

Techniques and Rule Patterns for Declaratively Querying Web Data with *FLORID*

Bertram Ludäscher Rainer Himmeröder Wolfgang May

Institut für Informatik, Universität Freiburg, Germany

Overview

- Introduction
- *FLORID* Web model
- Integration of Web Access with DOOD paradigm
- Data Integration: A Case Study
- Navigation
- Conclusions

Goal: A uniform framework/system for

- *Querying the Web:*
 - express declaratively how to query/navigate on the Web
 - extract data from Web pages for populating a database (*Web-data warehousing*)
 - *Management of Semistructured Data:*
 - structure is irregular, partial, unknown, implicit in the data
 - example: HTML pages
 - querying/navigation using *general path expressions* (both in the web (via links) and in the database)
 - discover structure
 - *Information Integration:*
 - heterogeneous sources with different structure
 - wrappers, mediators
-

DOOD Paradigm:

- *deduction*: data-driven exploration of the Web and high level querying
- *object-orientation*: flexible modeling of semistructured data (optional methods instead of NULLs)



Web-FLORID: extension of *F-logic* for querying and restructuring the Web:

- declarative rule-based programming style: uniform language for wrappers & mediators
 - meta features: schema browsing/reasoning, variables at class/method positions
 - restructuring of information
 - navigation by (general) path expressions
 - uniform access to local db & Web data \Rightarrow integration of heterogenous information
-

Basic Constructs:

```
Object:Class
SubClass::Class
```

% ISA-relation, "∈"
% SUBCLASS-relation "⊆"

```
Class[Method@(P-types) => R-type ]
Class[Method@(P-types) =>> R-types ]
```

% SIGNATURE: single-valued
% ... and multi-valued

```
Object[Method@(Params) -> R ]
Object[Method@(Params) ->> {R1,R2} ]
```

% DATA: single-valued
% ... and multi-valued

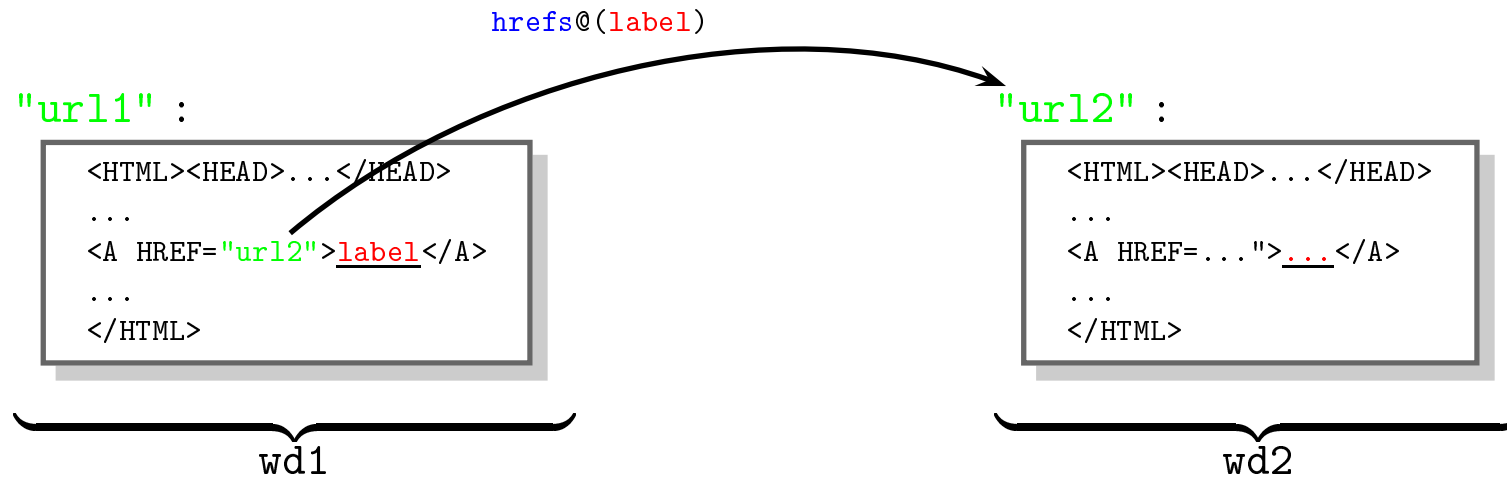
```
Obj .M1@(P1)[Spec1] .M2@(P2)[Spec2]
```

% PATH EXPRESSION

Object Creation via Path Expressions in the Head:

```
X.father:man ← X:person.
X.mother:woman ← X:person.
?-_:person.M:C.
M=father, C=man;
M=mother, C=woman
```

The Web = Graph, consisting of *nodes* (urls; containing *web documents*) and *links*



Link Structure:

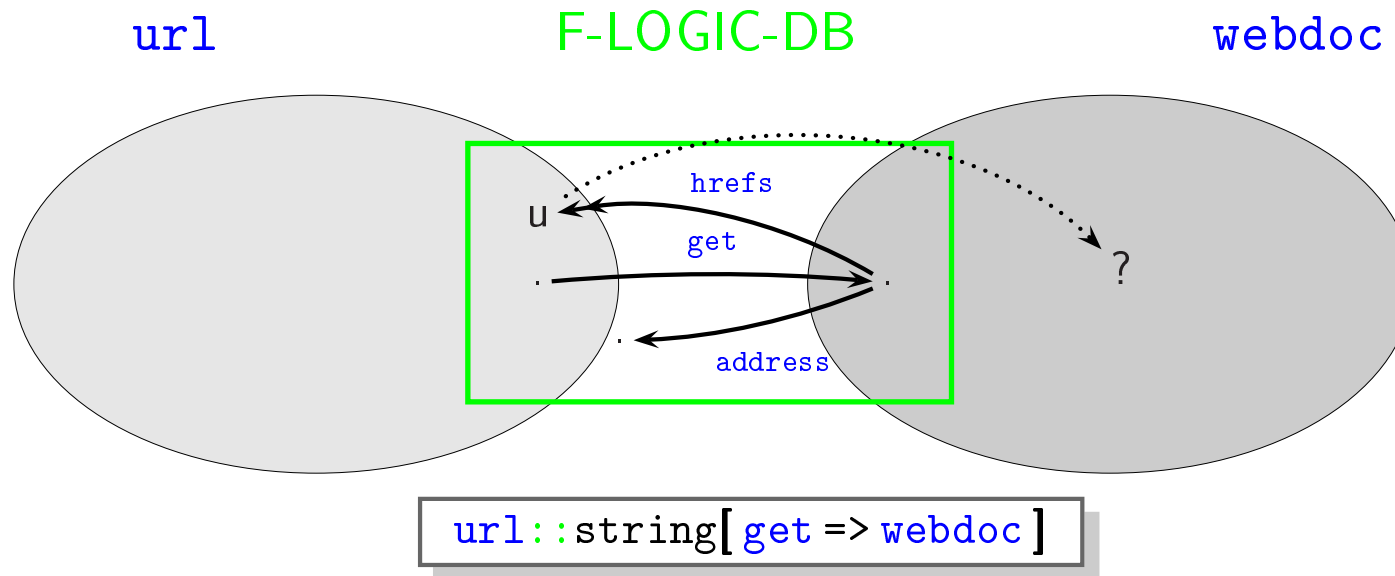
Signature: `webdoc[hrefs@(string) =>> url]`

Example: `wd1:webdoc[hrefs@("label") ->> "url2"]`

Further Attributes:

`webdoc[self => url; address => string; modif => string; ... ; error =>> string].`

Additional: user-programmed evaluation of the web documents.



Rule-Based Exploration:

`U.get[] ← U:url, ...`

% generate OID

\Rightarrow `U.get:webdoc`

% ... add to webdoc

\Rightarrow `U.get[address -> ...; hrefs@(...) ->> ...]`

% ... fill in slots

`U:explored ← U:url.get[].`

`NewU:url ← U:url[hrefs@(_) ->> NewU].`

- **Path Expressions** [FLU-VLDB-94]

closure axioms \Rightarrow extended Herbrand universe \overline{U} , Herbrand base \overline{HB}

- **Web Interface**

- set of *reserved names* R (`get`, `url`, `hrefs` ...)

- $\text{explore}: URL \rightarrow \wp(\overline{HB}(URL \cup R))$

% maps URLs to sets of new facts

- **Web Access Axiom:** for $H \subset \overline{HB}$:

$$H \models u : \text{url} \wedge u.\text{get} \Rightarrow H \models \varphi_{new} \text{ for all facts } \varphi_{new} \in \text{explore}(u)$$

“if `get` is defined for a URL u , then all explored data is in H ”

\Rightarrow **minimal Herbrand Web Model**

- **Integration with Bottom-up Evaluation:**

$$T_{\overline{P}}^{\mathcal{W}}(H) := H \cup T_{\overline{P}}(H) \cup \bigcup_{u: \text{url}, u.\text{get} \in T_{\overline{P}}(H)} \text{explore}(u)$$

\Rightarrow **declarative semantics:** if $\text{explore} := \emptyset$ then *Web-FLORID* = *FLORID*

CIA WORLD FACTBOOK (CIA)

- geography, people, government, economy, ... **no cities** (apart from country capitals)
- information: link structure, **formatted text**
- flat (text) structure, quite regular, only , <I>,
-tags used for structuring

WORLD ONLINE (WOL)

- administrative divisions, **main cities**
 - information: link structure, **tables**
 - structured (tables), but not regular (different table layout, columns)
-

EXAMPLE: INTEGRATION CIA WORLD FACTBOOK and WORLD ONLINE

Netscape: The World Factbook page

total population: 76.41 years
 male: 73.78 years
 female: 79.17 years (1996 est.)
Total fertility rate: 1.82 children born/woman (1996 est.)
Nationality:
noun: Briton(s), British (collective plural)
adjective: British
Ethnic divisions: English 81.5%, Scottish 9.6%, Irish 2.4%, Welsh Pakistani, and other 2.8%
Religions: Anglican 27 million, Roman Catholic 9 million, Muslim 760,000, Sikh 400,000, Hindu 350,000, Jewish 300,000 (1991 est.)
note: the UK does not include a question on religion in its census
Languages: English, Welsh (about 26% of the population of Wales and Scotland)

Netscape: The World Factbook Regional I

K...
 ▪ [Romania \(30 KB\)](#)

S...
 ▪ [San Marino \(25 KB\)](#)
 ▪ [Serbia and Montenegro \(31 KB\)](#)
 ▪ [Slovakia \(28 KB\)](#)
 ▪ [Slovenia \(30 KB\)](#)
 ▪ [Spain \(32 KB\)](#)
 ▪ [Svalbard \(21 KB\)](#)
 ▪ [Sweden \(30 KB\)](#)
 ▪ [Switzerland \(29 KB\)](#)

Netscape: GIF image 351x752 pixels

emacs: cia-wol.flp

```
File Edit Apps Options Buffers Tools Recent Files Help
%%
%% RULE-BASED OBJECT FUSION (COUNTRIES)
%%
%% fuse two countries if they have the SAME CONTINENT AND NAME
C1 = C2 :-
  C1:country[continent->CT;name@{S1}->N],
  C2:country[continent->CT;name@{S2}->N],
  not S1=S2, not C1=C2.

%% ... or the CIA CAPITAL is a WOL MAIN CITY (and same continent)
C1 = C2 :-
  C1:country[continent->CT]..main_cities[name@{wol}->N]
  C2:country[continent->CT;capital->N; name@{cia}->_],
  not C1=C2.

?- sys.echo@("").
?- sys.echo@("*** FUSING CIA and WOL COUNTRIES ***").
?- sys.strat.doIt.

***
*** XEmacs: cia-wol.flp (Flp RCS:1.3 Font) L289 80%
*** EQUATING: cid(wol,"Czech Rep.") = cid(cia,"Czech Republic")
*** END EVALUATION

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

```
1 output(s) printed

Answer to query : ?- _:country[name@{ } -> C; capital -> N],
N; population -> P].
N/"Vienna" C/"Austria" P/"1,583,000"
N/"Prague" C/"Czech Republic" P/"1,215,000"
N/"Prague" C/"Czech Rep." P/"1,215,000"
***XEmacs: *Flp* (Inferior-Flp: run) L341 95%
```

Netscape: Global Statistics - Home

File Edit View Go Bookmarks Options Directory Window Help

Global Stats | World | Charts | Africa | America | Asia | Europe | Oceania | Home

Europe

- Germany
 - Germ I
 - Germ II
- Greece
- Hungary
- Italy
- Netherlands
- Poland
- Portugal
- Romania
- Russia
- Spain
- Sweden
- Switzerland
- United Kingdom
- Ukraine

administrative divisions
 main cities

main cities of the United Kingdom

cities	population 1994 est.
London	6,967,500
Birmingham	1,008,400
Leeds	724,400
Sheffield	530,100
Bradford	481,700
Liverpool	474,000
Manchester	431,100
Bristol	399,200
Kirklees	386,900
Wirral	331,100

© 1997, Profiler

CIA Factbook: [Matching](#) via [Regular Expressions](#):

accessing relevant pages:

```
C[url@(cia)->U] :- C:continent[file@(cia)->FN], strcat(cia.src,FN,U).
```

```
U:url.get :- C:continent[url@(cia)->U].
```

```
cid(C):country[url@(cia) -> U; name@(cia)-> Label; continent -> CT ] :-  
    CT:continent.url@(cia).get[hrefs@(Label) ->> U].
```

```
U:url.get :- _:country[url@(cia)->U].
```

extracting “raw data”:

```
pattern(capital_name, "/Capital:.*\n(.*)/").
```

```
pattern(total_area, "/total area:.*\n(.*) sq km/").
```

```
C[Method -> X] :- pattern(Method, RegEx),  
    pmatch(C:country.url@(cia).get, RegEx, "$1", X).
```

restructuring and data cleaning:

```
C:real_country :- C:country[capital_name->CA], not substr("none", CA).
```

+ [Patterns and rules for commalists \(ethnic groups, languages\)](#)

WOL Pages: [Parsing](#) (nsgmls-Parser integrated into *FLORID*) and [Evaluating](#):

[Accessing + parsing relevant pages](#):

```
U:url.parse :- C:country[url@(wol)->U].      %% Generates parsetree of the document
```

```
?- Tab:(U:url.parse.table)
```

```
element(Tab,Row,Col) [contents->Cont;type->Type] :-
```

```
  Tab:(U.parse.table),  Tab.table@(0) [tbody@(Row)->_[tr@(Col)->X[Type@(0)->Cont]]].
```

[Identifying Main-Cities-Table and column attributes](#)

```
C[main_city_tab -> T[header_row->HZ;pop_year@(PS)->Y;city_col->>CS;pop_col->>PS]] :-
```

```
  C:country[url@(wol)->U], T:(U.parse.table),
```

```
  element(T,_,_) [contents->Cont], substr(Cont,"main cities"),
```

```
  element(T,HZ,CS) [contents->Header1;type->th], substr("city",Header1),
```

```
  element(T,HZ,PS) [contents->Header2;type->th], substr("pop",Header2),
```

```
  pmatch(Header2,"/19([0-9][0-9])/","$1",Y).
```

[Evaluation of main-cities-table](#):

```
C[main_cities ->> cty(C,CN):city[country->C;namestr->N;population@(Y)->P]] :-
```

```
  C:country[main_city_tab -> T[city_col->>CS;pop_col->>PS;pop_year@(PS)->Y]],
```

```
  element(T,DZ,CS) [contents->CN;type->td],
```

```
  element(T,DZ,PS) [contents->P;type->td].
```

QUERY: *"Name the capitals (from CIA) with their population (from WOL)"*

```
?- _:country[name@(cia) -> Country; capital_name -> City],
    _:city[name@(wol) -> City; population -> P].
```

```
P/"1,583,000"    City/"Vienna"    Country/"Austria"
P/"1,215,000"    City/"Prague"    Country/"Czech Republic"
P/"2,152,423"    City/"Paris"    Country/"France"
P/"3,472,009"    City/"Berlin"    Country/"Germany"
```

Fusing Country Objects:

```
C1 = C2 :- C1:country[name@(cia)->N], C2:country[name@(wol)->N].
```

```
C1 = C2 :- C1:country[continent->CT]..main_cities[name@(wol)->N],
           C2:country[continent->CT;capital_name->N; name@(cia)->_].
```

Linking Capitals to Countries:

```
C:country[capital->Cap[name@(cia)->CN]] :-
    C:cia_rel_country[capital@(cia)->CN; main_cities->>Cap[name@(wol)->>CN]].
```

```
?- _:country[name@(cia) -> Country].capital[name->City;population -> P].
```

... same answer as above.

Matching: Link structure known, document structure fixed and known.

Parsing+Evaluation: Link structure known, varying document structure.

⇒ *content-based* queries (data extraction).

Fishing in the Web: Link structure not known. Must be extracted.

Def. A **semistructured database** is a finite set of labeled edges:

$$(x, \ell, y) \in D \quad \Leftrightarrow \quad x \xrightarrow{\ell} y \quad \Leftrightarrow \quad x[\ell \rightarrow \{y\}].$$

Mapping a ssdb to F-logic:

$$X:\text{node}, Y:\text{node}, L:\text{label}, X[L \rightarrow \{Y\}] \leftarrow \text{ssdb}(X, L, Y).$$

Example: Web Skeleton Extractor

P_{ext}:

```

root[src ->> {u1, ..., un}].                                % define root nodes
node:: url.                                                  % nodes are urls
(U:node).get ← root[src ->> {U}].                            % get root nodes
-----
Y:node, L:label, X[L ->> {Y}] ←                               % define new nodes/labels/links...
    X:node.get[hrefs@(L) ->> {Y}], φ.                        % ...by following hrefs
Y.get ← Y:node, ψ.                                          % access nodes which satisfy ψ

```

Specialization of the Skeleton Extractor for DBLP

- | | |
|--|--|
| $\text{root}[\text{src} \rightarrow \{\text{dblp}\}].$ | $\text{dblp} = \text{"http://www.informatik.uni-trier.de/~ley/db/"}$ |
|--|--|
- $\varphi = \text{substr}(\text{"trier"}, Y)$, and *(consider only url's containing "trier")*
- $\psi = \text{substr}(\text{" /db/journals/is/"}, Y)$, *(restrict to IS journal)*

⇒ Queries with **path expressions**: `?- dblp.."Inf. Systems"..L.."Michael E. Senko".`

Def. General path expressions *GPE*:

- $\mathcal{L} \cup \{\text{any}\} \subseteq GPE$,
- if $M, N \in GPE$ and $n \in \mathbb{N}_0$, then the following are in *GPE*:
 $(M \cdot N)$, $(M | N)$, $(M)^*$, $(M)^+$, $(M)^?$, $(M)^{-1}$, $(M)^n$,
- if φ is binary relation symbol, then $\text{if}(\varphi) \in GPE$,
- if $\ell \in \mathcal{L}$ and ψ is a unary relation symbol then $\mu(\ell)$, $\mu(\ell, \psi) \in GPE$.

⇒ specification/implementation by simple path expressions + rules

Summary

- DOOD paradigm attractive for querying and restructuring the Web
- uniform access to local db & Web data \Rightarrow integration of heterogenous information
- seamless integration of an SGML parser
- reasoning about document structure and Web structure
- use of search engines (AltaVista)



- Implementation in *Web-FLORID* (= Florid 2.0):
<http://www.informatik.uni-freiburg.de/~dbis/florid/>
-