports the use of Cloud Computing resources such as Amazon Elastic Compute Cloud (EC2). The new server architecture also introduces intelligent load-distribution amongst the available processing units resulting in more efficient resource utilization and reduced management overhead.

Structure of the paper. The rest of the paper is structured as follows. In Section 2, we briefly describe the four major OMI application components. In Section 3 we survey the processes of creating a wrapper, of defining the application data flow, and of data cleansing. The overall process is referred to as *Web ETL*, i.e., extraction, transformation, and loading of Web data. In Section 4 we describe how extreme scalability is achieved through a cluster architecture and through cloud computing. Section 5 presents a case study of an application in the computers and electronics market. Finally, some brief concluding remarks are given in Section 6.

2. OMI APPLICATION COMPONENTS

In online market intelligence, the web data extraction toolkit is used to set up an analytic practice for an in-depth analysis of online markets. Typical web sources such as online shops and other sorts of online distribution channels are used to extract relevant product information including product configurations and prices. This information is subsequently fed into the analysis platform to provide a structured view of the market. Online market intelligence helps market participants achieve a better understanding of important market events directly from the source.

Online market intelligence applications comprise the following components: First, a web data extraction unit to access the relevant online market data. This component needs to query web sites and transform the returned unstructured data format, i.e. a web page, into a structured format that can be used for automatic data processing (refer to [9, 15, 16, 17, 19] for an overview of academic, commercial and open-source wrapper-generation software). In the context of online market intelligence, this component needs to scale out well for processing a high number of pages typically in the range of few hundred thousand per day. We call such a component a Web Connector and will detail the specifics later on.

Secondly, a data cleansing component that performs data cleansing and record linkage between data sets from different web sources. This is necessary as data from the web is usually not in a homogeneous format, e.g., naming and formatting of both names and values of data objects will differ substantially. However, to compare data from different sources a common standard needs to be implemented. This is achieved by providing mapping rules that map the input data set onto a common output data model.

Thirdly, an analytics package for digging into the data and for setting up reports and dashboards that enable users to experience the value hidden in the data. The data is thus loaded into a data warehouse and existing business intelligence toolkits can be used to perform further analysis. The real challenge of this part is to understand the domain and the benefits associated with certain analytics.

Fourthly, to use the dataset in other enterprise applications such as yield management, pricing, or revenue management, connectors to these systems are required. Directly importing the market data into these applications provides additional value beyond the mere analytical aspect and allows for closing the loop, i.e., market data can drive the pricing process which sets prices for offerings on the market. Without market data, pricing has to rely on insufficient information leaving out an important aspect - the market.

3. WEB ETL MODELLING

In this section we describe the creation of Web Connectors. A Web Connector comprises means to extract data from Web applications, transform and clean the extracted data, and load the data via a staging database to a data warehouse (DWH). Hence, in DWH notation using a Web connector can be referred to as an ETL process. Lixto provides visual and interactive tools for the creation of Web ETL connectors.

In particular, the *Lixto Visual Developer* is used for creating an extraction program ("wrapper") that understands the logic of the Web application and extracts facts from it. The *Lixto Transformation Server* is used for configuration of OMI scenarios that extract data from various Web sites, iterating over Web forms and managing extraction jobs. Finally, the *Lixto Retail Solution* harmonizes extracted data, performs product matching and provides the analytic data structure on which reports are created.

3.1 Wrapper Creation

3.1.1 Architecture

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. In the retail case, this comprises product names, prices, EAN numbers and several product properties. Such a data model is a shared entity throughout a project. It ensures on the one hand smooth collaborative work and on the other hand a number of used robustness algorithms rely on properties of the data model.

Figure 2 shows 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, and on the right hand side the DOM tree of the currently active page is displayed. At the bottom, in the Property View, navigation and extraction actions can be inspected and configured. Moreover, in the right corner the network traffic is shown.

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, generalization heuristics, domain-independent and retail-specific templates. The application designer creates the wrapper based on samples, both in the

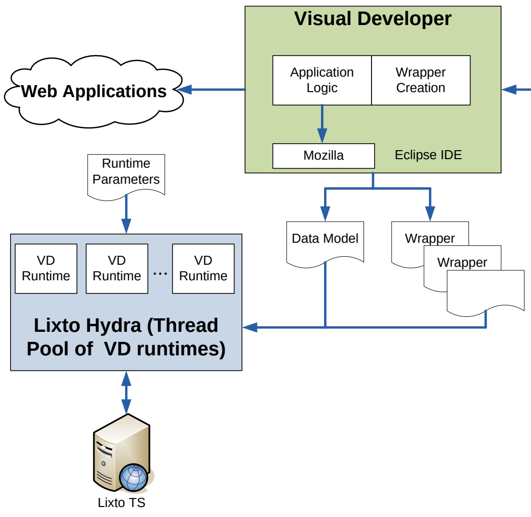


Figure 1: Visual Developer Architecture

case of navigation steps (e.g. use a particular product category) and in the case of extraction steps.

The internal extraction language *Elog* [3, 11], the web object detections 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, as well as technical aspects such as authorization and dialogue handling.

Wrappers and data models are uploaded to the server. In the retail scenario, the OMI Edition of the Lixto Transformation Server (TS) is used (rf. to Section 3.2.1 below). In the SOA-oriented architecture of Lixto, servers such as the TS access the VD Runtimes via Web Service or Java RMI. The simple variant is to use one machine with a pool of runtime components, as depicted in Figure 1. In Section 4, the highly scalable Extraction Cluster is presented. 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 its own process, using its own browser instance (during such executions the browser GUI is irrelevant and suppressed). *Lixto Hydra* spawns a number of VD heads and communicates results back to the server. Additionally, since Web applications can act unreliably, Hydra 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.

3.1.2 Web Processes and Deep Web Navigation

One important aspect of data extraction from the Web, often neglected in academic work (except e.g. in [1]), is how to reach pages where data is extracted from. Today's Web portals are no longer a collection of linked pages, but are rich internet applications (RIA) with a complex application logic. Therefore, Lixto VD is capable to:

- support deep Web navigation;

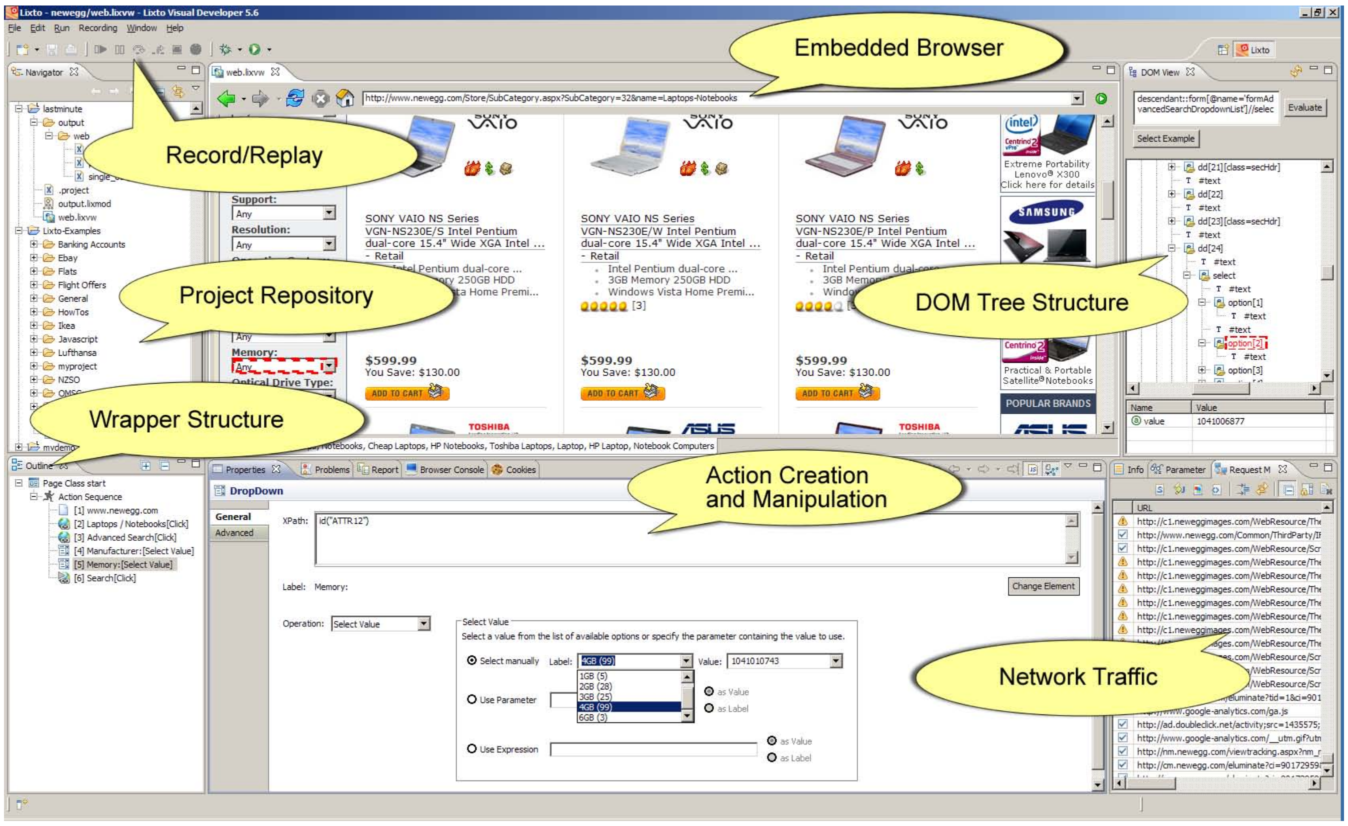


Figure 2: Visual Developer GUI

- behave like a human user and issue clicks on web objects instead of simple request-response tracking;
- understand the application logic of the Web site and model possible cases, using a process flow language;
- support pages that use asynchronous requests (Ajax), in particular support to freeze the current state of a page to do data extraction upon.

The navigation and process flow language of the VD was first described in [2]. In VD, a wrapper comprises a list of *page classes*. A page class is a template that contains a procedure of actions and default responses. Two primitive actions are *Mouse Actions* and *Key Actions*. Mouse actions include mouse move and mouse click elements, whereas key actions enter textual data in fields. Usually, both actions identify Web objects using XPath statements (rf. to Section 3.1.4 for explanations how these are generated in a robust way).

VD offers two variants of executing these actions.

- Visually: On replay at execution time, the web object referred by the given XPath is fetched and its associated coordinates in the view are computed. In consequence, a mouse click is performed on this element (by default in the center, except for image maps, where the click is executed at precisely the same location within the image at which it was recorded during wrapper generation).
- Event-based: The corresponding DOM Events are triggered.

The first approach has the advantage that the wrapper behaves exactly like a human user, as it leaves all the logic of interpreting the clicks to the browser. Based on these primitive actions, various Web objects are supported by tailor-made actions, such as: Buttons, Single and Multiple Select Boxes, Dropdown List Boxes, Checkboxes, Radio Button Groups, and Text Boxes. Each action supports an individual set of operations.

As example, consider the Text Box Action, which supports the following operations: *Click*, *Click if exists*, *Submit*, *SetValue*, *AppendValue*, and *GetValue*. *GetValue* stores the current value in a variable, and *SetValue* sets a value given by the application designer or an external parameter. Another example are Drop Down Actions. These actions concatenate mouse clicks to select the list plus the actual value selection. Figure 2 illustrates the definition of such an action in the GUI of the VD.

VD supports *automatic recording* of deep Web navigation sequences by offering buttons to record, pause and stop recording. During a recording process, all browser interactions of wrapper designers are stored by the system and the respective actions are created. As well as mouse and key interactions, this includes switching to popups and creating default answers to prompt dialogues (e.g. passwords for basic authentication). During the recording process, the system generalizes how Web objects will be detected during replay (as discussed in Section 3.1.4).

Popups are treated like new browser tabs, and in the *Switch* Action, a wrapper designer can decide to switch to the first or last opened one, or to identify a tab by name,

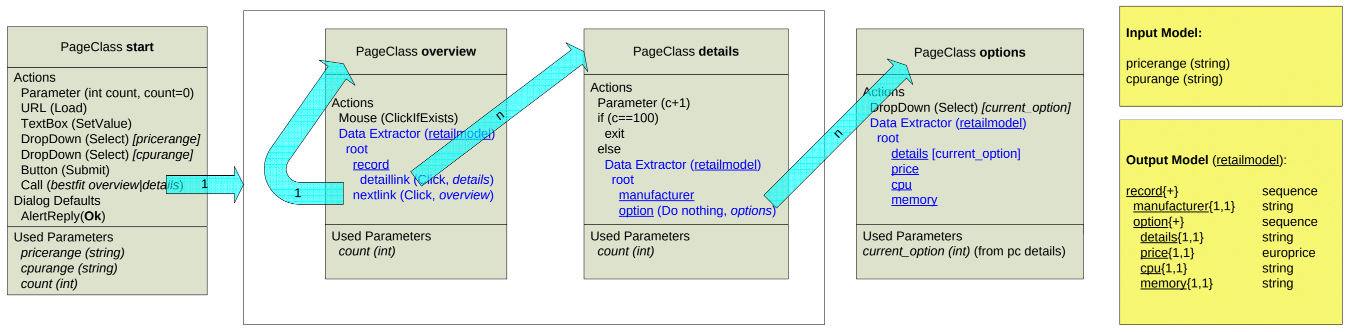


Figure 3: Page Class Concept and Input/Output Model

title or an occurring object. Prompt replies such as basic authentication fields, JavaScript alerts, proxy password dialogues and similar are managed separately to actions. They are not applied at one particular step, but instead default responses are stored for each page class. This way, if an alert occurs in a rather random fashion, the wrapper always knows how to handle it. *Download* Actions store files in the result XML or on disk.

Further actions enable the wrapper designer to execute arbitrary JavaScripts on the current DOM tree, and to configure all kind of browser settings. The latter includes setting the user agent, turning images on/off, applying content policies (such as disable JavaScript or loading certain URLs), and execution of external tools for special MIME types.

In many Web applications it is necessary to understand the application logic, and hence to distinguish cases that can happen during navigation, to handle exceptions, and to branch to a multitude of detail pages. Hence, the VD supports the following process flow concepts:

- *If* and *While Actions* allow comparison based on comparisons in XPath or scripting languages.
- *Call Actions* give the possibility to explicitly jump to a different page class.
- *Return* and *Exit* break a page class execution and return to the previous one resetting the previous browser state, and exit the wrapper execution, respectively.
- *Try-Catch-Finally* blocks handle exceptional cases, e.g. by carrying out special action in case a mouse action on a Web object can not be performed.
- *Page Class Recursions* are a concept applied in data extractors (described in the following section) to branch to a new page and call it x times. A typical case is an overview page where a number of items are described, each featuring a detail link. The data extractor detects all links and calls an appropriate page class for each of these links. At the very end of page class execution, the execution returns to the page class where it was called.
- *Parameter Actions* create new or modify existing parameters. Parameters can be external input values, created by actions, or populated by extracted values. Each action can use parameter values at all places, e.g. for filling textual values or XPath fragments.
- *DB Loop Actions* receive a list of values that are used to execute further page classes with those values as parameters.

Figure 3 gives a conceptual view on a wrapper, and illustrates some of the above process flow actions. It illustrates the input parameters to a wrapper, the output model that the data is mapped to, and the wrapper itself. This example wrapper features four page classes. Page class "start" is called from another process (such as the Lixto Server), providing values for the parameters given in the input model. The actions to set a global counter and to select values from Dropdown lists are executed sequentially. In case an alert popups throughout, it is answered with the "Ok" button. Finally, the Call Action jumps to the page class "overview". The first action here is to perform a mouse click in case a particular advertisement page pops up. Next, data extraction on the current page is applied. In particular, links to detail pages and to the next page (if any) are extracted by invoking the data extractor (see Section 3.1.3), and followed. This illustrates that navigation steps and extraction steps are closely intermingled with each other rather than being two separate processes.

For all detail links, the detail page class is called. In this class, a counter is used to ensure that no more than 100 items are extracted, and manufacturer data is grabbed. In this case, several options/configurations for each item exist, and a further branching to the page class "Options" iterates through all of them. A parameter for the currently selected configuration is passed on to perform an option selection in the new page class.

In the page class "start", the Call Action does not explicitly refer to a page class, but jumps to the best fitting one. To perform this, the system creates a fingerprint of how typical entry pages look like for each page class. During execution, based on the properties, the best fit is chosen. The fingerprint contains both structural and syntactical properties, and can easily distinguish between typical overview list pages and detail pages with one item.

Wrappers can be executed fully or partially, in run or debug mode. The system highlights selections and offers a step-by-step walk-through, using breakpoints, and inspection of parameter states.

3.1.3 Web Data Extraction

A *data extractor* is a special action applied during the navigation flow. It freezes the DOM tree, and applies rules for extracting a nested data structure. Each wrapper comprises a hierarchical structure of *patterns*. Each pattern matches one kind of information on a Web page (e.g. prices). Patterns are structured hierarchically, and usually the extraction is created top-down. The application designer can decide to create patterns on the fly, or refer to existing model patterns; the being latter especially useful in case many wrappers have to map to the same data structure. On the right-hand side of Figure 3, an output model and its constraints are specified; in the data extractors of Figure 3 model patterns are underlined; by default only model pattern instances are reflected in the output.

The constraints given in the output model are used to verify if a data extractor is still extracting correctly; in case of deviations, alerts are sent to the operator of the application.

Each pattern comprises a set of filters. Application designers create filters in a visual and example-based way. Usually, first one example is selected directly in the browser, and generalized by the system. In the next step a wrapper designer can decide to impose further conditions, e.g. that particular attributes need to be present and match a regular expression, something is directly before, or an image is contained.

The *Lixto* data extraction process uses internally a declarative extraction language called *Elog* (described in more detail in [3, 4]), which relies on a datalog syntax and semantics and is completely hidden from the wrapper designer. It is ideally suited for representing and successively incrementing the knowledge about patterns described by designers. This knowledge is generated in an interactive process consisting of successive narrowing (logical *and*) and broadening (logical *or*) steps. *Elog* is flexible, intuitive and extensible. Moreover, the language is very expressive: In [11], the expressive power of a kernel fragment of *Elog* has been studied, and it has been shown that it captures monadic second order logic, hence is very expressive while at the same time easy to use due to visual specification. The Elog language operates on Web objects that are HTML elements, lists of HTML elements, and texts. Web objects can be identified based on internal, contextual, and range conditions and are extracted as so-called "pattern instances". Conditions are realized as predicates that occur in the Elog rule body. The usage of a declarative extraction language helps to reduce maintenance efforts and ensures better robustness, as usually only single patterns need to be exchanged in case of page changes.

One sample Elog rule (the one used in the data extractor in page class "overview" shown in Figure 3) is illustrated here:

```
record(X0, X1) :-
    root(_, X0), subelem(X0,
      (/html/body/div[3]/lixto:nearest(table)/tr,
      [("class", "midCol", substring)]), X1),
    before(X1, ../tr/td[2], [("text",
      "^Search.*", regexp)]), 0, 100, X2, X3).
```

The "record" 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.

Elog uses various means to detect web objects. In the productive system, XPath 2 and regular expressions are supported. In research prototypes, web objects can also be identified based on their visual rendition [10]. Further predicates can be used in an Elog rule to support semantic concepts, for instance to express that a value x is extracted only if also the "isCity(X)" predicate holds true for this value, and "isCapital(X,_)", i.e. a city is a capital of any country, is true.

Lixto VD offers an integration with GATE [8]: Wrapper designers can create tokenization patterns. These patterns use natural language processing techniques offered in the Information Extraction Module ANNIE of GATE. Wrapper designers select the concepts that are to be detected in a free-text block. For instance, the system extracts price values into the pattern "price" and creates child patterns "amount" and "currency" putting in the attribute values of this annotation.

On the extracted values, output rules can be applied to reformat values to standard formats and provide a first data normalization. A number of pre-defined output rules is offered: clean white-spaces, create checksums, store HTML fragment, map to standard date format, and replace characters, to name a few of them.

Although Lixto is primarily designed for data extraction from HTML, connectors to other data formats are supported. Wrapper designers can choose to call document converters for particular MIME types. This way, Excel, Word, CSV and PDF documents are supported. With PDF documents, our own document understanding techniques are applied to create an HTML document that is not optimized in display, but in structure [13]. For instance, PDF tables are understood and converted to HTML tables. Elog Rules operate on the generated HTML tables.

Extraction programs can communicate with databases, for instance, to check whether a description for a particular product is already available in the database or should be added during extraction.

3.1.4 Templates and Heuristics

One of the most important aspects of a data extractor is whether it can be trusted. Trust is earned by reliability, i.e. by being resilient against changes in Web applications. One way to reach this is to generate a robust XPath or regular expression, interactively and supported by the system. During wrapper generation, in many cases only one labelled example object is available, especially in automatically recorded deep Web navigation sequences. In such cases, efficient heuristics in XPath generation and fallback strategies during replay, are required. Typical heuristics during recording for identifying such single Web objects include:

- Generalization of a chosen XPath by using form properties, element properties, textual properties and formatting properties.

- DOM Structural Generalization – starting from the full path several generalized paths are created, using only characteristic elements and characteristic element sequences. A number of stable anchor points are stored from which relative paths to this object are created.

- Positional information is considered if the structurally generalized paths identify more than one element.

- Attributes and properties of elements are taken into account, in particular the element of choice, but we also consider ancestor attributes if the element attributes are not sufficient.
- Attributes that make an element unique are preferred, i.e. similar elements are checked for distinguishing criteria.
- Attribute Values are considered if attribute names are not sufficient. Attribute Value Fragments are considered if attribute values are not sufficient (using regular expressions)
- The ID attributes are used as far as possible. If an ID is unique and meaningful for characterizing an element it is considered in the fallback strategies with a high weight.
- Textual information and label information is used only if explicitly turned on (since this might fail in case of a language switch).

The output of the heuristic step is a set of XPath expressions, each identifying a particular web object. The heuristics are prioritized and are only used in the fallback strategies. The Fallback Strategy Algorithm runs through a number of steps to identify the best suited web object:

1. Depending on the type of element, an optimal XPath expression is chosen, relying on certain predefined properties to be used for a particular web element. For some web objects the identification is straightforward, and it is e.g. clear to first try the id, and if the id fails a structural generalization.
2. In case there are no or multiple matches for this XPath expression, further adaptations are considered. The system tries other heuristics generated during recording time, and searches for a single match. If this fails, the current XPath expression is used to generate new expressions (by invoking alternative choices according to Step 1).
3. The algorithm makes use of additional attributes and consider fragments of stored attribute values.
4. The algorithm chooses the Web object that matches most relative paths from the anchor points determined during recording.

In addition, for data extraction, machine learning techniques for learning Lixto extraction rules based on multiple labelled examples can be used. This compound filter learning strategy is described in [7].

Another aspect of robustness are changes in the application logic of a Web site. Due to the different operations (such as *ClickIfExist*), the conditional actions and the page class concept, this kind of robustness can be efficiently modelled by a wrapper designer.

Another advanced aspect of wrapper generation is the usage of templates that further optimize the time for wrapper generation and maintenance. Lixto VD supports both domain-independent and domain-specific templates. Domain-independent templates include wizards for tabular data and pre-defined page class structures for typical page types (overview pages with multiple records, detail pages with single records). Domain-specific templates extend this by using pre-defined models, such as for the retail domain. Additionally, grammars for retail-specific vocabularies have been generated. This domain knowledge is used to propose automatically created extraction rules, mainly based on tokenization patterns.

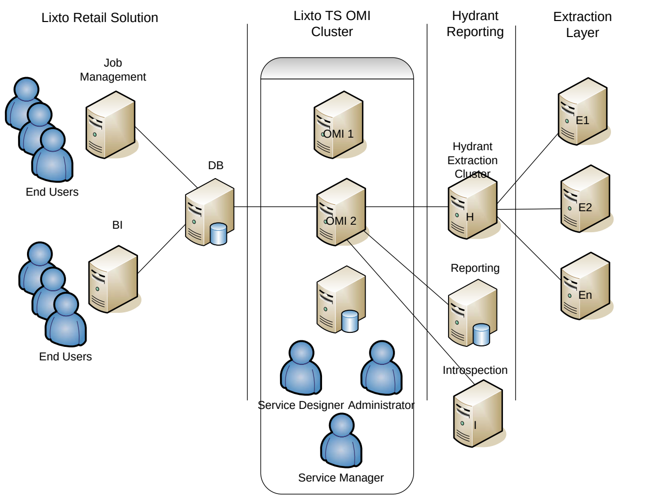


Figure 4: Lixto OMI Cluster

3.2 Data Flow Creation

3.2.1 Iteration Process Definition

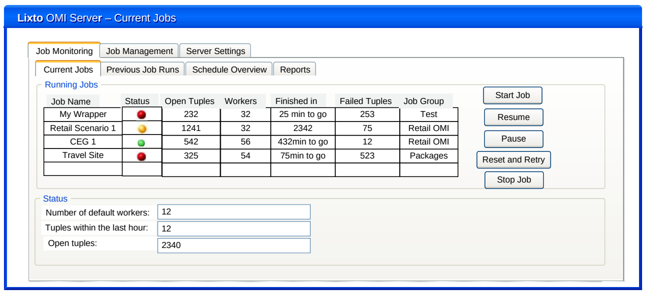
Lixto Transformation Server [14] is an enterprise-class execution environment for Web data extractors, based on Java Application Server technology. It is specifically designed to run Web data extraction services that need to perform sophisticated data transformation and process integration tasks. By utilizing a visual data-flow language, XML-based data streams can be set up in a short amount of time. Visually created mappings generate XSLT and XQuery code.

The Lixto Transformation Server OMI Edition is geared towards large online market intelligence scenarios. The focus of this edition is on extensive data collection efforts which need sophisticated control and monitoring capabilities. Data are continuously gathered and stored in databases. Figure 5 illustrates the architecture of a Lixto OMI Cluster.

Usually, the extraction job is generated by the Job Management Panel of the Lixto Retail Solution, either as a regularly scheduled job or as ad hoc job defined by an end user. Based on the given selection parameters, the system generates a list of tuples. Each tuple contains a set of parameters that is passed on to each or some of the wrappers. For example, when extracting data from the sites of a PC retailer, a tuple could include lower price limit, upper price limit, available memory and CPU type.

Lixto TS offers a Web-based GUI to create and configure OMI scenarios. The ingredients of an OMI scenario are

- a list of parameterizable wrappers;
- dataflows transformating results and mapping results to a database
- an extraction job containing a list of input tuples;



Job Name	Status	Open Tuples	Workers	Finished in	Failed Tuples	Job Group
My Wrapper	🔴	232	32	25 min to go	253	Test
Retail Scenario 1	🟡	1241	32	2342	75	Retail OMI
CEG 1	🟢	542	56	432min to go	12	Retail OMI
Travel Site	🔴	325	54	75min to go	523	Packages

Number of default workers:	12
Tuples within the last hour:	12
Open tuples:	2340

Figure 5: Lixto Introspection Console

- an extraction plan.

As a first step in the configuration, the application designer maps the input tuples to the appropriate sources. This includes parameter transformation, merging and splitting of parameters. Consider the wrapper given in Figure 2. For this, the parameters lower price limit and upper price limit are concatenated so that in the wrapper a particular price range is chosen in the Web form.

As a second step, the dataflow is generated. The default dataflow offered by the system performs no further data transformation and generates a default mapping to the result database. Application designers can choose to override the selection and create their own mappings.

Finally, the application designer creates the extraction plan. In particular, the designer defines the harvest cycle time, the number of tuples to expect, an initial average estimate about the processing time for each tuple, and a range of parallel workers processing the tuples. Furthermore, s/he defines when to apply retries and notification alerts.

The state of the iteration process is persisted, and can be inspected via the Introspection Console. Figure 5 sketches the Lixto Introspection Console. An overview on running jobs and processed tuples is given.

During execution, the system adapts the number of workers based on the average processing time, on job priority, and on the chosen cycle time. Due to interaction with the scalable Lixto HydraNT server, new resources for performing data extractions can be added on the fly (as described in Section 4).

3.2.2 *Data Cleansing Steps*

A number of data cleansing steps are already configured in the extraction process (e.g. in the output formatting functions of VD), especially with respect to normalization and tokenization to generate a canonical data representation. The Lixto Retail Solution supports configuration of domain-specific cleansing and record linkage. Record Linkage is the process of identifying that two record entries describe the same entity. In this context, record linkage is often referred to as product matching. Overviews on several techniques applied in data cleansing can be found in [6, 12].

The ETL process is shown in Figure 7. The Retail Solution operates on three databases: The result database is populated with Web data by Lixto TS. The retail database contains the harmonized and matched canonical data. Finally, data is leveraged to a data warehouse cube for convenient reporting. The data warehouse and the BI Reporting on top is described in Section 5.

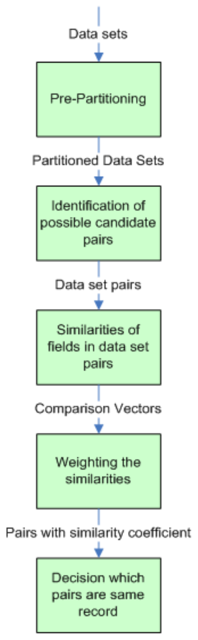

Data sets
Pre-Partitioning
Partitioned Data Sets
Identification of possible candidate pairs
Data set pairs
Similarities of fields in data set pairs
Comparison Vectors
Weighting the similarities
Pairs with similarity coefficient
Decision which pairs are same record

Figure 6: Data Cleansing Steps

The data cleansing steps are depicted in Figure 6. Mostly rule-based approaches are exploited. Configuration parameters of these steps can be derived beforehand by processing labelled sample data.

- *Normalization and Tokenization* (not depicted): Functions such as turn to lower case or resolve acronyms can be applied on attributes of records. As output, normalized records are returned.

- *Pre-Partitioning:* This phase is a preparation for the record linkage process. Data is grouped into partitions. Records within the same partition do not need to be compared if they denote the same entity (for instance, usually there are no duplicates within one portal).

- *Blocking:* Possible candidate pairs are created using efficient algorithms. This step outputs candidate pairs that are analyzed more closely in the next steps. The blocking phase reduces the quadratic complexity of having to compare each record to every other. We use a multi-pass sorted neighborhood algorithm: Several keys are created for sorting the data and moving a sliding window over the records. If the similarity of a scanned record is below the threshold of the previous one, a new block is created. After multiple passes, two records form a candidate pair if they occur together within at least one block.

- *Similarities of fields:* In this step, the similarity of data fields within compared records is computed. Either domain-specific comparisons or domain-independent string similarity metrics can be applied. In the training phase, application designers can find out which string distance metrics and thresholds are ideally suited, based on provided training data.

- *Weighting the similarities:* Based on *a priori* knowledge about the average rate of duplicates, the attributes

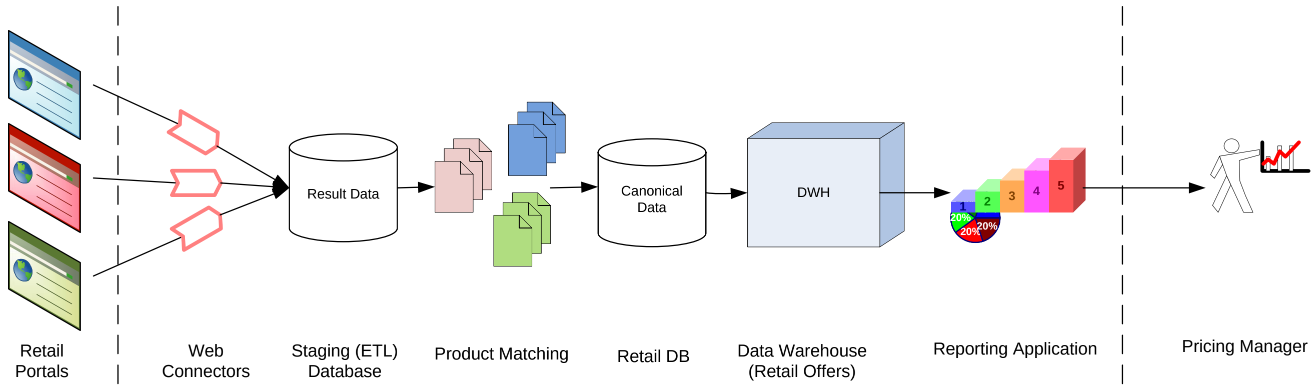


Figure 7: ETL Process and Product Matching

can be weighted to compute a value describing record similarity.

- The final step decides which records describe the same entities and data is fused together based on the paradigm that as little data as possible is lost during this process.

In some scenarios, it is not necessary to run the full process. For instance, the unique EAN number is often present as part of the product description and can be used as a reliable matching criterion. Over time, a list of synonyms (e.g. for describing a processor) is created by the system, based on previous runs, and is used in future normalization and similarity checking steps.

4. LARGE SCALE EXTRACTION

4.1 Architecture

In OMI scenarios, a large amount of data is extracted regularly to obtain an in-depth daily picture of the market. Therefore, it is crucial to be on the one hand very performant, and on the other hand to provide means for extreme scalability, especially in cases with a high peak load at certain times.

The Lixto Retail Solution offers end users the ability to schedule extraction jobs on demand. Lixto TS creates an extraction plan based on the regularly scheduled jobs and the on-demand jobs. Data extractions are executed via the *HydraNT Extraction Cluster*. TS uses HydraNT as directory service, asking for a free VD runtime head to be used in the next execution. HydraNT queues the request and assigns the best suited head, based on the weights described below. As depicted in Figure 8 machines can be registered on HydraNT and inform HydraNT about the number of running VD heads and the machine parameters. HydraNT distributes the load and, as described in the next section, can add machines if the load gets too high and the extraction plan can not be finished in the required time.

4.2 Cloud Usage

In *Cloud Computing*, dynamically scalable and often virtualized resources are accessed as service. Cloud Services provide an agile infrastructure and scalability on demand. Cloud Services can be used to transfer heavy computations to an external service that is usually paid by usage time or

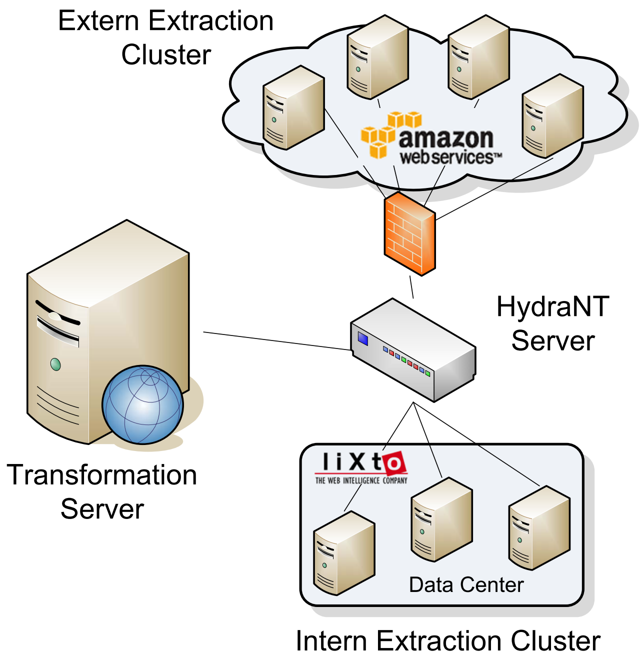

Extern Extraction Cluster
amazon webservices™
HydraNT Server
Transformation Server
lixto THE WEB INTELLIGENCE COMPANY
Data Center
Intern Extraction Cluster

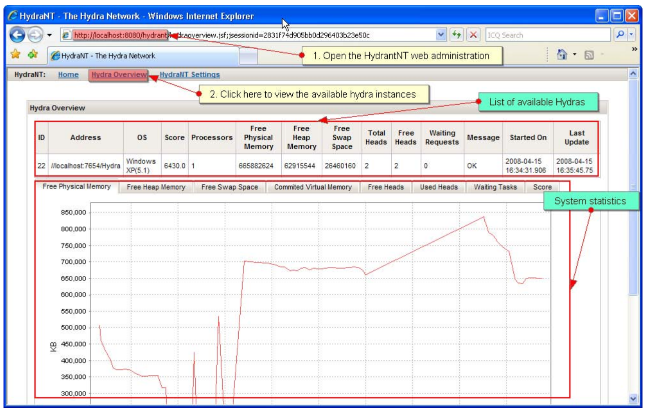
Figure 8: Extraction Cluster Architecture

processed data. Cloud Services are advantageous if at certain times high computation load is expected, or the load can not be estimated beforehand and varies a lot, or a computation is only needed at certain times. Refer for instance to [18] for using cloud services in SaaS applications.

In the Lixto OMI scenario, computations are Web extractions. Cloud clients receive a wrapper and extraction parameters, and pass back reports and data. No data is persisted on cloud instances. Even if cloud reliability might be lower than a traditional server center, the provision of a retry mechanism ensures that all data is processed successfully.

One of the most advanced cloud services is provided by *Amazon Web Services*[1] (AWS). Instances are controlled via Web Services. User-defined images can be created, based on various operating systems. Required software is either pre-installed or loaded on the fly from the Amazon storage

[1]http://aws.amazon.com


HydraNT - The Hydra Network - Windows Internet Explorer
http://localhost:8080/hydrant/hydraoverview.jsf;jsessionid=2831f74d905bb0d296403b23e50c
ICQ Search
HydraNT - The Hydra Network
1. Open the HydrantNT web administration
HydraNT: Home Hydra Overview HydraNT Settings
2. Click here to view the available hydra instances
List of available Hydras
Hydra Overview
Free Physical Memory | Free Heap Memory | Free Swap Space | Commited Virtual Memory | Free Heads | Used Heads | Waiting Tasks | Score
System statistics
850,000 800,000 750,000 700,000 650,000 600,000 550,000 500,000 450,000 400,000 350,000 300,000
KB

ID	Address	OS	Score	Processors	Free Physical Memory	Free Heap Memory	Free Swap Space	Total Heads	Free Heads	Waiting Requests	Message	Started On	Last Update
22	//localhost:7654/Hydra	Windows XP(5.1)	6430.0	1	665882624	62915544	26460160	2	2	0	OK	2008-04-15 16:34:31.906	2008-04-15 16:35:45.75

Figure 9: Hydrant Usage Overview

service S3 when an instance of this image is started. On starting an instance, user defined parameters are given. In our case, these parameters include the number of VD heads, which VD version to use, and on which HydraNT the cloud machine registers.

4.3 Extraction Cluster

Figure 9 illustrates the management GUI of HydraNT. It shows all registered instances, both in the server center and in the AWS Cloud. Each registered machine regularly broadcasts its status, including machine properties and current execution statistics. These settings, such as the number of used VD heads, free RAM and CPU load are shown and used for deciding which machine to use for the next incoming request.

Application managers can decide on the weight of all parameters, e.g. assigning the free heads parameter a higher weight. The total score of each machine is given in the overview; the machine with the highest score is picked next. Requests first go to a global queue if they can not be assigned to a machine immediately.

As well as configurable weights, application managers create rules when to start new machines or shut down existing ones. Typical rules comprise:

- Start an instance if no instance is currently running.
- Start an instance if the average request load over X minutes is higher than Y.
- Shut down an instance in case there are no waiting tasks for Y minutes.

Shutdown can be initiated immediately or after a full consumption hour. In the latter case, the chance of the instance to get selected decreases as the shut down time approaches. In case a new machine will be restarted in the meanwhile, one of these instances is revived first.

HydraNT's Inspection Panel provides overview statistics of number of document loads, wrapper execution requests, retries and similar, for each client that accesses the extraction cluster.

5. CASE STUDY

In the following case study, we will give an example of an existing business application in the domain of computers

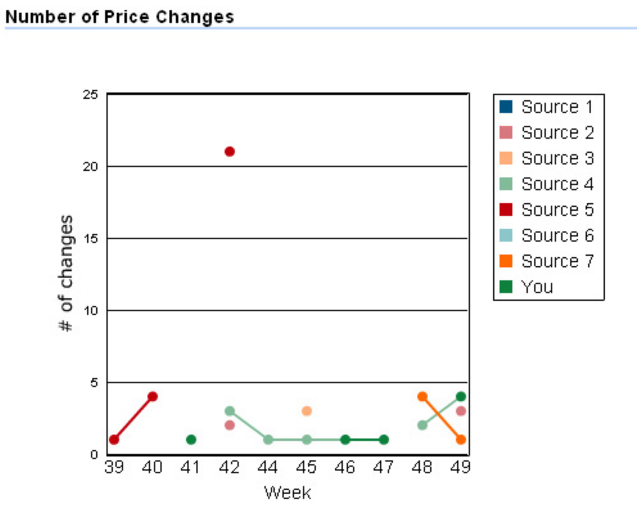


Figure 11: Number of price changes over time

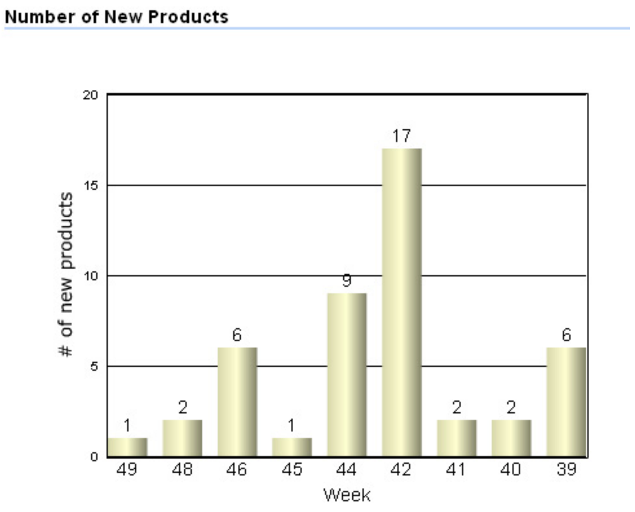
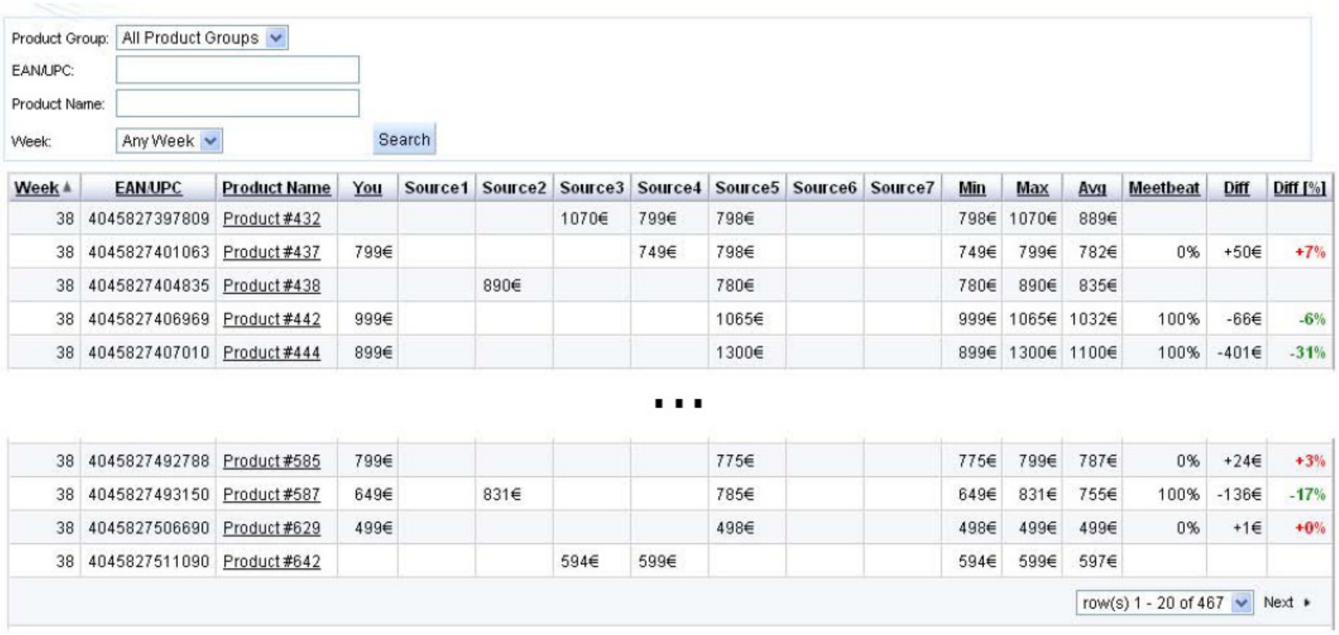


Figure 12: Number of new products introduced

and electronics consumer goods. This example application covers the main aspects of an online market intelligence solution, starting from web data extraction, data cleansing and rectification, and moving to data analysis and data visualization.

5.1 The Requirements in the Computers and Electronics Market

In the computers and electronics market we can observe a large number of online sales channels such as online stores and online distributors. For computers and electronic consumer goods it is important, when comparing products of different suppliers, to compare apples to apples, i.e., to truly understand the configuration of individual products and to provide a feature based comparison matrix. It is therefore a strong requirement that the web data extraction process is able to identify and extract objects on web pages including detailed features of those objects. In the context of com-



Product Group:	All Product Groups
EAN/UPC:	
Product Name:	
Week:	Any Week

Week ▲	EAN/UPC	Product Name	You	Source1	Source2	Source3	Source4	Source5	Source6	Source7	Min	Max	Avg	Meetbeat	Diff	Diff [%]
38	4045827397809	Product #432				1070€	799€	798€			798€	1070€	889€			
38	4045827401063	Product #437	799€				749€	798€			749€	799€	782€	0%	+50€	+7%
38	4045827404835	Product #438			890€			780€			780€	890€	835€			
38	4045827406969	Product #442	999€					1065€			999€	1065€	1032€	100%	-66€	-6%
38	4045827407010	Product #444	899€					1300€			899€	1300€	1100€	100%	-401€	-31%

...

38	4045827492788	Product #585	799€					775€			775€	799€	787€	0%	+24€	+3%
38	4045827493150	Product #587	649€		831€			785€			649€	831€	755€	100%	-136€	-17%
38	4045827506690	Product #629	499€					498€			498€	499€	499€	0%	+1€	+0%
38	4045827511090	Product #642				594€	599€				594€	599€	597€			

Figure 10: Comparison of prices

puters and electronic consumer goods, such an object could be e.g. a notebook computer, including all detailed features of these product such as available memory, hard disk, CPU, graphics, and screen size.

Once the data is collected and stored in a structured format, it is necessary to match the values of all product features against a standardized product feature database. Without this data cleansing, step values of product features would not be comparable, e.g., the hard disk size could be stated as ”HD 160GB” or ”hard disk size 160gb”. This needs to be harmonized to be comparable and requires to give a canonical description such as name: size, value: 160, unit:GB (as described in Section 3.2.2).

End users such as pricing managers will typically use an analytics framework to dig into the aggregated data. What users require to see is how their own offering performs in comparison to the offering from competitors. Comparison can take several dimensions into account: first, the pricing, i.e., what are the prices and how do these prices develop over time; second, the inventory, i.e., which products are offered and and how is the change rate of offered products; third, promotions and other sales campaigns, i.e., which products are specifically featured or advertised for sale.

5.2 Sample Reports and Dashboards

To investigate the market situation, we provide a number of pre-configured reports and dashboards that provide a standardized view on the aggregated market data. Figure 10 shows such a sample report that is very popular with pricing managers. On this grid view, the user can see the current price according to the own pricing versus the competitive pricing for each product that is offered on the market. Moreover, certain statistics and key performance indicators help the user to immediately relate their own pricing to the current market pricing. A filter allows to select only certain products that should be part of the comparison matrix. This helps category managers to examine exactly the products for which they are responsible.

Statistics and key performance indicators that are calculated are

- *Min Price*: This measure gives the minimal price observed among all offerings of the same product.
- *Max Price*: This measure gives the maximal price observed among all offerings of the same product.
- *Average Price*: This measure gives the average of all prices observed on the market.
- *Price Difference*: This measure shows the difference of the own price versus the cheapest available offer on the market, both as an absolute currency value and as a relative percentage value.
- *Meet Beat Ratio*: This measure defines the number of competitive offers that are more expensive than the own offering, e.g., if two out of three competitive prices would be more expensive than the own price for the same product, we observe a meet beat ratio of 67%.

This grid view can also be used to spot important market events that call for action, e.g. setting a new price that better reflects the current market situation. In such a case, we restrict the view to show only products that are clearly out of the observed market price band. Products can be either much too cheap – in which case we lose margin, because we could price higher without losing market share – or much to expensive. In this case we will suffer from reduced demand because the same product can be bought at a more attractive price from one of our competitors.

Whereas reports on the market situation show only a snapshot at a certain point in time, market trend reports highlight the market development over time. Typical diagrams are shown in Figure 11 and Figure 12. Figure 11 shows the volatility of the market in terms of price changes on each observed sales channel over time. This graph helps to identify those competitors that are very agile and often reconsider their pricing. Figure 12 shows the number of newly introduced products per week. This shows the rate of change in inventory of competitors. It helps the market manager to better understand the product dynamics that are given in certain product categories for a specific sales channel.

All of these reports and data visualizations are based on a data warehouse that enables drilling into the market data and slicing and dicing the data according to the user's specific needs. This data warehouse gives the flexibility to provide individual reports according to the user needs.

6. CONCLUSION

In this paper we have given an overview of the new Lixto OMI software and have discussed some aspects in depth. This software is a unique combination of leading data extraction technology with features specifically designed and developed for online market intelligence applications. By using Lixto OMI, such applications can be implemented quickly and efficiently. Moreover, the special features enabling the usage of extraction clusters and cloud computing make Lixto OMI highly scalable, which is an essential requirement in cases where massive amounts of Web data need to be extracted.

This scalability has already proven beneficial not only to Lixto's customers but also to the Lixto company itself. In fact, Lixto provides OMI *services* to various clients, especially in the areas of retail and tourism. In order to obtain consistent pricing data in these areas, it is essential to gather large amounts of data in a very short time, a requirement that can only be fulfilled through the parallelization of extraction tasks. The local solution would be to maintain an server farm with hundreds of servers. This would require, in addition, an Internet connection with an unusually high bandwidth. This is of course a very costly solution. The servers would be idle between successive runs but would require maintenance. Cloud computing is therefore the ideal solution, especially for peaks, and has helped us and our clients save costs.

To the best of our knowledge, Lixto is the first company to offer a comprehensive solution to online market intelligence requirements. We think that *Web ETL for OMI* will soon become a very hot topic for the Fortune 1000 companies, as well as many smaller enterprises. Lixto is therefore not only looking for further customers, but also for partners who recognize and appreciate the tremendous potential of the technology reported in the present paper.

7. REFERENCES

[1] V. Anupam, J. Freire, B. Kumar, and D. Lieuwen. Automating Web navigation with the WebVCR. *Computer Networks*, 33(1–6):503–517, 2000.

[2] R. Baumgartner, M. Ceresna, and G. Ledermüller. Deep web navigation in web data extraction. In *Proc. of IAWTIC*, 2005.

[3] R. Baumgartner, S. Flesca, and G. Gottlob. Declarative Information Extraction, Web Crawling and Recursive Wrapping with Lixto. In *Proc. of LPNMR*, 2001.

[4] R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. In *Proc. of VLDB*, 2001.

[5] R. Baumgartner, W. Gatterbauer, and G. Gottlob. Web Data Extraction System. In *Encyclopedia of Database Systems (to appear)*. Springer-Verlag New York, Inc., 2009.

[6] R. Baxter, P. Christen, and T. Churches. A comparison of fast blocking methods for record linkage. In *Proceedings of the ACM Workshop on Data Cleaning, Record Linkage and Object Identification*, 2003.

[7] J. Carme, M. Ceresna, and M. Goebel. Web wrapper specification using compound filter learning. In *Proceedings of the IADIS International Conference WWW/Internet 2006*, 2006.

[8] H. Cunningham, K. Bontcheva, and Y. Li. Knowledge Management and Human Language: Crossing the Chasm. *J. of Knowledge Management*, 9(5), 2005.

[9] S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli. Web wrapper induction: a brief survey. *AI Communications Vol.17/2*, 2004.

[10] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *Proc. of WWW*, May 8–12, 2007.

[11] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for Web Information Extraction. In *Proc. of PODS*, 2002.

[12] L. Gu, R. Baxter, D. Vickers, and C. Rainsford. Record linkage: Current practice and future directions. Technical report, CSIRO Mathematical and Information Sciences, 2003.

[13] T. Hassan and R. Baumgartner. Table recognition and understanding from pdf files. In *Proc. of ICDAR*, 2007.

[14] M. Herzog and G. Gottlob. InfoPipes: A flexible framework for M-Commerce applications. In *Proc. of TES workshop at VLDB*, 2001.

[15] S. Kuhlins and R. Tredwell. Toolkits for generating wrappers. In *Net.ObjectDays*, 2002.

[16] A. H. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. In *Sigmod Record 31/2*, 2002.

[17] B. Liu. Web Content Mining. In *Proc. of WWW, Tutorial*, 2005.

[18] R. Mietzner and F. Leymann. Towards provisioning the cloud: On the usage of multi-granularity flows and services to realize a unified provisioning infrastructure for saas applications. In *Proceedings of the International Congress on Services*, 2008.

[19] B. Ribeiro-Neto, A. H. F. Laender, and A. S. da Silva. Extracting semi-structured data through examples. In *Proc. of CIKM*, 1999.

[20] R. Tredwell and S. Kuhlins. Wrapper Generating Tools, 2003. http://www.wifo.uni-mannheim.de/ kuhlins/wrappertools/.

[21] Wikipedia. Entry: *Market Intelligence*, 2009. As of April 15, 2009.