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 ⇒ integration of heterogenous information

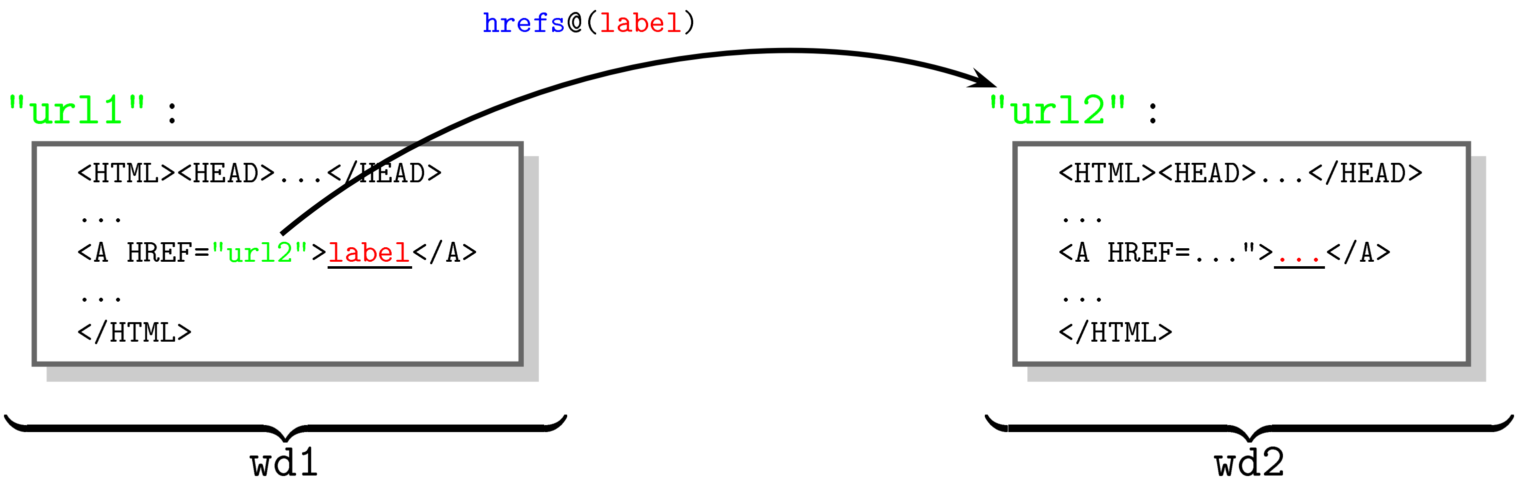
Basic Constructs:

```
Object:Class                                      % ISA-relation, "∈"
SubClass::Class                                   % SUBCLASS-relation "⊆"

Class[Method@(P-types) => R-type]                 % SIGNATURE: single-valued
Class[Method@(P-types) =>> R-types]               % ... and multi-valued

Object[Method@(Params) -> R]                      % DATA: single-valued
Object[Method@(Params) ->> {R1,R2}]               % ... 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:

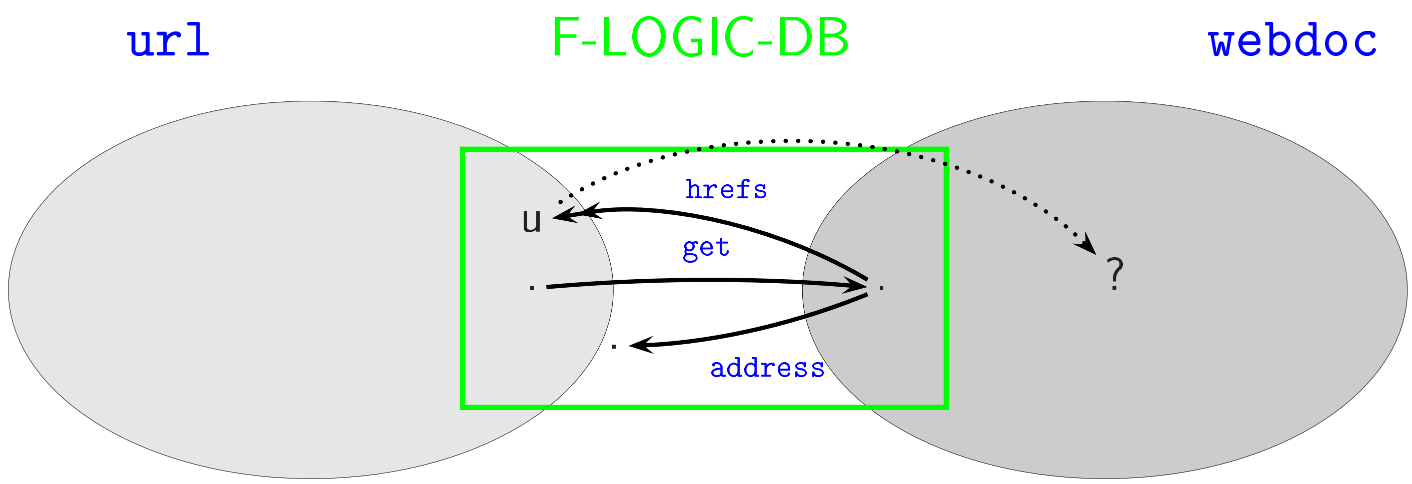
Signature: `webdoc[hrefs@(string) =>> url]`

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.



`url::string[get => webdoc]`

Rule-Based Exploration: `U.get[ ] ← U:url, ...` *% generate OID*

⇒ `U.get:webdoc` *% ... add to* `webdoc`

⇒ `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 $\overline{\mathcal{HB}}$
- **Web Interface**
  - set of *reserved names* R (`get`, `url`, `hrefs` ...)
  - *explore*: $\mathcal{URL} \rightarrow \mathfrak{P}(\overline{\mathcal{HB}(\mathcal{URL} \cup R)})$ *% maps URLs to sets of new facts*
- **Web Access Axiom**: for $H \subset \overline{\mathcal{HB}}$:

  $H \models u : \texttt{url} \wedge u.\texttt{get} \quad \Rightarrow \quad H \models \varphi_{new}$ for all facts $\varphi_{new} \in$ *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**:

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

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

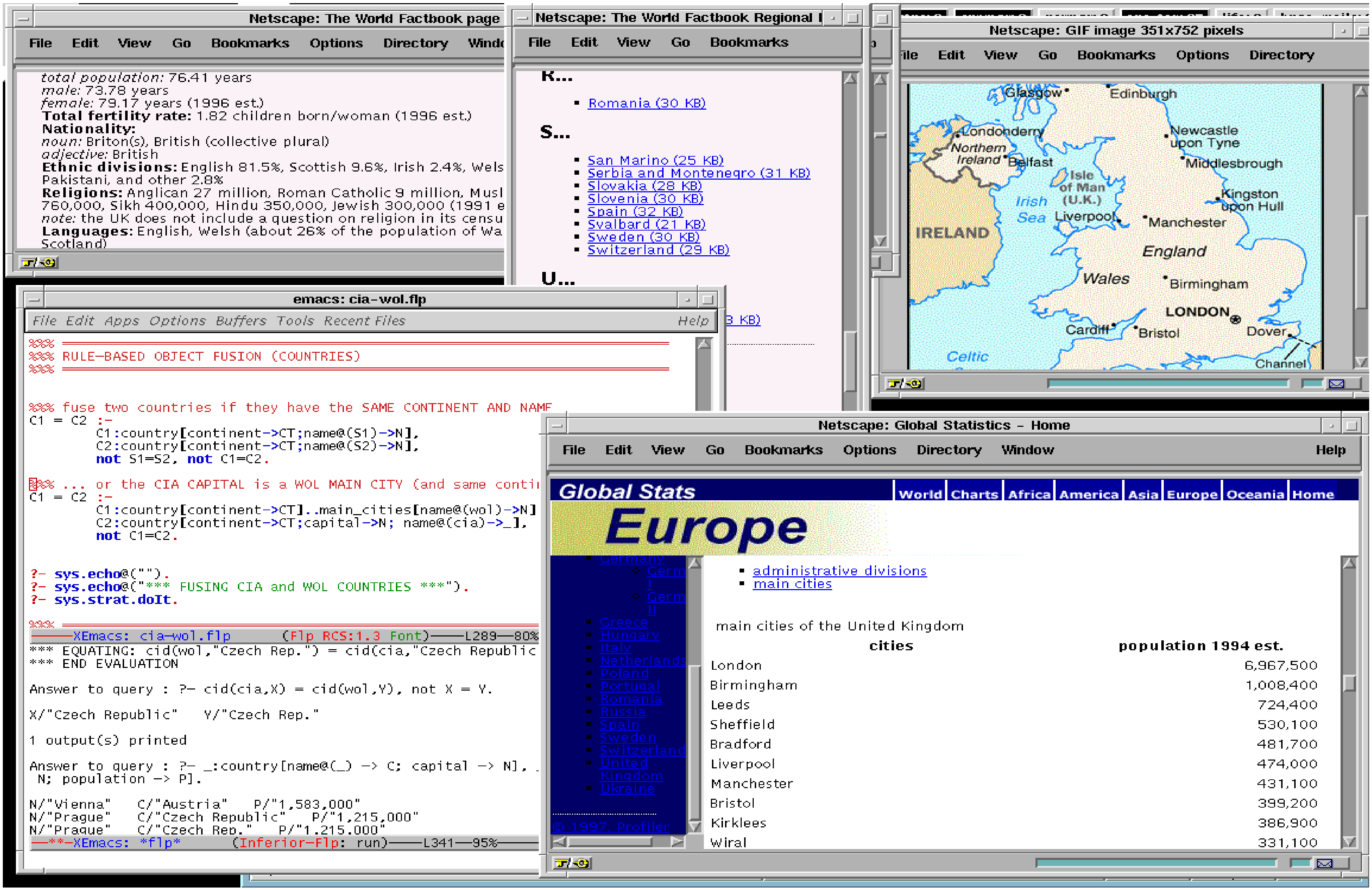
CIA WORLD FACTBOOK (CIA)

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



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 (ethic 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.

$\Rightarrow$ *content-based* queries (data extraction).

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

<u>**Def.**</u> A **semistructured database** is a finite set of labeled edges:

$$(x, \ell, y) \in D \quad \Leftrightarrow \quad x \overset{\ell}{\texttt{->>}} y \quad \Leftrightarrow \quad x[\ell \texttt{ ->> } \{y\}] \; .$$

Mapping a ssdb to F-logic:

```
X:node, Y:node, L:label, X[L ->> {Y}]  ←  ssdb(X,L,Y).
```

Example: Web Skeleton Extractor

P_{ext}:

```
root[src ->> {u_1, ..., u_n}].                         % 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/lables/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

- root[src ->> {dblp}]. dblp = "http://www.informatik.uni-trier.de/~ley/db/".
- φ = substr("trier",Y), and (*consider only* `url`*'s containing "trier"*)
- ψ = substr("/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 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/`