Search, Analysis, and Integration of Web Documents: A Case Study with FLORID

Rainer Himmeröder Paul-Th. Kandzia
Bertram Ludäscher Wolfgang May Georg Lausen
Institut für Informatik, Universität Freiburg, Germany
{himmeroe,kandzia,ludaesch,may,lausen}@informatik.uni-freiburg.de

Abstract

Languages supporting deduction and object-orientation seem particularly promising for querying and reasoning about structure and contents of the Web, and for the integration of information from heterogeneous sources. FLORID, an implementation of the deductive object-oriented language F-logic, has been extended to provide a declarative semantics for querying the Web. This extension allows extraction and restructuring of data from the Web and a seamless integration with local data. Since the functionality of wrappers and mediators is integrated into a single declarative language, the development of advanced applications based on the Web as an information source is significantly simplified. This claim is substantiated using a comprehensive example.

1 Introduction

The enormous impact of the World-Wide Web has originated novel questions, but also renewed the interest in problems which have been studied before in different contexts. Obviously, during the design of a non-trivial Web application, various research areas may be involved: For example, the recent work on *semistructured data* has discovered the Web as an important and interesting example [Abi97, Bun97, Suc97]. Typical features of semistructured data include the following: the structure is irregular, partial, unknown, or implicit in the data. One research goal is to find an appropriate data model and query language. In contrast to full-fledged object-oriented languages, languages for semistructured data do not allow to benefit from known structure or to handle "meta-knowledge" like schema information in an adequate way.

Closely related to semistructured data is the research on *Web query languages*. Languages like W3QS [KS95] and WebSQL [MMM97] allow to specify conditions which refer to both, the (hyperlink) structure and the contents of Web documents. However, in this area the focus is on efficient searching of Web documents, not on their further processing and organization.

The main interest in the area of *heterogeneous databases* [GMPQ+97, AMM97, LRO96] lies on the semantic integration of different data sources. The typical architecture consists of *wrappers* which map data sources to a common data model, on top of which *mediators* take care for the semantical integration. In this context, the search for relevant Web documents is not covered. Moreover, in advanced applications, the hierarchical architecture may not be flexible enough, e.g., when wrappers for additional sources are required depending on the results of a mediator, or when a data-driven wrapper can profit from the knowledge of a mediator.

WebOQL [AM98] is designed as a unified platform for the aforementioned tasks and uses a graph-based data model. Thus, like other languages for semistructured data, it does not allow to reorganize data into a rich object-oriented data model, which is often desirable from the user's viewpoint.

Several approaches take *Logic Programming* and *Deductive Databases* as a starting point for Web (query) languages in order to support the restructuring of documents [BHT97]. Unfortunately, to our knowledge, there is no language integrating Web access with a clear and declarative semantics. Moreover, object-oriented modeling of data, which has proven useful for various tasks when managing Web documents, is not directly supported by languages in the style of Prolog or Datalog. In contrast, ADOOD [GMP97, GMNP97] and WebLog [LSS96] add object-orientation to logic-based Web management. However, WebLog has not been implemented (yet); moreover, both languages do not provide a formal semantics for the actual Web access.

In this paper we present some experiences in designing Web applications with FLORID. FLORID[1] (**F-LO**gic **R**easoning **I**n **D**atabases) is an implementation of F-logic with *path expressions* [FLU94], which turn out to be particularly useful for navigating on semistructured data. All deductive and object-oriented features of F-logic are supported by the system. Recently, FLORID has been extended to provide a declarative semantics for querying Web documents [HLLS97, HLLS98]. The proposed extension allows extraction and restructuring of data from the Web and a seamless integration with local data. A main advantage of the approach is that it brings together the above-mentioned issues in a unified, formal framework and supports rapid prototyping and experimenting with all these features. In particular, FLORID programs may be used (simultaneously) as wrappers, mediators, or for "ordinary" queries. Hence, we claim that FLORID is quite convenient for programming all query aspects of a Web application.

2 Preliminaries: F-Logic in a Nutshell

We briefly review the basic constructs of F-logic and its extension by path expressions; for details see [KLW95] and [FLU94, HLM+], respectively. Consider the following fragment of an F-logic program:

```
person[name ⇒ string; children@(integer) ⇒⇒ person].   % signature of class person
employee::person.                                      % subclass relationship
john:employee[                                         % instance relationship and
    name → "John Smith"; children@(1998) →→ {mary,bob}]. % ... some example data
X.father:man  :– X:person.                             % object creation by ...
X.mother:woman  :– X:person.                           % ... path expressions
```

First, the *signature* of class person is specified: The *single-valued* method name yields instances of class string, whereas the *multi-valued* (and parameterized) method children yields instances of person. The *subclass* relation employee::person states that all members of employee are also members of person. Next, john:employee defines that the object named john is an *instance* of class employee; the specification inside [...] defines the actual data values for name and children.

The last two rules demonstrate how path expressions in the head can be used to create new *object identifiers* (*oids*): If X is bound to an instance of person, then the single-valued method father becomes defined for X. The newly "created" father is referenced by the *path expression* X.father and is made an instance of man. In particular, john[father → john.father] and (john.father):man hold (similarly for mother and woman). Thus, the dot "." corresponds

[1] Available through [FLO].

to navigation along single-valued methods ($\rightarrow$) like name and father, while ".." is used to navigate along multi-valued methods ($\twoheadrightarrow$) like children, e.g., as in john..children@(Y)[...].

The use of single-valued path expressions for object creation is crucial for our approach to Web exploration (see below). Object creation has to be used with care in order to avoid infinite universes and nontermination: e.g., the rule X.father:man :- X:man, if added to the program, creates an infinite number of objects john.father, john.father.father, ...

3 Exploring the Web with F-Logic

The Web Model. We adopt the usual model, where the Web is conceived as a graph-like structure consisting of documents and links between them. More precisely, we distinguish between the class url of (potential) urls, and webdoc of (accessed) Web documents. Urls are instances of string, for which a special method get is definable (see below). Typical elements of class webdoc are SGML/HTML pages, but other document types may also be included (e.g., BibTeX, ASCII text, etc.). If a Web document has been accessed, a number of *system-defined* methods may become defined for it: e.g., url (the url of the Web document), author, modif (time of last modification), type, and—most notably—the multi-valued method hrefs@(*label*) representing the outgoing links of the Web document (see Fig. 1). Note that hrefs is parameterized with the label of the link.[2] If the Web access fails, error returns the reason of the failure (e.g., *server does not exist*, *page not found*, etc.).

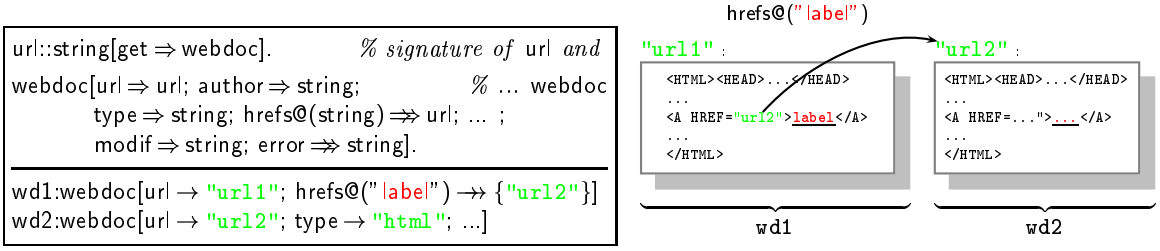

Figure 1: F-logic Web model: signature and example data

Web Exploration. A Web document is accessed and added to the local F-logic database by defining the method get for an instance u of class url in the head of a rule, thereby creating the new oid "u.get" of the fetched Web document. After the oid u.get has been created, system-defined methods are automatically "filled in" by FLORID, and the Web document named u.get becomes an ordinary F-logic object (conceivable as a large string). Thus, u.get is "cached" and the url u is accessed only once. Note that the *potentially* system-defined methods for class webdoc are fixed by the FLORID system—the *actually* system-defined (i.e., "filled") methods for u.get depend on the result of accessing the url u. For example, if error is defined, then hrefs@(...) will be undefined. Here, the advantages of using an object-oriented framework like F-logic become apparent: although the instances of a certain class may typically define a certain set of methods, some (or all) of these methods may be omitted. Thus, the use of NULL values as in the relational model can be avoided.

Apart from this first document analysis done by the system, the main power of the approach lies in the possibility to use all features of F-logic (and path expressions) for fetched documents.

[2] Sometimes also the offset of the link in the document may be useful [MM97].

Declarative Semantics. Typically, browsing the Web means to access documents depending on some information (in the most simple case an url), using the newly derived information of those documents to find new documents, and so on. Clearly, this principle of data-driven Web exploration lends itself to a direct integration with the forward-chaining style of reasoning as employed in deductive databases. Thus, a seamless integration with the bottom-up evaluation strategy of FLORID and a declarative semantics of the language is achieved [HLLS97]. The special semantics of Web access can be integrated by adding a function *explore* which describes the semantics of an url using a subset of the *extended Herbrand base* $\mathcal{HB}^\star$.[3] For example, for "url1" in Fig. 1, *explore* yields facts of the form

$$\text{"url1".get[type} \rightarrow \text{"text/html"]},\ \text{"url1".get[hrefs@("label")} \twoheadrightarrow \{\text{"url2"}\}],\ \text{"url2":url},\ \ldots$$

The immediate consequences operator T_P can be easily adapted to incorporate this semantics, simply by adding the corresponding facts for all documents which have been "requested" via get:

Let $\mathcal{URL}$ be the set of all url's, $\mathcal{R}$ a set of *reserved names* (0-ary functors), and $\mathcal{HB}^\star$ the extended Herbrand base. $\mathcal{R}$ contains the names for system-defined methods; here, at least the methods url, get, hrefs, and errors. Then, a *Web Interface* $\mathcal{W}$ is a tupel $(\mathcal{R}, \mathit{explore})$, where

$$\mathit{explore} : \mathcal{URL} \rightarrow 2^{\mathcal{HB}^\star}$$

is a function mapping each $u \in \mathcal{URL}$ to a set of new facts (representing what is known after accessing and analyzing u). A *Web interpretation* wrt. a Web interface $\mathcal{W}$ is a Herbrand interpretation $\mathcal{H} \subseteq \mathcal{HB}^\star$ that additionally satisfies the *Web access axiom*:

$$\text{for all } u \in \mathcal{URL},\ \text{if } u\text{:url}, u.\text{get} \in \mathcal{H} \text{ then } \mathit{explore}(u) \subseteq \mathcal{H}\ .$$

Then, the computation of the (inflationary) Herbrand Web model is integrated into FLORID's bottom-up evaluation by the following extended immediate consequence operator:[4]

$$T_P^{\mathcal{W}}(\mathcal{H}) := \mathcal{H} \cup T_P(\mathcal{H}) \cup \bigcup\nolimits_{u:\text{url}, u.\text{get} \in T_P(\mathcal{H})} \mathit{explore}(u)\ .$$

4 Designing a Web Application with FLORID

To demonstrate the power and flexibility of FLORID for designing advanced Web applications, we present an example which integrates data from two different sources, the *CIA World Factbook* and *World Online*. This and further examples can be obtained (and executed) via the FLORID homepage [FLO]. Although one may argue that the data in those sources is quite regular and should be put on the Web as a database in the first place, such a database is *not* available from the Web, whereas the semistructured HTML pages are. Moreover, the data actually contains some irregularities and idiosyncrasies which have to be considered when extracting information.

The structure of the FLORID program below mirrors a kind of "methodology" that is frequently encountered in the context of integration of Web sources: After accessing/browsing

[3] $\mathcal{HB}^\star$ is defined over an extended Herbrand universe $U^\star$ containing, in addition to the usual constants, so-called *pure references* like john.father.mother or "url1".get; see [HLLS97, HLM$^+$].

[4] In addition to inflationary semantics, FLORID also supports user-defined stratification, which is useful for programs involving negation, or to enforce a certain evaluation order. Here, for simplicity, we assume that P also includes F-logic's closure axioms [KLW95].

the pages of interest, the "raw" remote data can be retrieved and then reorganized, thereby obtaining a local (re)structured database. When this has been done for all data sources, an integration step follows in which the data from the different sources has to be correlated, for example, by identifying synonyms for common real world entities.

4.1 Accessing Entry Pages

In our example this task is not very difficult, since the urls of the interesting Web-servers and their principal structure are known in advance. Nevertheless, the data structuring facilities of F-logic turn out to be useful:

First, the url of the *CIA World Factbook* and of a local mirror are defined for the object cia, our "entry" object to the World Factbook:[5]

```
cia[home→"http://www.odci.gov/cia/publications/nsolo/factbook/";
    mirror→"http://www.informatik.uni-freiburg.de/~dbis/.mir/ciawfb/"].
```

To allow for easy substitution of the data source, a generic name cia.src is defined:

```
cia[src→cia.mirror].      % use the local mirror
```

On the top-level, the World Factbook is organized as a set of pages, one page for each continent. The following facts define the local names of those pages as the result of the method file@(cia) on continent names, which additionally are made instances of class continent:

```
"Europe":continent[file@(cia)→"eur.htm"].
"Asia":continent[file@(cia)→"asia.htm"].
    ⋮
"Antarctic Region":continent[file@(cia)→"ant.htm"].
```

The actual url U of the page for a continent C is obtained by concatenating (via the built-in strcat) the CIA root url and C's file name FN. Each such U is made an instance of class url and accessed by defining the method get for it:

```
C[url@(cia)→U]  :-                                  % define the continents' urls
    C:continent[file@(cia)→FN],                     % ... using the filename
    strcat(cia.src,FN,U).                           % ... and the CIA source url
U:url.get  :- C:continent[url@(cia)→U].  % add urls to class url and get all continent pages
```

As mentioned in Section 3, the method hrefs@(*label*) is automatically defined by the system (unless an error has occurred) for all accessed Web documents (i.e., for which get is defined). The links found on a continent page refer to the country pages of this continent, hence they are used to populate a class country. The result of the following rules is needed for further access to Web sites; hence, this represents an example of a mutual dependency between Web search and information found on Web documents:

```
cid(cia,C):country[url@(cia)→U; name@(cia)→C; continent→CT]  :- % define new country objects
   CT:continent.url@(cia).get[hrefs@(Label)↠U],          % ...using labels of continent pages
   match(Label,"\(.*\) ([0-9]","\1",C).                  % ...if they have a certain form
```

[5] The local mirror is not only useful when the main site is unavailable, but also when the original Web pages are completely reorganized. Then the mirror can be used until the "wrapper part" of the FLORID program has been adjusted to the new structure.

For each continent CT, the labels and their corresponding urls U are inspected: If the label has a certain form (e.g., "Spain (32 KB)"), then the country name can be extracted from the label using a regular expression and the built-in match (Section 4.2). We use a Skolem functor cid to construct a new oid (internal name) for each country. Since we will later use another data source for countries, we add "cia" as a parameter to the oid. The reason for parameterizing the single-valued method name (which holds the external name of the country) will become clear when country objects from different source are fused into a single object; see Section 4.4.

Finally, the individual country pages can be accessed by making their urls instances of class url and defining get for them:

```
U:url.get  :- C:country[url@(cia) → U].   % access all cia countries
```

4.2 Retrieving "Raw" Data

The penultimate rule of the previous section (defining new country objects cid(cia,C)) is an example where data is extracted using the *link structure* of Web pages. Although such hyperlinks (represented by the method hrefs) and other structural features often carry meaningful information, often also textual components of a document have to be analyzed.

To this end, built-in predicates for extracting and analyzing data from accessed Web documents have to be provided. A simple, yet flexible and powerful approach used in FLORID and also in many other systems, is to view Web documents as (large) strings. Then, using *regular expressions*, patterns in Web documents can be easily exploited, e.g., to extract all strings between pairs of HTML tags like <h2> and </h2> (level-2 headings), or to analyze tables or lists. The regular expressions employed in FLORID include groups and format strings thereby providing a quite expressive language: The predicate

match(*Str*, *RegEx*, *Fmt*, *Res*)

finds all strings in the input string *Str* which match the pattern given by the regular expression *RegEx*. The *format string Fmt* describes how the matched strings should be returned in *Res*. This feature is particularly useful when using *groups* (expressions enclosed in \(...\)) in regular expressions. For example,

```
?- match("Time heals all wounds", "\(.*\) heals \(.*\) \(.*\)", "\1 \3 \2 heels", X).
```

yields X="Time wounds all heels". Instead of *Fmt* and *Res*, also lists of format strings and result variables can be given.

Continuing the World Factbook example, data from the country pages can be extracted and stored in the F-logic database as follows:

```
C[capital → X]          :- match(C:country.url@(cia).get, "Capital:.*\n\(.*\)", "\1", X).
C[total_area → X]       :- match(C:country.url@(cia).get, "total area:.*\n\(.*sq km\)", "\1", X).
C[external_debt → X]    :- match(C:country.url@(cia).get, "External debt:.*\n\(.*\)", "\1", X).
```

These rules show a strong regularity. Thus, one can take advantage of the meta-programming facilities of F-logic (here: variables at method position) and replace the code by a single generic rule and facts describing the used patterns:

```
C[Method → X]  :- pattern(Method, RegEx), match(C:country.url@(cia).get, RegEx, "\1", X).
pattern(capital,"Capital:.*\n\(.*\)").
pattern(total_area,"total area:.*\n\(.*sq km\)").
pattern(external_debt,"External debt:.*\n\(.*\)").
```

Clearly, such a "pattern-base" may be extended easily for other methods.

4.3 Organizing and Extracting Information

Once the necessary sites are found and interesting information is extracted, it can be organized in an object-oriented way. If –as usual– the amount of data is large or not clearly structured, an interactive exploration may be very helpful to find an appropriate representation.

Queries like the following give a first impression of the World Factbook:

```
?– N = count{C ; C:country}.                          % use aggregation to count the countries (Q1)
?– _:country[name@(cia) → N; capital → C].            % name all countries and their capitals (Q2)
```

Query (Q1) yields N=266 countries (see [FHKS97] for details on aggregation in FLORID). However, (Q2) outputs a binary relation (Country,Capital) with only 256 entries (note, that as in Prolog, variables starting with an underscore are conceived as existentially quantified and are not shown in the answer).

Data Cleaning. To shed some light on this apparent contradiction, the next query reveals the 10 "countries" for which the method capital is not defined:

```
?– _C:country[name@(cia) → N], not _C.capital.  % (Q3)
```

Among others, answers are "Antarctica", "Atlantic Ocean", and "World". Moreover, it turns out that there are some "countries" which have the method capital defined, yet do *not* have a proper capital. For example, for "Bouvet Island" the method capital yields the result "none; administered from Oslo, Norway". Thus, we may specify the class of *real* countries as follows:

```
C:real_country ← C:country[capital → CA], not substr("none", CA).
```

Now, the query ?– C:country, not C:real_country discloses 25 more "false countries" (apart from the 10 above) like "Bouvet Island", "Clipperton Island", and "Western Sahara".

A characteristic advantage of F-logic is the possibility of *schema browsing*. Consider the following queries:

```
?– _:country[M → _].                                  % what scalar methods are defined for countries? (Q4)
?– _:country.M, _C:country[name@(cia) → N], not _C.M. % what methods are undefined for C? (Q5)
```

Query (Q4) yields all single-valued methods (capital, total_area, land_area, etc.) potentially defined for countries (i.e., defined for at least *some* country). The first literal _:country.M of (Q5) is a syntactic variant of (Q4); together with the rest of (Q5), "countries" C with undefined methods M are reported: e.g., for C="Ashmore and Cartier Islands", the method M="labor_force" is undefined.

Extracting Structured Data. Often, data sources already contain some structured data, e.g., in the form of lists (like HTML lists or *comma(-separated) lists* in plain text). In the World Factbook, ethnic groups, religions, and languages are given as comma lists of the form

```
Ethnic Divisions: Pashtun 38%, Tajik 25%, Uzbek 6%, Hazara 19%,
minor ethnic groups (Aimaks, Turkmen, Baloch, and others).
```

Such information can be extracted in two steps. First, the whole comma list after Ethnic Divisions is read as a string (and augmented with an additional ","). In the second step, individual list elements are extracted from the string. For every pair (*country, group*), a new object with oid ethgrp(*country, group*) is created, which provides the methods name and perc:

```
C[egrps→L2] :-
    match(C:country.url@(cia).get, "Ethnic divisions:.*\n\(.*\)","\1", L), strcat(", ", L, L2).
C[ethnicgroups↠ethgrp(C,G)[group→G; perc→P]] :-
    match(C.egrps," , \([^0-9,]*\)\([^a-zA-Z]*\)\%",["\1","\2"], [G,P]).
```

Here again, *generic* rules and patterns can be given to extract information from *several* comma lists, (here: ethnic divisions, languages, and religions):

```
C[M→L2] :-
    commapattern(M,F,S,N), strcat(S,":.*\n\(.*\)",S2),
    match(C:country.url@(cia).get, S2, "\1", L), strcat(", ",L,L2).
C[M↠cr(F,C,X)[N→X; perc→P]] :-
    commapattern(M,F,S,N),
    match(C.M," , \([^0-9,]*\) \([^a-zA-Z]*\)\%",["\1","\2"],[X,P]).

commapattern(ethnicgroups,egrp,"Ethnic divisions",group).
commapattern(languages,lg,"Languages",lang).
commapattern(religions,rel,"Religions",rel).
```

Then, e.g., the ethnic groups of the United Kingdom can be queried:

```
Answer to: ?- _[name@(cia)→"United Kingdom"]..ethnicgroups[group→G; perc→P].
P/"2.8" G/"and other"
P/"81.5" G/"English"
P/"9.6" G/"Scottish"
P/"2.4" G/"Irish"
P/"1.9" G/"Welsh"
P/"1.8" G/"Ulster"
```

4.4 Integration of Different Sources

In the running example, we have shown how the country information of the World Factbook can be retrieved and "cleaned". However, often it is not sufficient to use only one source of data; rather information from different sources has to be integrated and restructured. Programming such problems again benefits very much from the features provided by F-logic. The World Factbook gives no information about cities. However, such data for the main cities of a country can be found on another Web server, the *World Online* server. Below, we illustrate how data from World Online can be integrated with World Factbook data.

The first steps are very similar as those for the World Factbook, since the top-level organization of the server is analogous. First, the server location as well as the needed pages are defined and the country pages are accessed. Note that continents have a unique name in the Web sources considered here, hence can be used directly as oids.

```
wol[src→"http://home.worldonline.nl/~quark/"].

"Europe"[file@(wol)→"europe/euix.htm"].
"Asia"[file@(wol)→"asia/asix.htm"].
    ⋮
"Africa"[file@(wol)→"africa/afix.htm"].

C[url@(wol)→U] :- C:continent[file@(wol)→A], strcat(wol.src,A,U).
U:url.get :- C:continent[url@(wol)→U].
```

When accessing the country pages (which are found by following all links except those containing certain words in the label), an irregularity has to be considered (in the sequel the discussion is restricted to Europe): There are two alternative pages for Germany, hence the rule excludes one of them (the one which contains a "2" in the name):

```
wol[exclude →» {"home","general data","population","economy","Profiler"}].
cid(wol,C):country[url@(wol) → U; continent → Cont; name@(wol) → C]  :-
    Cont:continent.url@(wol).get[hrefs@(C) →» U],
    not substr(wol..exclude,C), not substr("2",U).
U.get  :- C:country[url@(wol) → U].
```

Now the information about the main cities of a country is retrieved from World Online's country pages. Oids for newly created instances of class city are defined using the Skolem functor cty. Note that we have to include both the city name and the country name in this oid to ensure uniqueness:

```
C[main_cities →» cty(CN,C):city[name@(wol) → CN; population → P]]  :-
  C:country[url@(wol) → U],
  match(U.get, "<..><..>.*</..><.. .....=right>.*</..></..>\n", ["\1","\2"],[CN,P]),
  not substr("align",CN), not substr("ALIGN",CN).
```

Object Fusion. The connection between data from both Web sources is established using country names. However, we have to deal with synonyms, since names are not unique over different sources, e.g., "Czech Republic" (World Factbook) vs. "Czech Rep." (World Online). Here, a simple heuristic is used to merge the information on countries: Different pages are about the same country, if they lie in the same continent and (i) the name of the country is the same in both sources, or (ii) the capital according to the World Factbook is one of the main cities according to World Online. If those conditions are satisfied, the objects describing the same country can be *fused* by equating their oids. As a result of such a derived equation, we get a single fused object combining all properties of both objects. Since the single-valued method name has been parameterized with the data source (cia or wol), no violation of the functionality constraint occurs. Note that this holds only under the assumption that country names are unique *within* the World Factbook and World Online, respectively.

```
C1 = C2  :-
    C1:country[continent → Cont; name@(S1) → N],
    C2:country[continent → Cont; name@(S2) → N], not S1=S2.
C1 = C2  :-
    C1:country[continent → Cont]..main_cities[name@(wol) → N],
    C2:country[continent → Cont; name@(cia) → _; capital → N].
```

To check which country objects have been fused, one can use the following query:

```
Answer to: ?- cid(cia,X) = cid(wol,Y), not X = Y.
X/"Czech Republic" Y/"Czech Rep."
X/"Germany" Y/"Germany I"
```

Now the user can freely query the integrated information from both sources, e.g., the capital of Germany together with its population:

```
Answer to: ?- _:country[name@(S1) → "Germany"; capital → N], _:city[name@(S2) → N; population → P].
N/"Berlin" S1/cia P/"3,472,009" S2/wol
```

5 Conclusion

We have demonstrated how several Web data management problems can be handled using the FLORID system. In this paper, we have focused on Web querying and information integration; the handling of semistructured data is detailed in [HLM+]. In contrast to other approaches, which typically consider the various aspects (i.e., Web querying, semistructured data, and information integration) in isolation, we propose a unified declarative framework based on F-logic.

A limitation of the current implementation is the exclusive use of regular expressions for analyzing Web documents. We are currently extending the Web model and implementation of FLORID to include the SGML (resp. XML) structure of Web documents using the SGML parser SP[6].

References

[Abi97] S. Abiteboul. Querying Semi-Structured Data. In *Intl. Conference on Database Theory (ICDT)*, number 1186 in LNCS, pp. 1–18. Springer, 1997.

[AM98] G. O. Arocena and A. O. Mendelzon. WebOQL: Restructureing Documents, Databases, and Webs. In *Intl. Conference on Data Engineering (ICDE)*, 1998.

[AMM97] P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. In *Intl. Conference on Very Large Data Bases (VLDB)*, 1997.

[BHT97] K. D. Bosschere, M. Hermenegildo, and P. Tarau, editors. *2nd Intl. Workshop on Logic Programming Tools for Internet Applications*, 1997. in conjunction with ICLP'97, Belgium, http://clement.info.umoncton.ca/~lpnet/iclp97.

[BRR97] F. Bry, K. Ramamohanarao, and R. Ramakrishnan, editors. *Intl. Conference on Deductive and Object-Oriented Databases (DOOD)*, number 1341 in LNCS, Montreux, Switzerland, 1997. Springer.

[Bun97] P. Buneman. Semistructured Data (invited tutorial). In *ACM Symposium on Principles of Database Systems (PODS)*, pp. 117–121, Tucson, Arizona, 1997.

[FHKS97] J. Frohn, R. Himmeröder, P.-T. Kandzia, and C. Schlepphorst. How to Write F-Logic Programs in FLORID, Version 2.0, 1997.

[FLO] The FLORID Home Page. http://www.informatik.uni-freiburg.de/~dbis/florid/.

[FLU94] J. Frohn, G. Lausen, and H. Uphoff. Access to Objects by Path Expressions and Rules. In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Intl. Conference on Very Large Data Bases (VLDB)*, Santiago de Chile, 1994.

[GMNP97] F. Giannotti, G. Manco, M. Nanni, and D. Pedreschi. Datalog++: A Basis for Active Object-Oriented Databases. In Bry et al. [BRR97].

[GMP97] F. Giannotti, G. Manco, and D. Pedreschi. A Deductive Data Model for Representing and Querying Semistructured Data. In Bosschere et al. [BHT97]. in conjunction with ICLP'97, Belgium, http://clement.info.umoncton.ca/~lpnet/iclp97.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2), 1997.

[6] http://www.jclark.com/sp/

[HLLS97] R. Himmeröder, G. Lausen, B. Ludäscher, and C. Schlepphorst. On a Declarative Semantics for Web Queries. In Bry et al. [BRR97], pp. 386–398.

[HLLS98] R. Himmeröder, G. Lausen, B. Ludäscher, and C. Schlepphorst. *FLORID*: A DOOD-System for Querying the Web. In *Demonstration Session at EDBT*, Valencia, 1998.

[HLM+] R. Himmeröder, B. Ludäscher, W. May, C. Schlepphorst, and G. Lausen. Managing Semistructured Data with *FLORID*: A Deductive Object-Oriented Perspective. submitted.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, July 1995.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A Query System for the World-Wide Web. In *Intl. Conference on Very Large Data Bases (VLDB)*, 1995.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogenous Information Sources Using Source Descriptions. In *Intl. Conference on Very Large Data Bases (VLDB)*, 1996.

[LSS96] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A Declarative Language for Querying and Restructuring the Web. In *Proc. Sixth International Workshop on Research Issues in Data Engineering (RIDE)*, 1996.

[MM97] A. Mendelzon and T. Milo. Formal Models of Web Queries. In *ACM Symposium on Principles of Database Systems (PODS)*, 1997.

[MMM97] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World Wide Web. *Intl. Journal on Digital Libraries (JODL)*, 1(1), 1997.

[Suc97] D. Suciu, editor. *Proc. of the Workshop on Management of Semi-Structured Data (in conjunction with SIGMOD/PODS)*, Tucson, Arizona, 1997. http://www.research.att.com/~suciu/workshop-papers.html.