Declarative Web data extraction and annotation

Carlo Bernardoni[1], Giacomo Fiumara[2], Massimo Marchi[1], and Alessandro Provetti[2]

[1] Dip. di Scienze dell'Informazione, Università degli Studi di Milano
Via Comelico 39, I-20135 Milano, Italy
{*carlo.bernardoni,marchi*}*@dsi.unimi.it*
[2] Dip. di Fisica, Università degli Studi di Messina.
Sal. Sperone 31. S. Agata di Messina, I-98166 Italy
{*fiumara,ale*}*@unime.it*

Abstract. We propose a software architecture for semantics-based annotation of data extracted from Web sources. Starting from the *LiXto* suite, which enables semi-automated extraction of XML data from regular documents, we present a solution for attaching background information to individual tags by means of so-called decorations. Decoration is carried out as an inferential activity in the formal context of Answer Set Programming. We discuss a motivating example that will serve as a validation to our approach.

1 Introduction

This article illustrates the architecture for Web data gathering and annotation that we have developed by combining novel and existing modules. The key functionalities of our application, i.e., data extraction from HTML pages and annotation of XML tags with new data, are defined in terms of logic programs. Summarizing, our architecture works as follows. Web sources, i.e., Web sites posting *dynamic* data (news, webcasts, blogs etc.) are routinely consulted and relevant information is selected, downloaded and saved into XML *tags*. The resulting tags are then translated into sets of Datalog-syntax *facts;* such facts are then added to logic programming rules, and deduction can start. Applying the rules can give these effects: i) some element of the tag is dropped, e.g., because it is deemed incorrect, uninteresting or superseded by other tags or ii) new tags are added. The idea is that the new tag brings out some *semantical* consideration that would not be found by simply accessing the text of the Web source. As a result, the Web data will be transformed into a marked-up XML version that mirrors the available data *as well as* the particular interpretation that has been applied. It should be noticed that there is no assumption on the shape of the original Web data, i.e., both the XML encapsulation and the subsequent transformation can be applied to arbitrary XML sources. Finally, the resulting XML tags are made available to Web services (WS) through standard channeling methods.

1.1 The rôle of Logic Programming

It is important to notice that in the project presented here each relevant component of the proposed architecture, including the *LiXto* suite, is related to Logic Programming. Indeed, the project presented here is part of our long-term research program (see, e.g., [10,3]) of studying *declarative policies* in the context of [Semantic] Web services.

In fact, even the wrapping of Web sources is done using the tools developed by the *LiXto* project [9] which in turn is based on *Elog*, an extension of DATALOG. The same can be said for the Transformation Server module. A formal account of the logical interpretation of XML transformations is given by Gottlob and Koch in [6]. However, it must be stressed that for this project a novel bijective mapping from XML tags to Datalog-syntax facts has been defined, whose details are in [4].

The automated reasoning and tag manipulation task is carried out in Answer Set Programming (ASP), which can be seen as an extension to DATALOG that deals with default reasoning. For lack of space, we refer the reader to the original work of Gelfond and Lifschitz [5] or to the excellent survey in [11] for a detailed introduction to ASP.

Although our work does not address the Semantic Web (SW) as it is commonly understood, i.e., in terms of managing/publishing data annotated with a SW language such as RDF or OWL, our understanding, supported by some preliminary experiments, is that the inferential part can be used to apply *annotation policies* that transformed the Web data into RDF/OWL tags.

This article is structured as follows. Section 2 gives an overview of the *LiXto* project and explains the main features of the *LiXto* suite that were used in our project. Our architecture is introduced in Section 3, where the adopted *xml2asp* transformation will be described. Section 4 describes the application example we have worked on to test and validate our blueprint. Finally, Section 5 summarizes the work done so far and discusses the lines along which this project is now being developed.

2 The *LiXto* architecture

The *LiXto Suite*[9] is a data extraction and transformation software kit for retrieving and converting information from regular documents, usually found on the World Wide Web. It is mainly composed of two applications:

1. the Visual Wrapper (VW), and
2. Transformation Server (TS).

that we are going to describe in more detail next. It should be remarked that the Transformation Server is indeed an application that treats arbitrary XML data and thus it is interesting in its own, i.e., for managing sources of native XML data.

2.1 The *LiXto* Visual Wrapper

The *Visual Wrapper* (VW) is a visual, interactive tool for generating *wrappers*. A wrapper is understood as a program that allows for automatic and flexible extraction of information from regular documents such as Web pages. Wrappers are designed to continually extract relevant information from dynamic Web pages and *organize* it into XML trees.

According to Gottlob et al. [8]:

> The VW allows a user to create wrappers by visually selecting relevant patterns directly on browser-displayed pages. It allows for extraction of target patterns based on surrounding landmarks, on the contents itself, on HTML attributes, on the order of appearance and on ontological or syntactic concepts. Extraction is not limited to tokens of some document object model, but also possible from flat strings. The VW also allows for more advanced features such as disjunctive pattern definitions, following links to other pages during extraction and recursive wrapping.

Therefore, *LiXto* can implement data manipulation tasks that are beyond pattern recognition, i.e., are *data-driven* and can adapt to the input. For further details on how the information extraction works and on its computational complexity, please refer to the presentations in [7] and [8].

2.2 The *LiXto* Transformation Server

The *Transformation Server* (TS) is a software that supports the design and execution of applications –called *pipes*– for processing XML data flows. The TS extracts data from Web sources and organizes them into XML trees, by mean of wrappers, designed with the Visual Wrapper described above.

Subsequently, *LiXto* TS allows the application designer to format, transform, merge and deliver XML data to various devices (e.g. HTML pages, XML pages, email, SMS). XML data manipulation is done by specialized and interacting modules (*components*) that the TS user creates, configures and connects (following a *pipeline* paradigm) in a completely visual environment. There exist several specialized components e.g. those for wrapping, standardization, integration or delivery purposes. In particular, it is worth mentioning the so-called *Shell* component which allows for executing external programs on XML data. We have exploited the shell component as a gateway to the inferential engine described next.

3 The proposed architecture

In this section we describe the core activity of our architecture: the inference-based manipulation of XML tags. The internal schema of our architecture is shown in Figure 3.

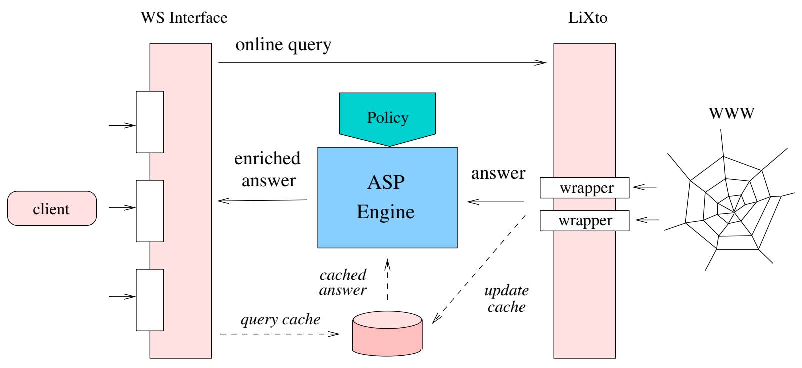

Fig. 1. The internal schema.

First, a simple Web Service collects data requests from its clients. Second, the needed data are routinely (e.g., once a day) extracted from the relative Web sources, (by *LiXto* wrapper), and encoded as an XML tag (by *LiXto* TS). Next, the *decoration* phase starts.

Decoration takes place in several steps. First, a Perl program called *xml2asp* [3] translates XML data into Datalog facts. Second, both the obtained facts and the so-called *policy,* i.e., the rules that guide the process, are fed to the ASP inferential engine. In our case, the ASP engine consists of the *lparse* grounder and *smodels* solver [12]. The *smodels* output, i.e., the answer set, is then filtered to retrieve the relevant facts describing the decorated XML tag. Another Perl program, called *asp2xml* will then re-create and actual tag, which finally will be served to the clients by some more-or-less standard Web service.

4 An Application example

We are implementing a service that allows a user to monitor hotels availability in Milan on certain dates. International travelers are accustomed to the *stars* rating system, where stars are proportional to the level of services and comforts available at the hotel. When the stars rating is absent, our service tentatively classifies the hotel facilities on the basis of the price range w.r.t. the Milan market or even w.r.t. the particular location.

4.1 The kelkoo.it on-line booking service

The `kelkoo.it` Website allows users to search for available hotels, in a given resort and for a given period, by querying many (in this case, more than a dozen) hotel websites. Searching is based on the following parameters:

[3] Both *xml2asp* and *asp2xml* are available from *http://mag.dsi.unimi.it/~carlo/#projects*

– resort;
– arrival date;
– departure date;
– room type;
– number of adults, and
– number of children.

The result page displays available hotels as an HTML table, one hotel per row.

The wrapper for kelkoo.it Using the *LiXto* Visual Wrapper, we implemented a wrapper to extract, for each hotel, the following information:

– name;
– address;
– brief facility description;
– room type, and
– price.

The following is a fragment of an XML output of the wrapper:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <rootPattern>
    <Hotel>
      <Name>HOTEL ANTARES ACCADEMIA</Name>
      <Address>Periferia VIALE CERTOSA 68 - 20155 MILAN</Address>
      <Room>
        <Description>Doppia</Description>
        <Price>92</Price>
      </Room>
      <HotelDescription>
        L’hotel e’ situato a pochi passi dalla fiera e dal Castello
        Sforzesco, e’ ben collegato al centro della citta’ dalla
        metropolitana.
      </HotelDescription>
    </Hotel>
    ...
  </rootPattern>
</document>
```

4.2 Annotating kelkoo.it

The *LiXto* transformation server provides a visual tool for creating pipelines of activities. The pipe meant to “enrich” the information fetched by the wrapper (Section 4.1) is depicted in Figure 4.2 below.

The *Source* component runs the wrapper described in Section 4.1 on a kelkoo.it result page, triggered by a proper querying URL whose parameters’ values (see Section 4.1) are user-definable.

Next, the *Shell* component is configured to invoke, through the Operating System, the *xml2asp* translator, the inferential engine and then the *asp2xml* translator.

The ASP program used to annotate these data is in charge of adding the `<Stars>` tags which we infer from the hotel room fares. This is an example rule:

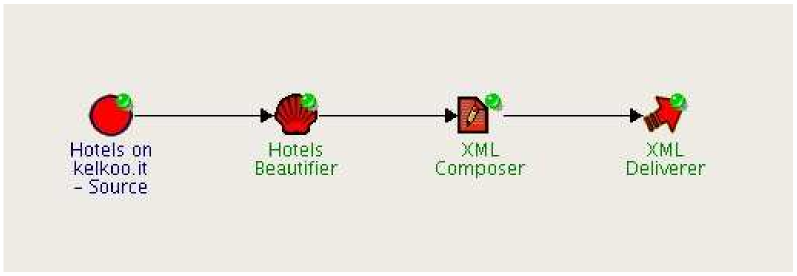

Fig. 2. A smart pipe for on-line hotels booking on kelkoo.it.

```
newNode(Hotel,"Stars",5):-
    tag(Hotel,"Hotel"),
    node(Hotel),
    tag(Room,"Room"),
    node(Room),
    parent_of(Hotel,Room),
    tag(Price,"Price"),
    node(Price),
    parent_of(Room,Price),
    isNumber(Price,Qty),
    Qty > 250.
```

The auxiliary predicates needed to reason about the tag structure are defined as follows:

```
parent_of(X,Y) :- node(X),
                node(Y),
                firstchild(X,Y).
parent_of(X,Y) :- node(X),
                node(Z),
                firstchild(X,Z),
                node(Y),
                has_brother(Z,Y).

has_brother(X,Y) :- node(X),
                    node(Y),
                    nextsibling(X,Y).
has_brother(X,Y) :- node(X),
                    node(Y),
                    node(Z),
                    nextsibling(X,Z),
                    has_brother(Z,Y).
```

Applying the rules above leads to the derivation of some *newnode* atoms which will be included in the answer set returned by *smodels*. The shell component will take the answer set and pass it, modulo dropping some irrelevant atom, to the *asp2xml* back-translator. As a result, a the output tag will show an additional `<Stars>` tag, whose value, between 1 and 5, expresses the inferred class for a hotel, as in the following fragment:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <rootPattern>
    <Hotel>
      <Name>HOTEL ANTARES ACCADEMIA</Name>
      <Address>Periferia VIALE CERTOSA 68 - 20155 MILAN</Address>
      <Room>
        <Description>Doppia</Description>
        <Price>92</Price>
      </Room>
      <HotelDescription>
        L'hotel e' situato a pochi passi dalla fiera e dal Castello
        Sforzesco, e' ben collegato al centro della citta' dalla
        metropolitana.
      </HotelDescription>
      <Stars>3</Stars>
    </Hotel>

    ...

  </rootPattern>
</document>
```

After the decoration phase, it is possible to use *LiXto* TS again to program, by the *Composer* component, some re-arrangement of the XML tag. In any case Composer has to handle the resulting tag over to the *LiXto Deliverer* component which, in our case, will simply save it in a file.

5 Conclusions

We have described an architecture that allows logic-based analysis and manipulation of Web data. Thanks to *LiXto* suite, the input data can be extracted practically from any Web source. Again thanks to *LiXto*, it remains easy to set up a Web service that supplies the result information over the Web. The core of the application, however, is the execution of sophisticated non-textual filtering operations, based on *inferences* about the source, the text or other issues. Our approach makes possible three kinds of tag manipulation:

1. a tag, obtained from *LiXto*, is filtered, i.e. left out of the final result whenever it is deemed irrelevant;
2. an XML tree, obtained from *LiXto*, gets annotated with extra tags, which would not be otherwise available by text analysis, however sophisticated and
3. filtering and decorations are carried out as a (default) reasoning activity, in the framework of Answer Set Programming with the *smodels* inferential engine.

Even though more test cases are needed for a careful assessment, we believe that the architecture proposed here can become a useful *platform* for the development of tools that act as bridges between the Web as we know it and more sophisticated forms of interactions. We believe that this is the case for the Semantic Web, since our system permits to program and execute the OWL (or RDF) marking up of data.

Hence, our approach could greatly simplify the process of importing Web data into the semantic Web. However, it should be stressed that in any case the import would remain a semi-automatic activity, where human judgment will remain essential in two phases. Let us discuss them now.

The first phase consists in the selection of the Web source and the finding of the required data on the page (document). This phase is assisted and made easier by the *LiXto* visual wrapper, but cannot be automated in full. The second phase of human intervention consists in the writing of the *beautifier rules.* The rules associated to a given Web source in effect represent the semantics of the source itself, and can embed its data inside the Semantic Web. In the follow up of this work we intend to join this research effort with that, reported in [2,3], aiming at a representation of default assumptions directly as a signature over RDF tags. In such a way, the RDF statements *produced* by our system would carry along an indication of the type of default assumption that, possibly, supports them.

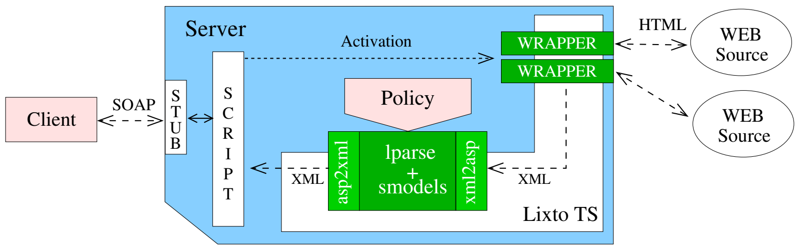

Fig. 3. Our architecture as a Web service.

Acknowledgments

Thanks to R. Baumgartner, E. Bertino, G. Gottlob, A. Mileo and M. Ornaghi. We are grateful to *LiXto* GmbH for giving us an free academic license of *LiXto* suite.

References

1. R. Baumgartner et al. 2005. *Web Data Extraction for Business Intelligence: the LiXto Approach.* Proc. of BTW Workshop.
2. E. Bertino, A. Provetti and F. Salvetti, 2003. *Answer Set Programming for the Semantic Web* In F. Buccafurri (editor) Proc. of *Joint Conference On Declarative Programming.* Univ. of Reggio Calabria Press, pp. 314–323.
3. E. Bertino, A. Provetti and F. Salvetti, 2005. *Reasoning about RDF statements with default rules.* W3C Workshop on Rule Languages for Interoperability. *http://www.w3.org/2004/12/rules-ws/*
4. C. Bernardoni, 2005. *Beyond LiXto: Automated Reasoning for the Annotation of Web Data Sources.* (in Italian) Graduation project in Computer Science, Univ. of Milan.
5. M. Gelfond and V. Lifschitz, 1998. *The stable model semantics for logic programming.* Proc. of the 8th International Workshop on Non-Monotonic Reasoning.
6. G. Gottlob and C. Koch, 2004. *Monadic Datalog and the Expressive Power of Languages for Web Information Extraction.* Journal of the ACM 51, 2004.
7. G. Gottlob et al. 2004. *The LiXto Data Extraction Project – Back and Forth between Theory and Practice.* Proc. of PODS 2004.
8. G. Gottlob, R. Baumgartner, S. Flesca, 2001. *Visual Web Information Extraction with LiXto.* Proc. of VLDB 2001.
9. *LiXto GmbH Website.*
   Web Site: *http://www.lixto.com.*
10. M. Marchi, A. Mileo and A. Provetti, 2004. *Specification and Execution of Policies for Grid Service Selection.* In Proc. of *ECOWS'04.* Springer LNCS 3250, pp 102–115. Available from *http://mag.dsi.unimi.it/PPDL/*

11. Marek, W., Truszczyński, M. 1999: Stable models and an alternative logic programming paradigm. The Logic Programming Paradigm: a 25-Year Perspective Springer-Verlag, pp. 375–398
12. I. Niemelä, P. Simons, and T. Syrjanen, 2000. *Smodels: a system for answer set programming.* Proc. of 5th ILPS Conference, pp. 1070–1080.