Applying ASP Inferential Engines to the Filtering, Decoration and Validation of Data from Web Sources

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

DSI - University of Milan,
Via Comelico, 39/41. I-20135 Milan, Italy,
marchi@dsi.unimi.it
Dept. of Physics, University of Messina,
Sal. Sperone, 31. S. Agata di Messina. I-98166 Messina, Italy,
{*fiumara,ale*}*@unime.it*
http://mag.dsi.unimi.it/

Abstract. We propose a software architecture for semantics-based annotation of data extracted from Web sources. Data can be extracted from arbitrary Web sources thanks to two wrapping engines: the *LiXto* suite, which supports semi-automated data extraction and XML formatting, and *Dynamo*, which is based on the novel approach of HTML meta-tag decoration. The XML tags produced by the wrappers are then fed to an Answer Set Programming inferential engine which applies filters or annotates tags with extra information. Unlike with XSLT-based tag manipulation, inference-based annotations can exploit meta-information (e.g., about the source of the data) or incomplete information to draw conclusions.

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 *default rules,* with Datalog-style syntax and Gelfond-Lifschitz Answer Set semantics [1]. Our architecture can be outlined 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 gathering process is performed by two modules, *LiXto*[1]and *Dynamo*.

The *LiXto* Suite can *wrap* generic HTML pages by inspecting their internal HTML schema. *Dynamo* is a new system [2,3] that we are currently developing around an alternative concept. It can wrap data by discovering special meta-tags that have been (deliberately) embedded within the HTML code[2].

[1] *LiXto* by now is a released software by *LiXto* GmbH: *http://www.lixto.com/*

[2] The deployment of *Dynamo* requires collaboration from the Web source to be polled. However, in some cases a pre-processing phase from the *Dynamo* side can make up

The XML tags generated by *LiXto* and *Dynamo* are then translated into sets of Datalog-syntax [4] *facts;* such facts are then added to the (non-ground) logic programming rules that describe a *tag manipulation policy*. The resulting ASP program is then fed to the Lparse grounder [5] and to the smodels inferential engine; deduction can start. We have defined tag manipulation so as to obtain the following effects (often in combination):

1. some element of the tag, is dropped, e.g., because it is deemed incorrect, uninteresting or superseded by other tags[3]
2. new tags are added, to annotate the present data with extra informations about, e.g., the source, its reliability or some framework information that help in understanding/classifying the data.

The idea here is that the new tags added through inference will bring 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 decorated 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. In this article we restrict to considering RSS channeling.

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 effort (see, e.g., [6]) on *declarative policies* in the context of Web services.

Interestingly, even the wrapping of Web sources is done using the tools developed by the *LiXto* project, which in turn is based on *Elog*, an extension of DATALOG which can described as Positive logic programs + built-in regular expressions. A formal account of the logical interpretation of XML transformations is given by Gottlob and Koch in [7]. *Dynamo*, on the other hand, is a simple Java servlet; it does not need sophisticated pattern matching and manipulation: data polled from Web sources already come in a standard format (albeit an HTML one). To do so, a novel bijective mapping from XML tags to Datalog-syntax facts has been defined; details are in [8].

The automated reasoning and tag manipulation task is carried out in Answer Set Programming (ASP), which can be seen as an extension to Datalog, sometimes called Datalog^{n}ot, that deals with default reasoning. For lack of space,

for the lack of some or all meta-tags. Please refer to [2,3] for a discussion on this issue.

[3] In some cases, the whole tag could be dropped on the basis of consistency considerations. This is the case when tags are coming from Dynamo sources, who may not comply with the agreed data semantics.

we refer the reader to the pioneer works of Gelfond and Lifschitz [9] and to the survey in [1] for a detailed introduction to ASP.

Although our work does not address the Semantic Web as it is commonly understood, i.e., in terms of managing or publishing data annotated with a Semantic Web 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 transform 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. In section 3 we describe the structure of *Dynamo* and its main features. Our architecture is introduced and explained in detail in Section 4. Section 5 describes the application example we have worked on to test and validate our blueprint. Finally, Section 6 summarizes the work done so far and discusses the current development lines.

2 The *LiXto* architecture

The *LiXto Suite* 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.

The VW is defined by Gottlob et al. [10]:

> 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 need to adapt to the input. For further details on how the information extraction works and on its computational complexity, please refer to the presentations in [10,11].

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 *Dynamo*

Dynamo is a new, experimental architecture for automated data collection and XML delivery of data from traditional albeit dynamic HTML Web sites. The data of interest are routinely polled from the actual sources by standard HTTP querying. In order to be able to extract relevant informations from plain HTML documents Bossa [2] defined a set of annotations in form of meta-tags, which can be inserted inside an HTML document in order to give it semantic structure and highlight informational content.

The meta-tags are enclosed in HTML comment tags, so they remain transparent to Web browsers and do not alter the original HTML structure of the document. Once HTML documents are processed by *Dynamo*, raw data and annotations are extracted and organized into a simple XML format which is stored and used as a starting point for document querying and transformation. The first deployment of of *Dynamo*[4]extracts news from two Web sites, namely *theserverside.com* and *java.net*, and publishes RSS1 and RSS2 feeds which, as described, are produced on the fly starting from XML data.

Finally, we show an example of what *Dynamo*produces by showing an HTML fragment taken from *theserverside.com* after the process of insertion of meta-tags (lightly simplified due to the formatting guidelines):

[4] The application described here is the subject of [2] and is running on the site: *http://dynamo.dynalias.org/*. The *Dynamo*source code and more informations can be accessed thereof.

```
<!-- <channel:image  url="http://www.theserverside.com/[...]/feed-logo.jpg"
  title='The Enterprise Java Community[...]'   link="http://www.theserverside.com" /> -->
<!-- <channel:extension  uri="http://purl.org/dc/elements/1.1/"  prefix="dc" localName="language" >
 en-us
</channel:extension> -->
<!-- <channel:title> --> The Enterprise Java Community[...]
  <!-- <channel:link> -->  http://www.theserverside.com<!-- </channel:link> -->
<!-- </channel:title> -->
<!-- <channel:description>-->Enterprise Java Community is a developer community[...]
<!-- </channel:description> -->
<!-- <channel:extension uri="http://purl.org/dc/elements/1.1/" prefix="dc" localName="date"> -->
<!-- </channel:extension> -->
<td colspan="2">
<h1><!-- <item:title index="1"> -->wingS 2.0 web framework released<!-- </item:title> --></h1>
<div class="iteminfo">
Posted by:
<!-- <item:link index="1"> -->
  <a href="/user/userthreads.tss?user_id=194346" title="view Joseph's recent threads [...]">
<!-- </item:link> -->Joseph Ottinger</a>on
<!-- <item:extension index="1" uri="http://purl.org/dc/elements/1.1/"  prefix="dc" localName="date" >-->
    December 08, 2005 @ 08:25 AM
<!-- </item:extension> --></div>
<p>
<!-- <item:description index="1"> -->
The <a href="http://www.j-wings.org/" target="_blank">wingS project</a>
has just released version 2.0 of its framework with lots of major improvements.
<br><br>
wingS is a component based web framework resembling the Java Swing API with its MVC paradigm and [...]
<!-- </item:description index="1"> -->
<br><br>
Version 2.0 comes with a completely rewritten rendering subsystem focusing on optimal
stylability via CSS [...]. Various
<a href="http://www.j-wings.org/[...]/Demo"    target="_blank">demo applications
   are available on-line</a>.
wingS is released under the LGPL license.
</p>
```

and the same fragment converted to XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<resource url="http://dynamo.dynalias.org/tss.jsp" rssId="tss.xml" timestamp="1139592119233">
    <channel>
      <title>Enterprise Java Community: Your Enterprise Java Community</title>
      <link>http://dynamo.dynalias.org/tss.jsp</link>
      <description>Enterprise Java Community is a developer community, containing up-to-date
        news, discussions, patterns, resources, and media</description>
      <image>
        <title>TheServerSide.com</title>
        <url>http://www.theserverside.com/[...]/feed-logo.jpg</url>
        <link>http://www.theserverside.com</link>
      </image>
      <extensions>
        <dc:language xmlns:dc="http://purl.org/dc/elements/1.1/">
          en-us</dc:language>
      </extensions>
    </channel>
    <item id="1139592119233-1" index="1">
      <title>wingS 2.0 web framework released</title>
```

```
      <link>http://feeds.feedburner.com/techtarget/tsscom/home?m=476</link>
      <description>
        The wingS project has just released version 2.0 of its framework with lots of major
        improvements. [...]
      </description>
      <extensions>
        <dc:date xmlns:dc="http://purl.org/dc/elements/1.1/">
          Fri, 10 Feb 2006 09:58:49 EST </dc:date>
      </extensions>
    </item>
    [...]
</resource>
```

4 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 1.

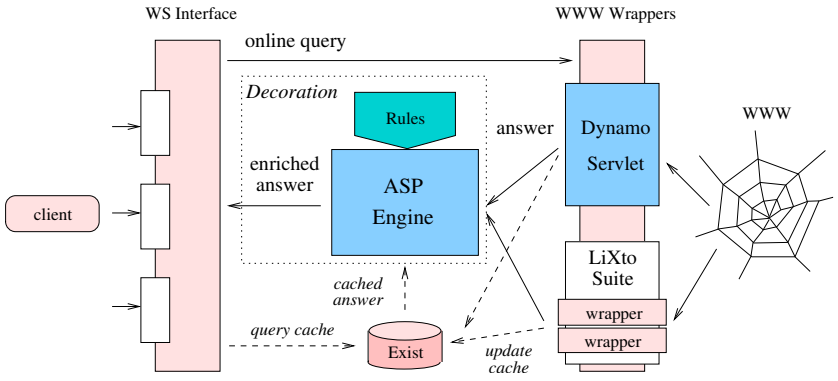

Fig. 1. The internal schema.

First, a simple Web Service collects data requests from its clients. The architecture serves each request by activating the *LiXto* WS. On the contrary, the *Dynamo* servlet is always active and performs routine polls. For the sake of performance, it is also possible to perform *off-line queries* by consulting the internal cache. Such cache is populated by *LiXto* and *Dynamo* throughout continuous scanning of the Web sites that are being monitored. The so-extracted data are then encoded as XML tags. Next, the *decoration* phase starts.

Decoration takes place in several steps. First, a Perl program called *xml2asp* [5] translates XML data into Datalog facts. Second, the obtained facts and the

[5] Both *xml2asp* and *asp2xml* are described in [8] and available from *http://mag.dsi.unimi.it/~carlo/#projects*

rules, i.e., the Datalog-syntax rules that describe tag manipulation, are fed to the ASP inferential engine. In our case, the ASP engine consists of the well-known *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. A detailed description of our architecture is depicted in Figure 2.

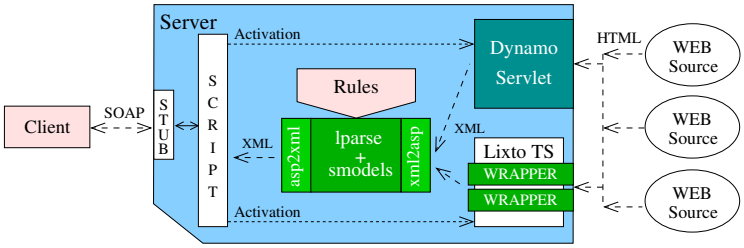

Fig. 2. Our architecture in more detail.

5 An Application example

We are implementing a service that allows a user to monitor hotels availability in big cities and on certain dates, e.g., during conferences. 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 city or even w.r.t. the particular location.

5.1 Wrapping an on-line hotel booking service

The Kelkoo portal[6]for e-commerce allows users, among other things, 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:

– location/resort;

[6] Kelkoo has developed several country-specific portals where customers can compare prices of competing e-commerce sites. In this Section we describe our work with the *kelkoo.co.uk* portal. However, the wrapper and the decorator could be almost effortlessly adapted to other Kelkoo national portals and, with some reprogramming, to other e-commerce sites.

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

– name,
– address,
– brief description of the facilities,
– room type, and
– price.

and organize it into an XML tree. The following is a fragment of an XML output of the wrapper:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <rootPattern>
    <Hotel>
      <Name>HOTEL DE CHAMPAGNE</Name>
      <Address>
            Rue de Faubourg Saint Denis,
            Paris, Paris, 75010 France
      </Address>
      <Region>Paris</Region>
      <Room>
        <Description>Double</Description>
        <Price>34.56</Price>
        <Currency>GBP</Currency>
      </Room>
      <HotelDescription>
          Our hotel is located in the heart of Paris,
          two steps away from the Gare du Nord.
      </HotelDescription>
    </Hotel>
    ...
  </rootPattern>
</document>
```

5.2 Annotating Kelkoo data

The *LiXto transformation server* provides a visual tool for creating pipelines of activities. The pipe meant to "decorate" the information fetched by the wrapper for *kelkoo.co.uk* (see Section 5.1) is depicted in Figure 3 below.

The *Source* component runs the wrapper described in Section 5.1 on a kelkoo.co.uk result page, triggered by a proper querying URL whose parameters' values (see Section 5.1) are user-definable.

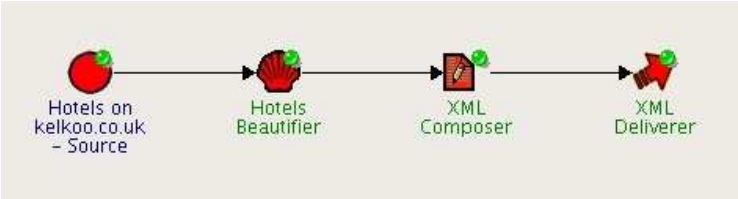

Fig. 3. A smart pipeline for on-line hotel booking on kelkoo.co.uk.

Next, the *Shell* component is configured to invoke (through the Operating System) the *xml2asp* translator, the inferential engine and finally the *asp2xml* back-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:

```
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),
    tag(Currency,"Currency"),
    node(Currency),
    parent_of(Room,Currency),
    isNumber(Price,Qty),
    tag(Region,"Region"),
    node(Region),
    parent_of(Hotel,Region),
    threshold_price(Region,Reg_Currency,Reg_Thld),
    convert_local(Reg_Thld,Reg_Currency,Loc_Thld),
    convert_local(Price,Currency,Loc_Price),
    Loc_Thld < Loc_Price.
```

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 DE CHAMPAGNE</Name>
      <Address>
            Rue de Faubourg Saint Denis,
            Paris, Paris, 75010 France
      </Address>
      <Region>Paris</Region>
      <Room>
        <Description>Double</Description>
        <Price>34.56</Price>
        <Currency>GBP</Currency>
      </Room>
      <HotelDescription>
            Our hotel is located in the heart of Paris,
            two steps away from the Gare du Nord.
      </HotelDescription>
      <Stars>3</Stars>
    </Hotel>

    ...

  </rootPattern>
</document>
```

After the decoration phase, it is possible, again through *Composer* module of *LiXto* TS to force some re-arrangement of the XML tag. In any case, the Composer module handles the resulting tag(s) over to the *LiXto Deliverer* component which takes care of forwarding them to the clients. In the example described above, we chose simply to save the resulting tags in a database. To facilitate further reuse, and beside to test the viability of the solution, we chose to employ the native XML database Exist [13] for the accumulation of the tags created

by our system. A much larger experimental tests, involving thousands of polling cycles a day, is currently under way to validate the approach in a realistic Web scenario.

For the sake of simplicity, this example shows only the simplest decoration activity, i.e., the adding of new tags. For the example above, the full inferential power (and therefore complexity) of Answer Set Programming is not needed. However, it is easy to imagine situations where a decision about whether to add, change or remove tags within a given, extracted tag will be based i) on lack of present data, which could be addressed by stratified negation as failure or ii) by reasoning by cases, which could be addressed by the generation of alternative answer sets.

6 Conclusions

We have described an architecture that allows analysis and manipulation of Web data based on default inferences with ASP, which is the core concept of our system. Indeed, 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 ASP programs [1]. In particular, Eiter et al. [14] have discussed the application of ASP to reasoning about data from the Web.

Thanks to *LiXto* suite, the input data could be extracted –albeit with some human intervention– practically from any Web source. Also, thanks to *Dynamo*, old-fashioned HTML Web sources can participate to our data collection. 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 itself or other issues. Our approach makes three kinds of advanced tag manipulation possible:

1. a tag, obtained from *LiXto* or *Dynamo*, is filtered, i.e. left out of the final result whenever it is deemed irrelevant;
2. a tag, again obtained from *LiXto* or *Dynamo*, 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.

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 seems unlikely to become fully automated. Note that in case of a *collaborative* Web source, the human intervention is even more important as meta-tags are to be inserted in the original Web HTML documents. 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 [6], 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 assumptions that, supports them.

Acknowledgments

This project was started with contributions from S. Bossa and C. Bernardoni's graduation projects. Thanks to R. Baumgartner, G. Gottlob and M. Ornaghi. We are grateful to *LiXto* GmbH for granting us an academic license of *LiXto* suite.

References

1. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. The Logic Programming Paradigm: a 25-Year Perspective, Springer-Verlag (1999) 75–398
2. Bossa, S.: Polling of arbitrary web sources. Graduation project in Computer Science (in Italian), Univ. of Messina. (2005)
3. Bossa, S., Fiumara, G., Provetti, A.: A ligthweight architecture for rss polling of arbitrary web sources. Submitted for pubblication. Available from *http://mag.dsi.unimi.it/* (2006)
4. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive datalog. ACM Transactions on Database Systems (TODS) **22(3)** (1997) 364–418
5. Solvers: Web location of some of the most known asp solvers. (
   Aspps: *http://cs.engr.uky.edu/ai/aspps/*
   CMODELS: *http://www.cs.utexas.edu/users/tag/cmodels.html*
   DLV: *http://www.dbai.tuwien.ac.at/proj/dlv/*
   NoMoRe: *http://www.cs.uni-potsdam.de/~linke/nomore/*
   Smodels: *http://www.tcs.hut.fi/Software/smodels/*
   PSmodels: *http://www.tcs.hut.fi/Software/smodels/priority/*)
6. Bertino, E., Provetti, A., Salvetti, F.: Reasoning about rdf statements with default rules. In: Rule Languages for Interoperability, W3C (2005)
7. Gottlob, G., Koch, C.: Monadic datalog and the expressive power of languages for web information extraction. Journal of the ACM **51** (2004)

8. Bernardoni, C.: Beyond *LiXto*: Automated reasoning for the annotation of web data sources. Graduation project in Computer Science (in Italian), Univ. of Milan. (2005)
9. Lifschitz, V., Gelfond, M.: The stable model semantics for logic programming. Proc. of 5th ILPS conference (1988) 1070–1080
10. Gottlob, G., Baumgartner, R., Flesca, S.: Visual web information extraction with lixto. Proc. of VLDB (2001)
11. Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., Flesca, S.: The lixto data extraction project - back and forth between theory and practice. In Deutsch, A., ed.: PODS, ACM (2004) 1–12
12. Niemelä, I., Simons, P., Syrjänen, T.: Smodels: a system for answer set programming. Proc. of the 8th International Workshop on Non-Monotonic Reasoning (ILPS) (2000) 1070–1080
13. Meier, W.: exist: An open source native xml database. In Chaudhri, A.B., Jeckle, M., Rahm, E., Unland, R., eds.: Web, Web-Services, and Database Systems. Volume 2593 of Lecture Notes in Computer Science., Springer (2002) 169–183
14. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: A generic approach for knowledge-based information-site selection. In Fensel, D., Giunchiglia, F., McGuinness, D.L., Williams, M.A., eds.: KR, Morgan Kaufmann (2002) 459–469