Annotating the Legacy Web with Lixto

Robert Baumgartner[1,3], Georg Gottlob[1,3], Marcus Herzog[1,3], and Wolfgang Slany[2,3]

[1]Institut für Informationssysteme, Technische Universität Wien,
Favoritenstr. 9, 1040 Vienna, Austria
{baumgart,gottlob,herzog}@dbai.tuwien.ac.at

[2]Institut für Softwaretechnologie, Technische Universität Graz,
Inffeldgasse 16b, 8010 Graz, Austria
slany@ist.tu-graz.ac.at

[3]Lixto Software GmbH, Donau-City-Str. 1/Gate 1, 1220 Vienna, Austria
{baumgart,gottlob,herzog,slany}@lixto.com

Introduction

The Semantic Web is still a vision. The unstructured Web of today contains millions of documents which cannot be queried and where layout and structure are heavily mixed. Moreover, they are not annotated at all. There is a huge gap between Web information and the qualified, structured data as required in corporate information systems. According to the vision of the Semantic Web, all information available on the Web will be suitably structured, annotated, and qualified in the future. However, until this goal is reached, and also, towards a faster achievement of this goal, relevant data can be (semi-)automatically extracted from HTML documents and automatically translated into a structured format, e.g., XML.

A program that automatically extracts data and transforms it into another format (markups the content with semantic information) is called a *wrapper*. Intelligent content extraction provides the foundation for automatic generation of semantic markup. Various approaches to automatic content extraction have been proposed, ranging from machine learning techniques to pattern recognition techniques. However, these approaches in general fail to produce useful results due to the complexity of Web pages. Other approaches suggest the manual editing of script files that wrap the relevant data from Web pages into more structured formats. Such processes are time-consuming, hard to understand for non-technical wrapper designers, and script files are not easy to maintain.

We propose another approach - a supervised and visual definition of content extraction. Based on interactively identifying and extracting relevant parts of HTML documents and translating content to XML format, we designed and implemented the efficient wrapper generation tool *Lixto Visual Wrapper* [2] which is well-suited for building HTML to XML wrappers. Such a wrapper can be applied to continually extract relevant information from this class of Web pages.

The process of wrapping consists in two steps: First, the *identification phase*, where relevant fragments of Web pages are extracted. Such extraction rules are semi-automatically specified by a wrapper designer (e.g., in a purely visual way in Lixto Visual Wrapper). This step is followed by the *structuring phase* where the extracted data is mapped to some destination format, e.g., enriching it with XML tags. With respect to the Semantic Web, a third phase is required: Each information unit needs to be put into relation with other pieces of information. Additionally, methods for automatically relating extracted information to existing domain ontologies are required.

The aspect of integration of extracted data plays an important role in querying the Web; data schemes, metadata, and background knowledge such as ontologies must be integrated to reach a uniform semantic interpretation of information. *Lixto Transformation Server* [6] covers the integration, annotation and delivery process as part of the Lixto set of tools. As long as the Web is not a source of machine-readable data but still merely accessible for human beings, wrapper technology provides the possibility for computers to query and interact with it [3].

Sample Technical Content that will be Demonstrated

We consider a simple yet illustrative real-world example. We show how relevant information is extracted and semantically annotated. The example deals with currency exchange rates of banks: A company is interested in which one of three banks at each moment offers the best EUR/USD exchange rate. Each bank publishes this information on the Web. The up-to-date information should be available in an annotated way, easily accessible to Semantic Web agents.

In the following, we describe a step-by-step construction of this example within Lixto from the viewpoint of an application designer who creates this application. First, in the *Lixto Transformation Server*, a designer creates a new empty information service, a so called “information pipe”. Assisted by the visual composer of this pipe, the operator of such a service inserts processing components, configures them, and interconnects them into a unidirectional graph, reflecting the flow of processed information.

Naturally, each bank's Web site is a source of information. However, these pages have to be wrapped in order to extract the relevant information. In our case, the operator is merely interested in the actual EUR/USD exchange rate and the name of the bank. Thus, a new *Source* component (the circular symbols in Figure 2, one for each bank) is created, starting with one for the Austrian National Bank OENB. Using this component, the operator can browse the Web to navigate to the appropriate page that contains the rates, including the filling of forms, the passing of authorizations, and cookies. The source component records all these actions. Once the service is configured, it will automatically replay the navigation. After selecting the page where the actual rates are given, the designer needs to specify how to extract the EUR/USD rates and the bank name. We will demonstrate how the Lixto tools allow the handling of this and similar processes.

Architecture

A human being tends to assign semantic meaning to parts of a Web page; a designer does not think of *table row* as of a set with text values, but rather as of an exchange rate between Euro and foreign currencies. Therefore, the basic building block of a wrapper program is a so-called *pattern*, a container for pieces of information with the same meaning. Examples of patterns are *bank_name*, *currency,* etc. Patterns are structured in a hierarchical fashion. In the lower half of the Visual Wrapper's UI an active example Web page is displayed for marking example instances: for each type of Web page, a separate wrapper has to be created. In the following the wrapper creation for OENB is illustrated.

In this case, the designer identifies one of the table rows as a pattern *eur_to_usd*. In the Visual Wrapper, a pattern with this name is created. Once a pattern is created, the operator continues with visually defining a filter, a crucial part of the pattern which specifies how to extract relevant information from its parent pattern instances. Internally, filters are represented in a declarative logic-based language (elog, see [2]), but the language is entirely hidden from the wrapper designer.

Defining a filter requires the operator to select an example instance with two mouse clicks on the example Web page (Fig. 1 on the left). Filter definition continues then with optional fine-tuning of properties for the generated generalization of the chosen example. It is possible to visually debug the wrapper program, i.e., to interactively test filters and patterns. Typically, operators test patterns and filters after adding new components. Based on the results, operators decide whether to extend (i.e., add a filter) or shrink (i.e., add condition to an existing filter) the set of matched instances.

The system displays a list of matched instances for the so-far created filter, e.g., by highlighting parts of the Web page. In this case all table rows are detected by the proposed generalization of the filter. Because the operator is interested in one particular table row only, an additional internal condition is added. While creating such a condition, the designer is asked to choose one example instance (i.e., table row) and then to specify with two mouse clicks a required element - in this case, a first cell containing the substring 'USD'. This is specified in the attribute fine-tuning of this condition (Fig. 1 on the right). Next a child pattern *exchange_rate* of the just defined pattern is created and then a filter extracting the third cell. The filter is again created using the two click selection inside an example parent instance and then specifying a concrete value for the Lixto system ‘columns’ attribute. Similarly, patterns *bank_name* and *date* are added. After uploading the wrapper to the Transformation Server the Source component returns the following data for the particular source component the wrapper is associated with:

```
<bank>
  <exchange-rate>0.9843</exchange-rate>
  <bank-name>OENB</bank-name>
  <date>August 12th 2004</date>
<bank>
```

In a next step the designer configures a *Scheduler* for the Source component to specify when the Web pages are polled for new information. In a similar manner the source components for the other two banks are created, including the navigation sequence, the wrapper, and the scheduler.

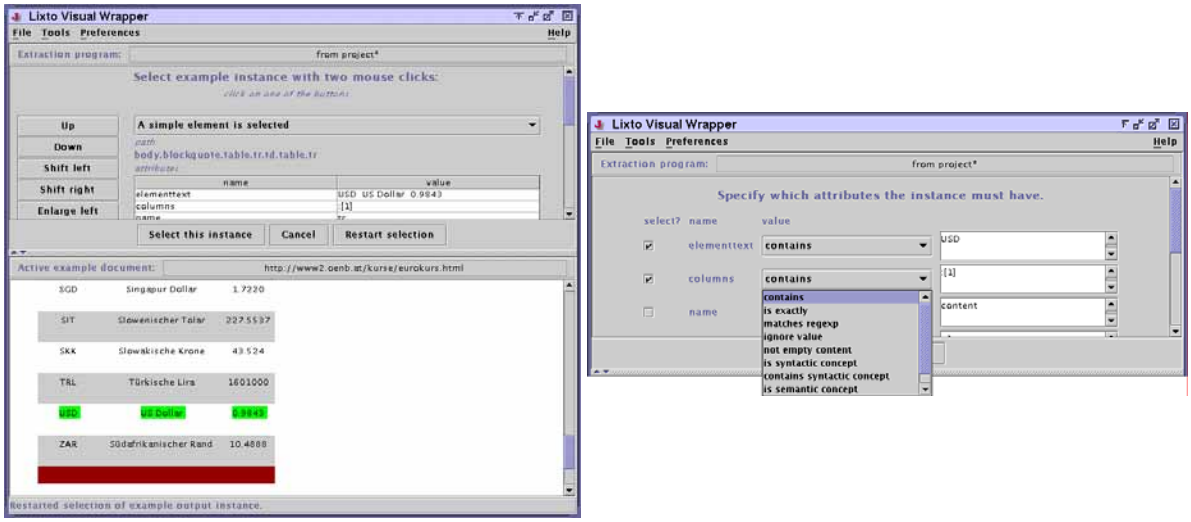

Figure 1: Instance and Attribute Selection

As a second step, the designer needs to integrate the three sources of information into a homogeneous structure. For this, an *Integrator* component is added to which the sources are connected. In Fig. 2, the three disks represent the three bank Sources and the arcs the XML information flow. The integrator component is depicted as a triangle. In the Integrator UI, the operator chooses (or constructs from scratch) the desired output XML structure and maps the elements of each input XML schema to an output schema. The designer also has the possibility of content format unifications. In our simple example this is not necessary, as, e.g., the dates already have the same structure and content format. In general, also different kind of data can be joined, e.g., integrating bank exchange rates with background data on the bank, e.g., news articles [1].

Delivering Semantics

At this point, the system already retrieved all the desired information. The final step is to semantically markup the data and to define the relations between the elements. This is covered in the *Annotation Deliverer* of the Transformation Server. The purpose is to transform the extracted data to a semantic representation. In this Deliverer component of the Transformation Server, a meta-data description of the input data can be generated.

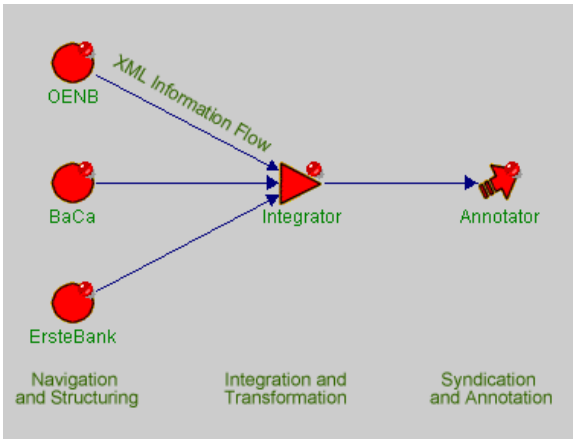

Figure 2: Pipe Architecture

To capture meanings, objects and relations between objects need to be described. The Resource Description Framework (RDF) and languages such as DAML/OIL [5] can be used for this purpose. In the *Annotation Deliverer* of TS, based on the XML output, class and subclass relations (e.g., in this example this might be “Bank” and one of its subclasses is “AustrianBank”) can be defined (using the actual XML data as instances of the respective classes). For classes, additional features such as *complementOf* or *disjointWith* can be defined. Each object can be described by properties. A bank can for instance be described by the property “hasHomeCurrency”. By these means the designer can state that each bank has exactly one home currency. Moreover, a relation “hasCurrencies” can be

defined to state which currencies are available in that bank, and give the exchange rate to the home currency. New relations could also be defined directly by the incoming XML data.

Properties can be the inverse of other properties, be transitive, symmetric, require specific cardinalities, etc. Several domain-specific axioms can be used for formal assignments and to apply reasoning. The application designer is visually supported to create ontologic descriptions such as the following:

```
<daml:Class rdf:ID="AustrianBank"/>
  <rdfs:subClassOf rdf:resource="#Bank"/>
</daml:Class>
....
<daml:ObjectProperty rdf:ID="hasCurrencies">
  <rdfs:domain rdf:resource="#Bank"/>
  <rdfs:range rdf:resource="#Currencies"/>
</daml>
```

This particular application does not fill all parts of the defined ontology with instances, but later other applications can provide additional data.

A snapshot of the completed information pipe is shown in Figure 2, with the annotation deliverer on the right. Finally, the created information pipe has to be activated. It will run independently of the operator, pushing the extracted information regularly into a semantic description. The annotated data can be added to some knowledge base and conveniently used by Semantic Web agents. Due to the robustness of Lixto wrappers, the application needs little maintenance, except if the Web pages completely change their structure. Semantic Web Agents can easily pose individual queries such as retrieving the best possible exchange rate of today, or even conclude new facts based on such descriptions.

Availability of the software

Lixto Visual Wrapper und Lixto Transformation Server are based on a solid theoretical framework [2,4,6]. Lixto applications collect data, transform the information into a common structure, and syndicate it to applications or devices in an annotated way. The development of the Lixto tools was initiated at the Institute of Information Systems at Vienna University of Technology and has since then been commercialized by the spin-off company Lixto Software GmbH. Research on extending the semantic capabilities of the Lixto set of tools is currently jointly undertaken with the research groups at Vienna University of Technology and Graz University of Technology.

Acknowledgment

This work has been partially funded by the REWERSE EU Network of Excellence.

Bibliography

[1] R. Baumgartner, S. Eichholz, S. Flesca, G. Gottlob, M. Herzog. Semantic Markup of News Items with Lixto. In: *Annotation for the Semantic Web*, S. Handschuh and S. Staab (eds.), IOS Press, 2003.

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

[3] R. Baumgartner, G. Gottlob, M. Herzog, and W. Slany. Interactively Adding Web Service Interfaces to Existing Web Applications. *SAINT* 2004.

[4] G. Gottlob and C. Koch. Monadic Datalog and the Expressive Power of Languages for Web Information Extraction. *PODS*, 2002.

[5] D. L. Mc Guiness, R. Fikes, J. Hendler, L. A. Stein. DAML+OIL: An Ontology Language for the Semantic Web. *IEEE Intelligent Systems, 17(5)*, 2002.

[6] M. Herzog and G. Gottlob. InfoPipes: A Flexible Framework for M-Commerce Applications*. TES Workshop at VLDB*, 2001