Information Extraction from the Web with FLORID

Wolfgang May
Institut für Informatik
Universität Freiburg
Germany

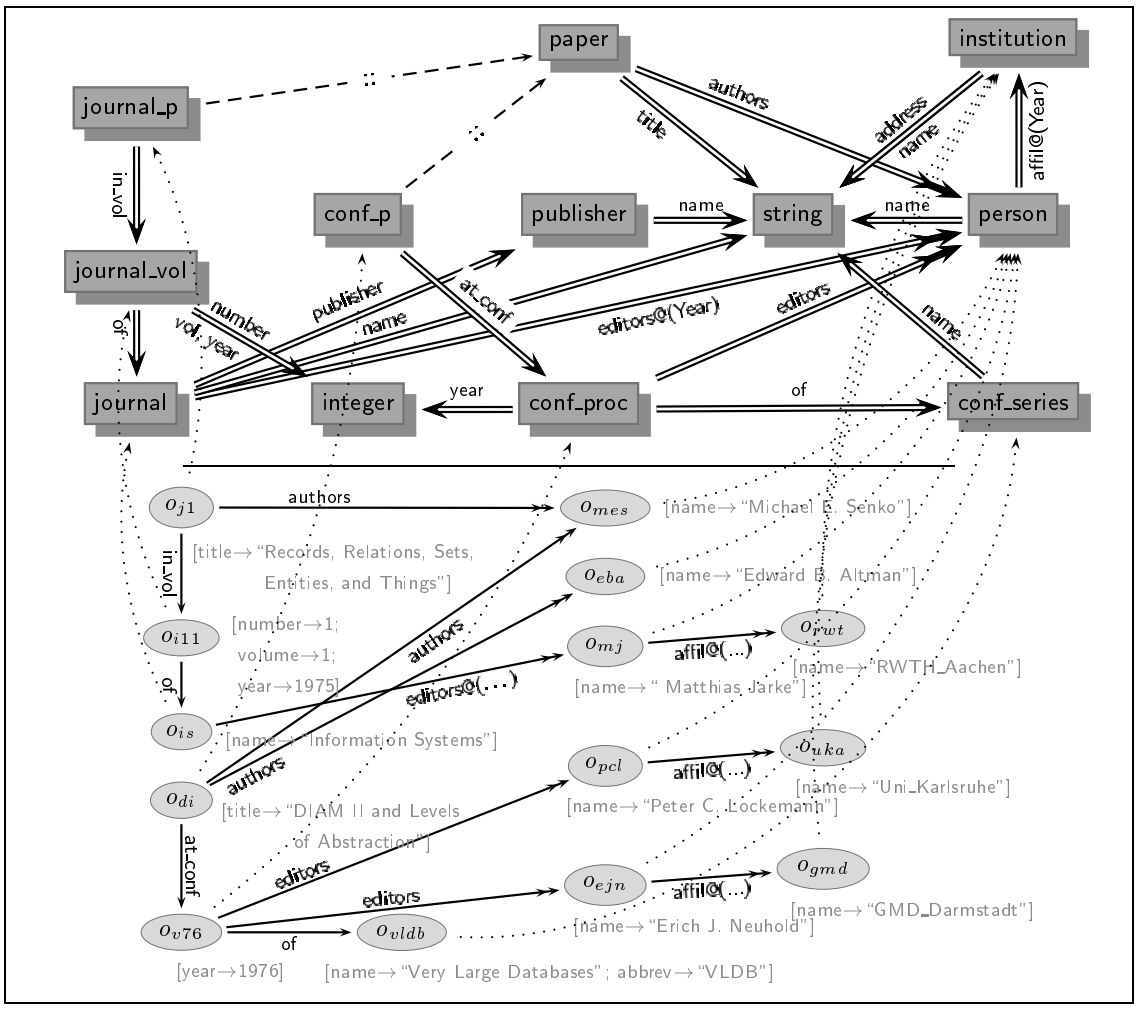
Florid

- Implementation of F-Logic (1992-1996)
- C++, DFG-Project
- Version 2.0 (1997/1998): Web Access
- Examples / Case Studies:
  - the DBLP Publications Web Server (single-site)
  - Geographical Information (multi-site)

F-Logic in a Nutshell

- *id-terms* are composed from object constructors and variables as usual.
- is-a atoms: o : c
- subclass atoms: c :: d
- Method applications to objects:
  o[m→v] (scalar)
  o[m↠v] (multivalued)
  analogous with arguments: o[m@(x_1,...,x_n)→v].
  inheritable:
  c[m•→v]
  c[m•↠v]
- Signatures of methods:
  c[m⇒v] (scalar)
  c[m⇒↠v] (multivalued)
- Variables allowed at all positions
- Entities can act as classes, objects and methods
- Rules over atoms: <head> :- <body>.
- Program: a set of rules

Example: F-Logic Model of DBLP



journal_p :paper. conf_p :paper.

paper[title⇒string; authors⇒⇒person].

journal_p[in_vol⇒journal_volume].

conf_p[at_conf⇒conf_proc].

o_{j1} :journal_p[title→"Records, Relations, Sets, Entities, and Things"; authors→→{o_{mes}}; in_vol→o_{i11}].

o_{di} :conf_p[title→"DIAM II and Levels of Abstraction"; authors→→{o_{mes}, o_{eba}}; at_conf→o_{v76}].

o_{i11} :journal_vol[of→o_{is}; number→1; volume→1; year→1975].

o_{is} :journal[name→"Information Systems"; editors@(...)→→{o_{mj}}].

o_{v76} :conf_proc[of→o_{vldb}; year→1976; editors→→{o_{pcl}, o_{ejn}}].

o_{vldb} :conf_series[name→"Very Large Databases"].

o_{mes} :person[name→"Michael E. Senko"]. o_{mj} :person[name→"Matthias Jarke"; affil@(...)→o_{rwt}].

o_{rwt} :institution[name→"RWTH_Aachen"]. ...

F-Logic in a Nutshell

- path expressions:
  $[\![o.m]\!]$: that o' s.t. o[m $\rightarrow$o']
  $[\![o..m]\!]$: all o' s.t. o[m $\twoheadrightarrow$o']

  ?- P:conf_proc.of[abbrev$\rightarrow$"VLDB"], P[year$\rightarrow$1976],
  P..editors[affil@(1976)$\rightarrow$A].

- object creation by path expressions in the head:
  o.m[...] $\leftarrow$ ...

- nonmonotonic inheritance, equivalent to Default Logic:

$$\frac{O:C\ ,\ C[M\bullet\!\!\rightarrow V]\ :\ \neg\exists C'(O:C'\wedge C'::C)\ ,\ O[M\rightarrow V]}{O[M\rightarrow V]}$$

- Derived equality via object fusion:
  $o_1 = o_2 \leftarrow$ ...
  implemented in the object manager.

- Aggregates: sum, count

  var = <aggr>{var$_0$ [var$_1$,var$_2$,...,var$_n$];
  <expr(var$_0$,...,var$_n$)>}

  S = count{P [A]; A:author[has_written$\twoheadrightarrow$P]}.

- FLORID: bottom-up inflationary semantics with user-defined stratification

I: The Web Model

- unified object-oriented model in F-Logic
  - the Web (carrier of information)
  - the application domain (carried information)
- graph-based model
- topology of the Web ((Web) skeleton):
  - nodes: Web documents,
  - (labeled) edges: hyperlinks between documents.
  - skeleton: no information apart from the link structure is available;
- The *page markup* (tags):
  induces a tree structure of the page contents.
- an object-oriented model of the application domain.

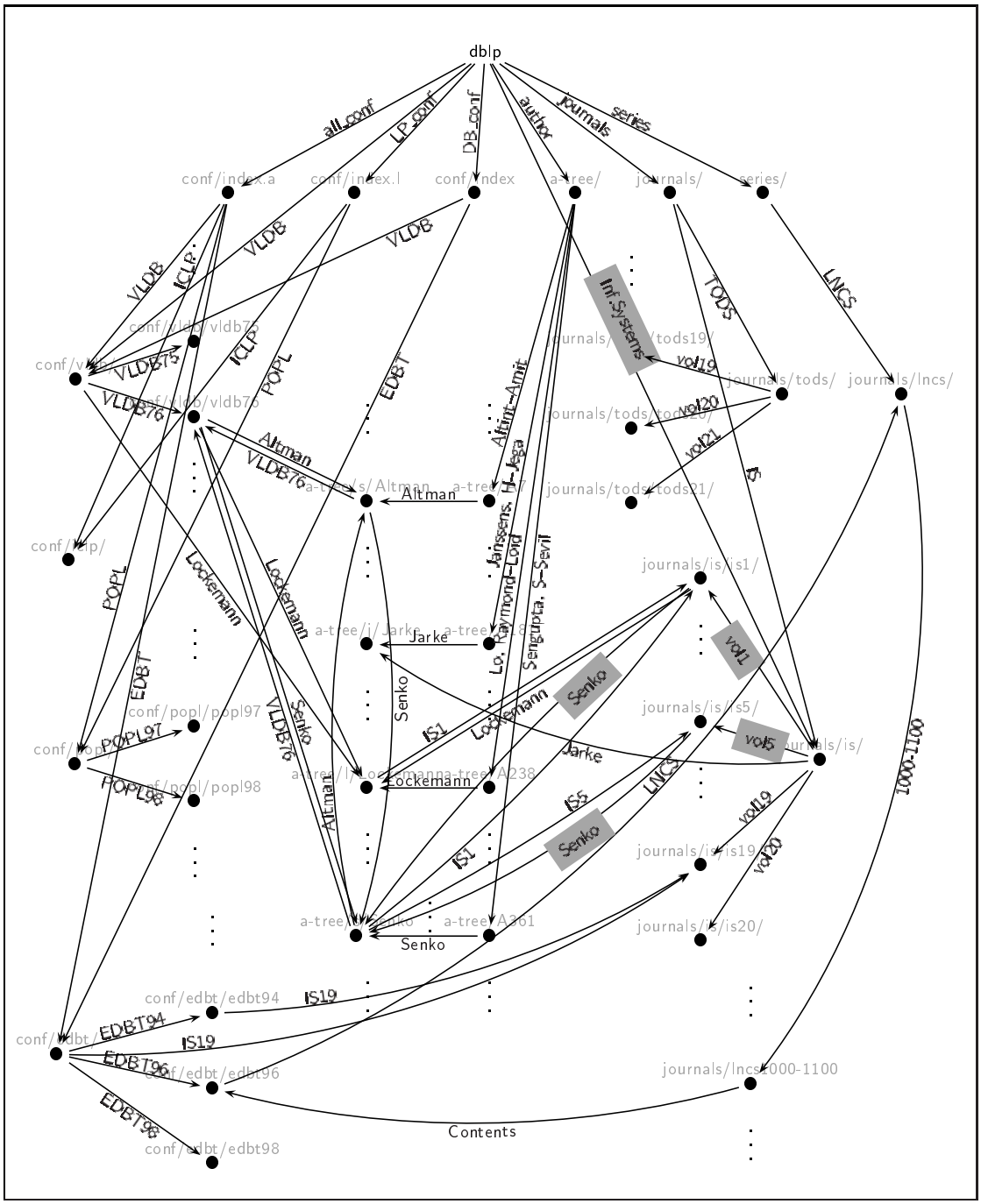
The Skeleton $\mathcal{SK}$: URL's and Web Documents

- Every resource in the Web has a unique *url*.
- document associated with a url contains *hyperlinks* to other url's

$$(x, \ell, y) \in \mathcal{SK} \qquad \Leftrightarrow \qquad x[\mathsf{hrefs@}(\ell) \twoheadrightarrow \{y\}] \ .$$

Example: The DBLP Server

Example: The DBLP Server

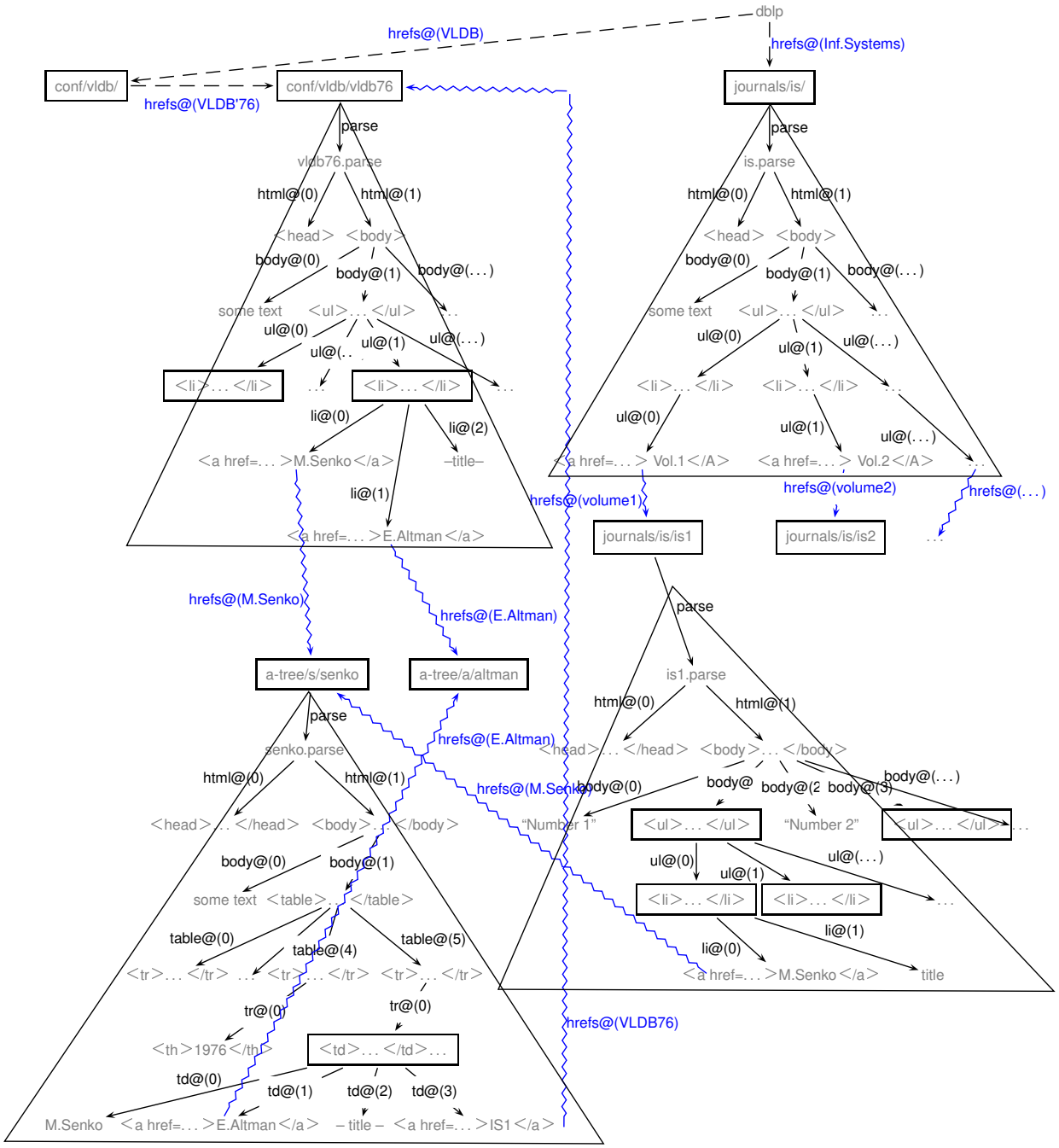


- skeleton corresponds to real world objects
  journals, conferences, books, and authors

Extending the Web Skeleton: Parse-trees

- real-world objects are represented as individual Web pages, or by substructures.
- integration of *parse-trees*

Example: Extended Web Skeleton of DBLP



Extended Web Skeleton

- *extended Web skeleton*: unified – but still based on the Web representation, not on the application semantics.
- many objects have already a direct counterpart in the extended Web skeleton.
  - objects have a Web representation as a Web page referencable via url.
  - objects correspond to nodes in a parse-tree (journal volumes and papers).
    referencable in HTML via page#anchor.
  - counterparts in several parsetrees
    $\Rightarrow$ Object fusion:
    Objects as objects in the Web representation *and* in the application model.
  - mapping between nodes/arcs of the extended Web skeleton and instances of distinguished classes/relationships of the application modeling.
- XML?
  Parse-tree $\equiv$ application-semantic model?

Requirements for Implementation

- non-logical features $\Rightarrow$ built-ins:
- Web Access via http-protocol,
- Parsing of HTML/SGML/XML,
- Matching with Perl Regular Expressions,
- Logical issues:
- Suitable modeling (classes)
- Object creation on demand
- Object fusion
- Navigation in the model
- Powerful, flexible reasoning

II: How is this done in FLORID

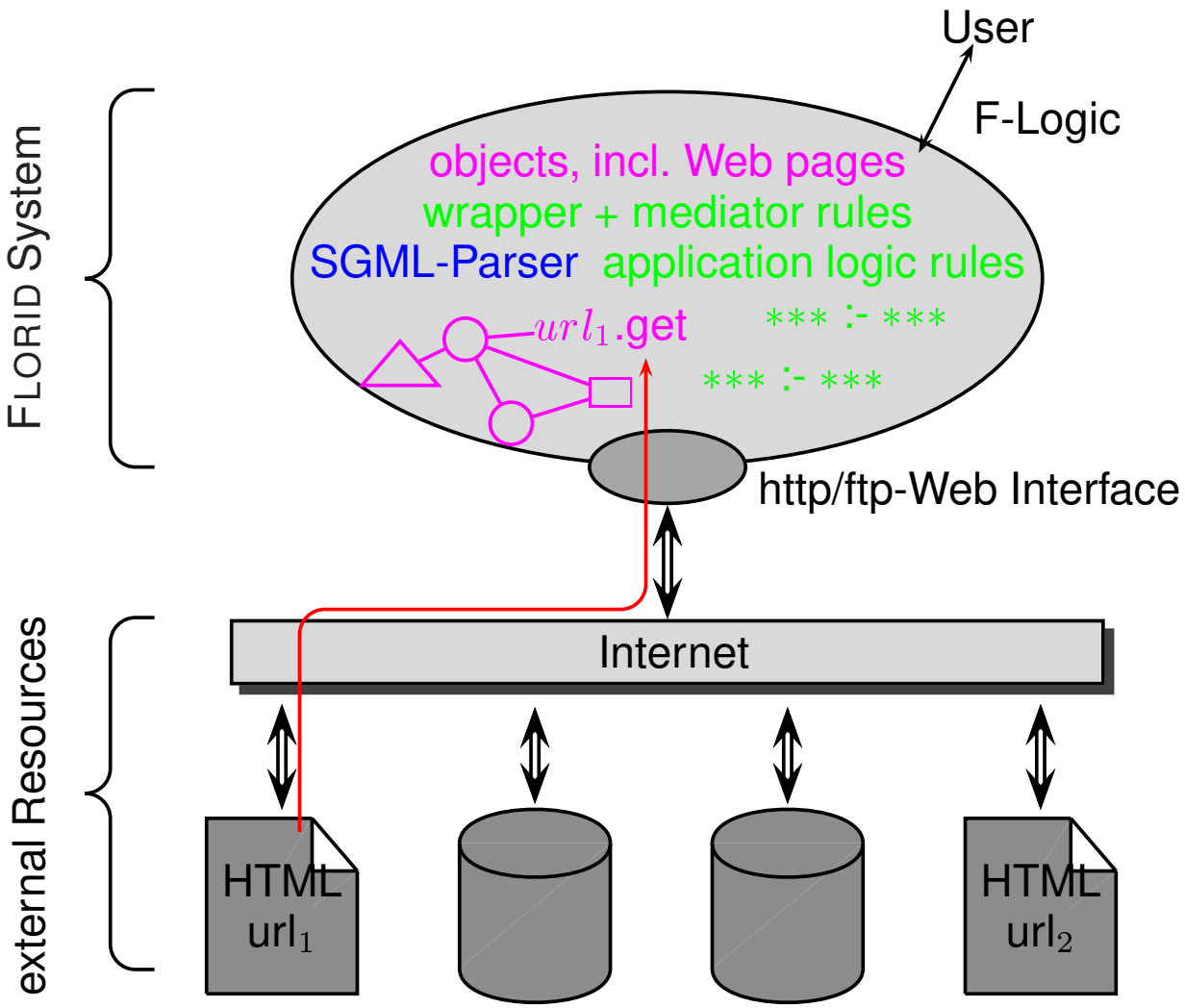
Common Architecture

Two-Level Wrapper & Mediator – Architecture

- Warehouse Approach:
  - all relevant data sources/Web pages are fetched and evaluated (by wrappers)
  - data integration by mediators
  - queries are stated against the “warehouse”
  - e.g. STRUDEL
- Virtual Approach:
  - Web pages are accessed on-demand
  - mediator redirects user-queries to wrappers
  - e.g. TSIMMIS

- different languages and data models for wrapping/mediating/querying

Architecture of Web Access with FLORID



- Unified, monolithic framework for wrappers and mediators
- F-Logic: unified data model, wrapper, mediator, and querying language
- hybrid approach combining virtual and warehouse

Exploration of the Web

- classes url and webdoc.
- class url: $url.\mathsf{get}$ implemented as an *active* method (C++):

  $u.\mathsf{get}$:- ...
  - accesses the Web document which is accessible via u
  - assigns it to $u.\mathsf{get}$ (object creation)
  - becomes an instance of class webdoc
  - and several properties are automatically filled in.

$u.\mathsf{get}[\mathsf{hrefs@}(\ell) \twoheadrightarrow u'] \Leftrightarrow$
$u.\mathsf{get}$ contains "<a href $= u'$ > ℓ ".

```
url::string[get⇒webdoc].
webdoc[url ⇒url; author ⇒string;
       type ⇒string; hrefs@(string) ⇒⇒url; ... ;
       modif ⇒string; error ⇒⇒string].
---------------------------------------------------
url1[get→wd1].      url2[get→wd2]
wd1:webdoc[url→url1; hrefs@("label")→→{url2}]
wd2:webdoc[url→url2; type→html; ...]
```

Formal Semantics of Web Access

Herbrand semantics of get and parse:

$$\text{explore}: \mathcal{URL} \rightarrow 2^{\mathcal{HB}}$$
$$\text{parse}: \mathcal{URL} \rightarrow 2^{\mathcal{HB}}$$

A Herbrand model $\mathcal{H}$ of an F-Logic program P is a model of P wrt. Web-Access (built-in semantics of $u.\text{get}$ and $u.\text{parse}$) if

- if $\mathcal{H} \models u : \text{url} \wedge u.\text{get}\}$, then $\text{explore}(u) \subseteq \mathcal{H}$
- if $\mathcal{H} \models u : \text{url} \wedge u.\text{parse}\}$, then $\text{parse}(u) \subseteq \mathcal{H}$

... integrated into the T_P-operator:

For an F-Logic program P and an H-interpretation $\mathcal{H}$,

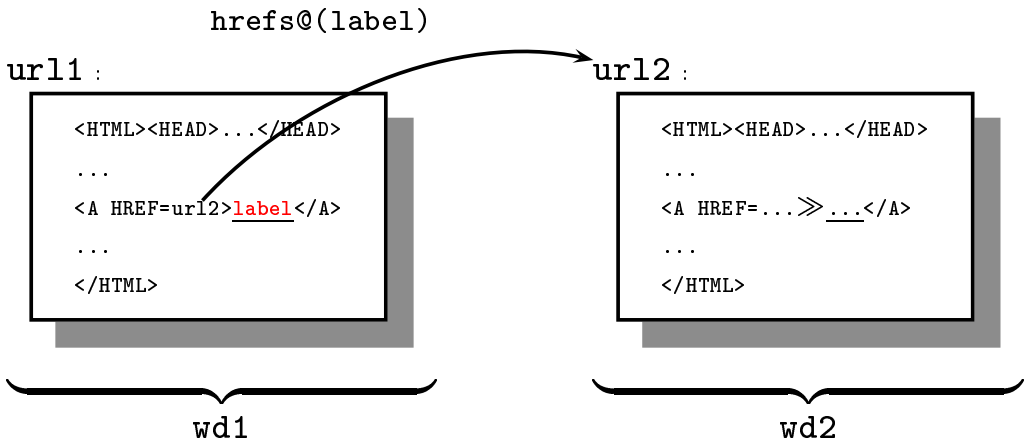
$$\begin{aligned}
T_P(\mathcal{H}) &:= \mathcal{H} \cup \{h \mid (h \leftarrow body) \in ground(P), \mathcal{H} \models body\} , \\
T_P^{\mathcal{W},0}(\mathcal{H}) &:= \mathcal{H} , \\
T_P^{\mathcal{W},i+1}(\mathcal{H}) &:= \mathcal{Cl}(T_P(T_P^{\mathcal{W},i}(\mathcal{H})) \\
&\quad \cup \bigcup\{\text{explore}(u) \mid T_P(T_P^{\mathcal{W},i}(\mathcal{H})) \models u : \text{url} \wedge u.\text{get}\} \\
&\quad \cup \bigcup\{\text{parse}(u) \mid T_P(T_P^{\mathcal{W},i}(\mathcal{H})) \models u : \text{url} \wedge u.\text{parse}\})
\end{aligned}$$

Then use $T_P^{\mathcal{W},\omega}$.

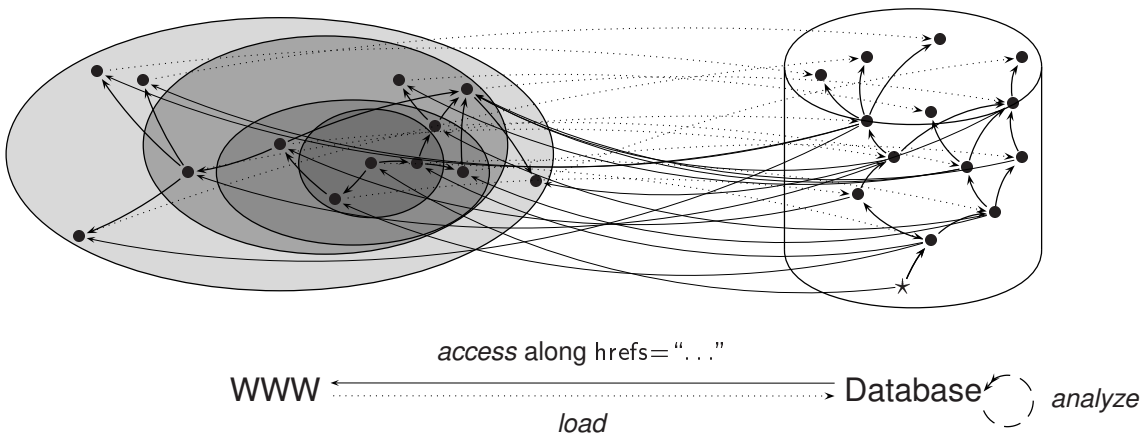
Data-Driven Web Exploration

- in course of the information extraction and restructuring process, additional pages are recognized to be relevant:

  U.get ← A:author[homepage→U].



- approach implements a hybrid concept by embedding *data-driven wrapping* into a *warehouse* approach



Wrapping

- url.get: *raw*, uninterpreted data
- wrapping by F-Logic Rules

- Logical Markup:
  Parser-based
  DBLP-server: sufficiently well-structured HTML
  - direct correspondence between HTML-nodes and objects (extended Web skeleton).
- Optical and Syntactical Markup:
  pattern matching via regular expressions
  - construction of object model from scratch/identifying new objects

Parsing of Web Pages

- url.parse generates F-Logic representation of the parse-tree,
  assigns it to the object u.parse : parsetree
  - *SGML-tagged* groups $<tag>$ ... $</tag>$ become objects,
  - classes $webdoc.<tag>$,
  - navigation: $o.<tag>$@(1), ..., $o.<tag>$@(n)
    are the segments inside $o.<tag>$
- attributes: $o.<attr>$

- tables whose header contains '1998' in any header row/column are identified by

  ?- T : wd.table,
  T.table@(Row).tr@(Col)[th@(0)→S],
  substr(S,"1998").

- the contents of the third column of the 17th row of a given table tab is addressed by

  tab.table@(17).tr@(3).THD@(0).

- hyperlinks emanating from the parse-tree:

  Z[hrefs@(Label)↠Url] ←
  Z:(U:url.parse.a), Z[a@(0)→Label; href→Url].

Example: DBLP Server

constructing the application model:

dblp[url→“http://. . . ”].
dblp.url:url.
dblp.url.get.

(dblp[journals_page→(X:url)],
dblp.journals_page.get ←
    dblp.url.get[hrefs@(“Journals”)↠X].

(dblp.conf_page→(X:url)],
dblp.conf_page.get ←
    dblp.url.get[hrefs@(“Conferences)↠X].

% conferences
S : conf_series[name→S, url→(U:url)],
U.get ←
    dblp.conf_page.get[hrefs@(S)↠U].

(S.year@(Year) : conf) [series→S; year→Year; url→(U:url)],
U.get ←
    (S : conf_series).url.get[hrefs@(Year)↠U].

% ... similar for journals

Example: DBLP Server

Every paper on a conference or journal volume page is represented in an <li> tag, e.g.

author$_1$, . . . , author$_2$: title pages.

```
conf_paper :: paper.
journal_paper :: paper.
% conference papers
p(P):conf_paper[parsenode↠P] ←
    C:conf, P:C.parse.li.
% journal papers
p(P):journal_paper[parsenode↠P] ←
    (V:journal_vol), P:V.parse.li.
% papers: titles
P[title→T] ← P:paper, P.parsenode[li@(_)→T], string(T).
% papers: authors
a(N):author[name→Name; url↠(U:url)], P[authors↠a(N)] ←
    ((P:paper).parsenode.li@(_)):(_.a)[hrefs@(Name)→U],
    substr("a-tree",U).
A[homepage→(U:url)] ←
    A:author.url.get[hrefs@("Homepage")↠U].
```

Example: DBLP Server

- single-site source
- “best case”-example
- well-structured HTML/SGML
- parser-based wrapping
- Model contains Web skeleton, parse-trees and application

More than Parsing

- not all Web pages provide logical markup
- well-structured pages need further wrapping:
  - keywords,
  - commalists,
  - text search for relevant words

  $auth_1$, $auth_2$, ... , and $auth_n$: $title$. number n in Volume v of $series$, pages $p_1 - p_2$, $year$.

Pattern Matching in FLORID

Perl regular expressions by the built-in predicate

pmatch(<string>,"/<regexp>/", [<fmt-list>], [X$_1$,. . . , X$_n$])

```
pmatch(STRING,
        "/\A ([:]*): (.*)\.\s
          Number ([0-9]*) in Volume ([0-9]*) of ([a-Z]*),
          pages ([0-9-]*), ([0-9]*)/",
       [$1,$2,"$4($3)", $5, $6, $7],
       [AuthList, Title, Num, Series, Pages, Year])
```

AuthList is a commalist ...

Generic Wrapping Tasks

Extracting contents of the pages:

- Logical Markup
  - HTML-Lists
  - HTML-Tables: Headers, Columns
- Optical Markup
  - Paragraphs
  - Boldfacing, Emphasizing
- Syntactical Markup:
  - Commalists, Semicolons, Parentheses

$\Rightarrow$ Generic Rules for these tasks

\+ Rules for application-specific interpretation

\+ Refined rules dealing with exceptions

- Modular Design

Case Study: The MONDIAL Geographic Database

- CIA World Factbook: Countries
- CIA World Factbook: Organizations
- GlobalStatistics: Cities and Administrative Divisions
- ... some additional smaller sources

$\Rightarrow$ Integration

- different layouts of data sources
- not “best-case”, but “real-world”

CIA World Factbook: Countries

Country Pages: Optical Markup
Processing: By Matching

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

cid(cia,C2):country
    [url@(cia) -> U;
     name@(cia)->> CountryName;
     continent -> Cont] :-
 Cont:continent.url@(cia).get[hrefs@(Label) ->> U],
 pmatch(Label,"/(.*) \([0-9]/","$1", CountryName).
```

Create country objects:

$o_{belgiumCIA}$: country
[url@(cia)→“. . . ”;
name@(cia)↠“Belgium”;
continent→o_{europe}] .

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

Countries: scalar properties by patterns

```
p(country,capital,"/Capital:.*\n(.*)/",string).
p(country,indep,"/Independence:.*\n(.*) [^(]/",date).
p(country,total_area,"/total area:.*\n(.*?) sq km/",num).
p(country,population,"/Population:.*\n([0-9,]+) /",num).
```

Generic Rule + Refinement:

```
Ctry[Method:Type -> X] :-
  p(country,Method, RegEx, Type),
  pmatch(Ctry:country.url@(cia).get, RegEx, "$1", X),
  not substr("million",X).
Ctry[Method:Type -> X2] :-
  p(country,Method, RegEx, Type),
  pmatch(Ctry:country.url@(cia).get, RegEx, "$1", X),
  pmatch(X, "/(.*) million/", "$1", X1),
  X2 = X1 * 1000000.
```

$o_{belgiumCIA}$[capital→“Brussels”; indep→“4 October 1830”;
total_area→30510; population→10170241]

Dates:

```
C[M@(date)->D2] :- C:country[M:date->S],
   perl(fmt.date,S,D),
   pmatch(D,"/\A([0-9]{2} [0-9]{2} [0-9]{4})/",
          "$1", D2).
```

Finetuning: Perl-Interface

- String Manipulation

perl(<perl-routine-as-string>,
[<variable-list$_1$>],
[<variable-list$_2$>])

Input: <variable-list$_1$>
Output: <variable-list$_2$>

Also used for

- Output Redirection
- Prototyping of built-ins (e.g., Aggregates)

Countries: Multivalued properties

```
commalist(country, ethnicgroups,
     "/Ethnic divisions:.*\n(.*)/", "\%", string).
commalist(country, languages,
     "/Languages:.*\n(.*)/", "\%", string).
commalist(country, religions,
     "/Religions:.*\n(.*)/", "\%", string).
commalist(country, borders,
     "/border countr[a-z]*:.*\n(.*)/", " km", country).
```

```
C[M:commalist->L2] :-
  commalist(T,M,S,_,_),
  pmatch(C:country.url@(cia).get,S,"$1",L),
  perl(drop.comments,L,L2).
```

$o_{belgiumCIA}$

[borders : commalist→ “France 620 km, Germany 167 km, . . .”]

```
C[M->>X] :- C:country[M:commalist->S],
            pmatch(S,"/([^,]*)/","$1",X).
```

$o_{belgiumCIA}$

[borders↠{ “France 620 km”, “Germany 167 km”, . . . }].

Countries: Name-Value-Pairs

```
C[M@(X)->V] :-  commalist(T,M,_,S,_),
  strcat("/([^0-9]*)\s* ([0-9\.]+)",S,S2),
  strcat(S2,"/", Pattern),
  pmatch(C:T..M, Pattern, ["$1","$2"], [X,V]).
```

Refinement:

```
C[M@(X)->#100] :- commalist(T,M,_,S,_),
   C:T[M->>X], S = ‘‘\%’’, not substr(S,X),
   N = count{X [C,M,T]; C:T..M@(X)[]},
   N = 1.
```

$o_{belgiumCIA}$[borders@(“France”)→620;
borders@(“Germany”)→167; . . .].

$o_{germanyCIA}$[languages@(“German”)→100].

Countries: Data Cleaning/References

```
C[M@(C)->V] :- commalist(T,M,_, S, Class),
               C:Class[name@(cia)->>X],
               O:T[M@(X)->>V].
```

$o_{belgiumCIA}$[borders@(o_{france})→620; .
borders@($o_{germany}$)→167; ...].

CIA World Factbook: Organizations

Organizations Pages:
Logical Markup (lists) + Optical Markup (commalists)
Processing: Parsing + Matching

```
U:url.parse :- ciaAppC[url->U].
```

```
<UL> ... <LI>European Union (EU)</LI>
 <UL>
  <LI><I>address</I>-
          c/o ... , B-1049 Brussels, Belgium</LI>
  <LI><I>established</I>-
          7 February 1992</LI>
  <LI><I>members</I>-
          (15) Austria, Belgium, Denmark, ... </LI>
  <LI><I>membership applicants</I>-
          (12) Albania, Bulgaria, ... </LI>
 </UL> ... </UL>
```

A list for every letter:

```
L:org_list :- L:(ciaAppC.parse.ul),
              L = ciaAppC.parse.root.html@(X).
```

Organizations Lists

$2k$-th element: organization header,
$2k + 1$-th element: corresponding data list.

```
org(Short):org
  [abbrev->Short; name->Long; data->DataList] :-
    L:org_list[ul@(K)->OrgHdr],
    L.ul@(K1)[li@(0)->DataList], K1 = K + 1,
    OrgHdr[li@(0)->Name], string(Name),
    DataList.ul@(0)[],
    pmatch(Name,"/(.*) \(([^)]*)\)/",
           ["$1","$2"],[Long,Short]).
```

o_{eu}:org[abbrev→"EU"; name→"European Union";
data→"<UL><LI><I>address</I> . . . </LI>
<LI><I>established</I> . . . </LI>
<LI><I>membership applicants</I>
- (12) Albania,. . . </LI>
</UL>"] .

Analyzing Organizations Lists

Patterns for identifying list elements:

```
p(org,address,"address",string).
p(org,establ,"established",date).
p(org,member_names,"member",membertype).
p(org,applicant_names,
      "membership applicants",membertype).
membertype::commalist.
```

```
O[MethName:Type->Value2] :-
    O:org[data->D],
    p(org,MethName,MethString,Type),
    D.ul@(X)[li@(0)->
             _[i@(0)->MethString];li@(1)->Value],
    pmatch(Value,"/\A-\s*(\([0-9]*\))*\s*(.*)/",
       "$2",Value2).
```

o_{eu}[address:string→“c/o . . . Brussels, Belgium”;
establ:date→“7 February 1992”;
member_names:membertype→“Austria, Belgium, . . .”;
applicant_names:membertype→“Albania, Bulgaria . . .”]

Organizations: Application-Specific Rules

Dates (cf. slide 34):

```
C[M@(date)->D2] :- C:org[M:date->S],
   perl(fmt.date,S,D),
   pmatch(D,"/\A([0-9]{2} [0-9]{2} [0-9]{4})/",
          "$1", D2).
```

Addresses:

```
O[seatcity->City;seatcountry->Country] :-
    O:org[address->A],
    pmatch(A, "/(.*)[,0-9] ([^,0-9]*)\Z/",
           ["$1","$2"], [City,Country]).
```

o_{eu}[seatcity→"Brussels"; seatcountry→"Belgium";
estabI@(date)→"07 02 1992"].

Organizations: Memberlists

generic rule pattern for commalists (cf. slide 35)

```
O[M->>X] :- O:org[M:commalist->S],
            pmatch(S,"/([^,]*)/","$1",X).
```

o_{eu}[member_names:membertype$\twoheadrightarrow${“Austria”, “Belgium”, . . . };
applicant_names:membertype$\twoheadrightarrow${“Albania”, “Bulgaria”, . . . }] .

Organizations: Data Cleaning/References

```
O[member@(Type)->>C] :- p(org,T,Type,membertype),
                        O[T:membertype->>CN],
                        C:country[name@(cia)->>CN].
```

o_{eu}[member@(“member”)$\twoheadrightarrow\{o_{austriaCIA}, o_{belgiumCIA}, \ldots\}$].

Global Statistics: Cities and Provinces

Country Pages: Logical Markup (Tables)
Layout specialties: different column semantics
Semantical specialties: multilevel administrative divisions
Processing: Parsing + Matching Keywords

```
U:country_url.parse :- _:country[url@(gs)->U].
```

Mapping Tables to a matrix (+ Refinements):

```
element(T,Row,Column)[contents->Contents;type->Type] :-
    U:country_url, T:(U.parse.table),
    T.table@(0)[tbody@(Row)->
                _[tr@(Column)->X[Type@(0)->Contents]]].
```

$o_{belgiumGS}$.url@(gs).parse.body@(2) :
$o_{belgiumGS}$.url@(gs).parse.table.

element($o_{belgiumGS}$.url@(gs).parse.body@(2),2,3)
[contents→"area (km^2)";type→"th"].

element($o_{belgiumGS}$.url@(gs).parse.body@(2),4,3)
[contents→"3,358";type→"td"].

Global Statistics: Column Headers

Identifying the header row(s)

```
T[header_row->>HR] :- U:country_url,
   T:(U.parse.table), element(T,HR,_)[type->th].
```

$o_{belgiumGS}$.url@(gs).parse.body@(2)[header_row$\twoheadrightarrow\{1\}$]

Header contents:

```
T[header@(HR,Col)->>String] :-
   U:country_url, T:(U.parse.table)[header_row->>HR],
   element(T,HR,Col)[contents->String;type->th].
```

$o_{belgiumGS}$header@(1,1)$\rightarrow$"province";. . . ;
header@(1,4)$\rightarrow$"pop.Dec.31,1991"].

Global Statistics: Table Columns

Identification of tables:

```
C[adm_div_tab -> T:admdivtab[name_col->0]] :-
      C:country[url@(gs)->U],
      T:(U.parse.table),
      element(T,_,_)[contents->I],
      pmatch(I,"/administrative divisions of (.*)",
             "$1",Name).
```

$o_{belgiumGS}$[admdivtab→$o_{belgiumGS}$.url@(gs).parse.body@(2)].

Identification of columns by keywords:

```
area_col:adm_div_tab_col[header->>{"area"}].
pop_col:adm_div_tab_col[header->>{"pop"}].
cap_col:adm_div_tab_col[header->>{"cap","haupt"}].

T[CName->Col] :- T:admdivtab[header@(_,Col)->>S],
         CName:adm_div_tab_col[header->>H],
         substr(H,S).
```

$o_{belgiumGS}$.admdivtab

[name_col→1; area_col=fd 3; pop_col→4; cap_col→2].

Global Statistics: Table Contents

```
C[adm_divs ->> prov(C,DRow):prov
        [country->C; name->PN;
         population->P; area->A]] :-
   C:country[adm_div_tab ->
        T[name_col->NS;area_col->AS;pop_col->PS]],
   element(T,DRow,NS)[contents->PN;type->td],
   element(T,DRow,AS)[contents->A;type->td],
   element(T,DRow,PS)[contents->P;type->td].

C[adm_divs ->> prov(C,DRow)[capital_name->CN]] :-
   C:country[adm_div_tab -> T[cap_col->CS]],
   prov(C,DRow):prov,
   element(T,DRow,CS)[contents->CN;type->td].
```

$o_{belgiumGS}$[adm_divs$\twoheadrightarrow\{o_{p_antwerp},\ldots\}$]

$o_{p_antwerp}$[name$\rightarrow$"Antwerp"; country$\rightarrow o_{belgiumGS}$;
population$\rightarrow$1610695; area$\rightarrow$3358;
capital_name$\rightarrow$"Antwerp"].

Global Statistics: Table Contents

Analogously: table of *main cities*

Global Statistics: Data Cleaning/References

```
Cty[province->>Prov] :-
    Cty:city[country->C;prov_name->PN],
    Prov:prov[country->C;name->PN].
P[capital->Capcty] :-
    P:prov[country->C;name->PN;capital_name->CN],
    Capcty:city[country->C;name@(gs)->>CN;
                province->>P[name->PN]].
```

$o_{antwerp}$:city[province→{$o_{p_antwerp}$}; ...]

$o_{p_antwerp}$[name→"Antwerp"; country→$o_{belgiumGS}$;
population→1610695; area→3358;
capital→$o_{antwerp}$].

Wrapping: Some Comments

- generic rule patterns for handling logical markup and textual formatting and typical situations of data integration,
- program skeleton completed by application-specific rules and refining rules (rapid prototyping).
- (semi-)automatical approaches for wrapper-generation: not (yet) provide a sufficiently fine granularity

Mediating: Integration and Restructuring

- every source defines a schema
- overlapping classes
- different names for objects

$\Rightarrow$ object fusion

```
C1 = C2 :- C1:Class[name@(A)->N],
           C2:Class[name@(B)->>N].
```

$o_{belgiumCIA} \equiv o_{belgiumGS}$

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

Inter-Source Links:

```
O[seat->Cty] :-
    O:org[seatcity->N; seatcountry->CN],
    Cty:city[name->> N], Cty.country[name@(cia)->>CN].
```

$o_{eu}[\text{seat} \rightarrow o_{brussels}]$.

III: Demonstration

- Separate program for each data-source
- Integration program computes common model
- Transformation into relational schema (SQL) or XML

IV: Conclusion

- practicable approach.
- unified model for Web representation and model of the application,
- integrated data model/language for wrapping, mediating, and querying.
- advantages of rule-based languages:
  - Prototyping, Refinement, Adaptability (autonomous sources)
- special features of F-Logic/FLORID:
  - variables at all positions

  $\Rightarrow$ powerful generic rules
  - object creation (wrapping)
  - object fusion (integration)
- Further Work:
  - “intelligent” wrapping (analyzing of tables)
  - Usage with search engines
  - Mapping XML to F-Logic
- Problems:
  - The more generic the rules ... the bigger the joins.
  - FLORID: only basic optimizations (prototype)
  - FLORA ?