Information Extraction from the Web with FLORID

Wolfgang May
Institut für Informatik
Universität Freiburg
Germany
may@informatik.uni-freiburg.de

TU Wien
Nov. 5th, 1999

Overview

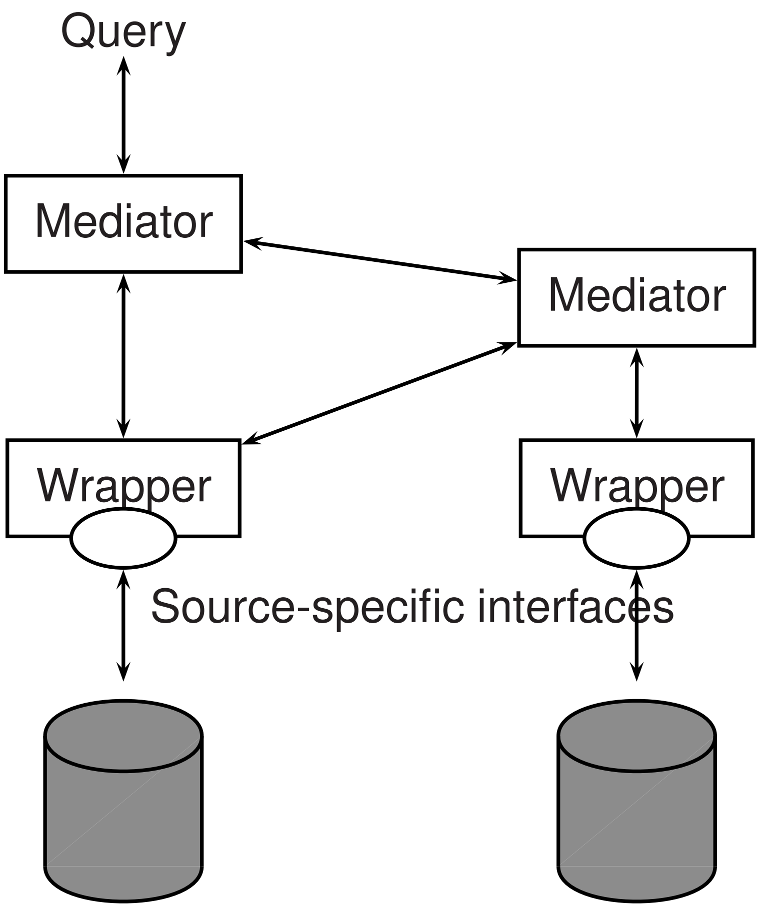
- Integrated Architecture for
  - Web querying
  - Web Data Extraction
- Unified World Model: Web structure + application level
- Unified Language for Exploration, Wrapping, Mediating, Querying
- Implementation: F-Logic/FLORID
- Examples / Case Studies:
  - DBLP Publications Web Server (single-site)
  - Geographical Information (multi-site)

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)

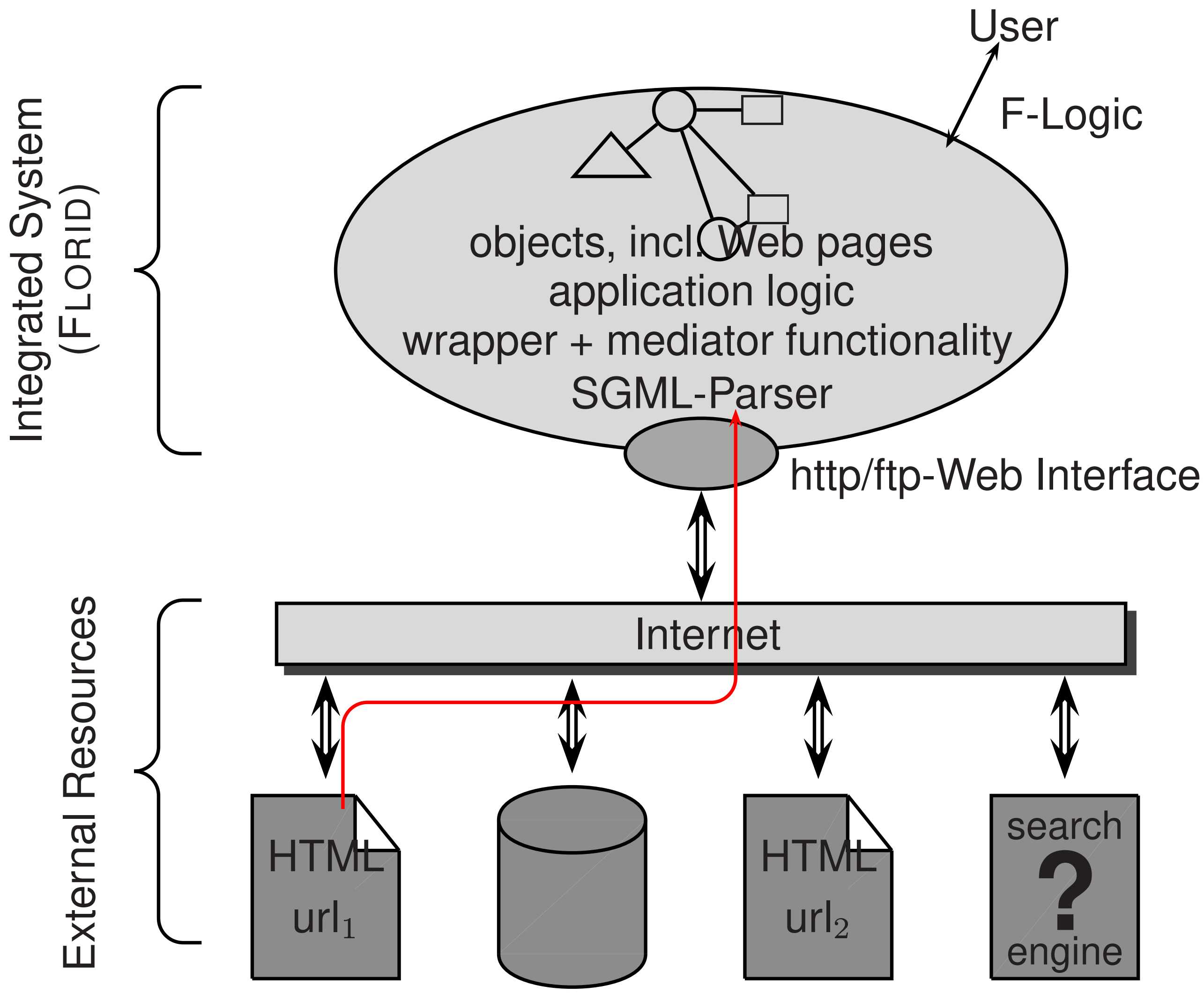
Common Architecture:

Two-Level Wrapper & Mediator – Architecture



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

Architecture of Web Access with FLORID



- Unified, integrated framework for wrappers and mediators
- F-Logic: unified data model, wrapper, mediator, and querying language
- Data Model: Representation of the Web fragment and application-level representation.
  Structure + Contents of the Web as a unit

The Web Model

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

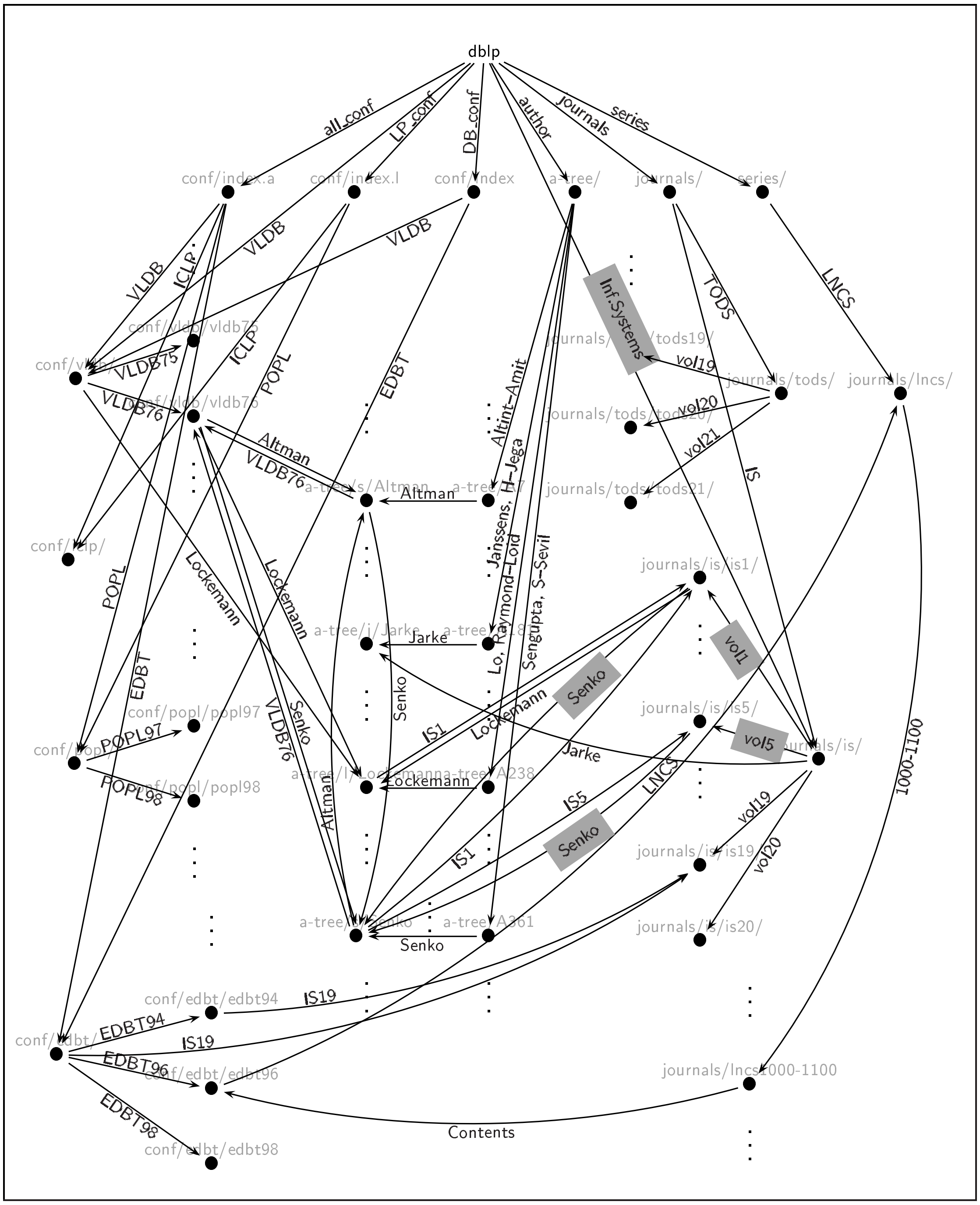
The Skeleton $\mathcal{SK}$: URL's and Web Documents

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

$(x, \ell, y) \in \mathcal{SK} \Leftrightarrow$ the Web document x contains a hyperlink labeled with ℓ to the Web document y.
("$\texttt{<a href} = y > \ell </\texttt{a}>$")

Example: The DBLP Server

Example: The DBLP Server

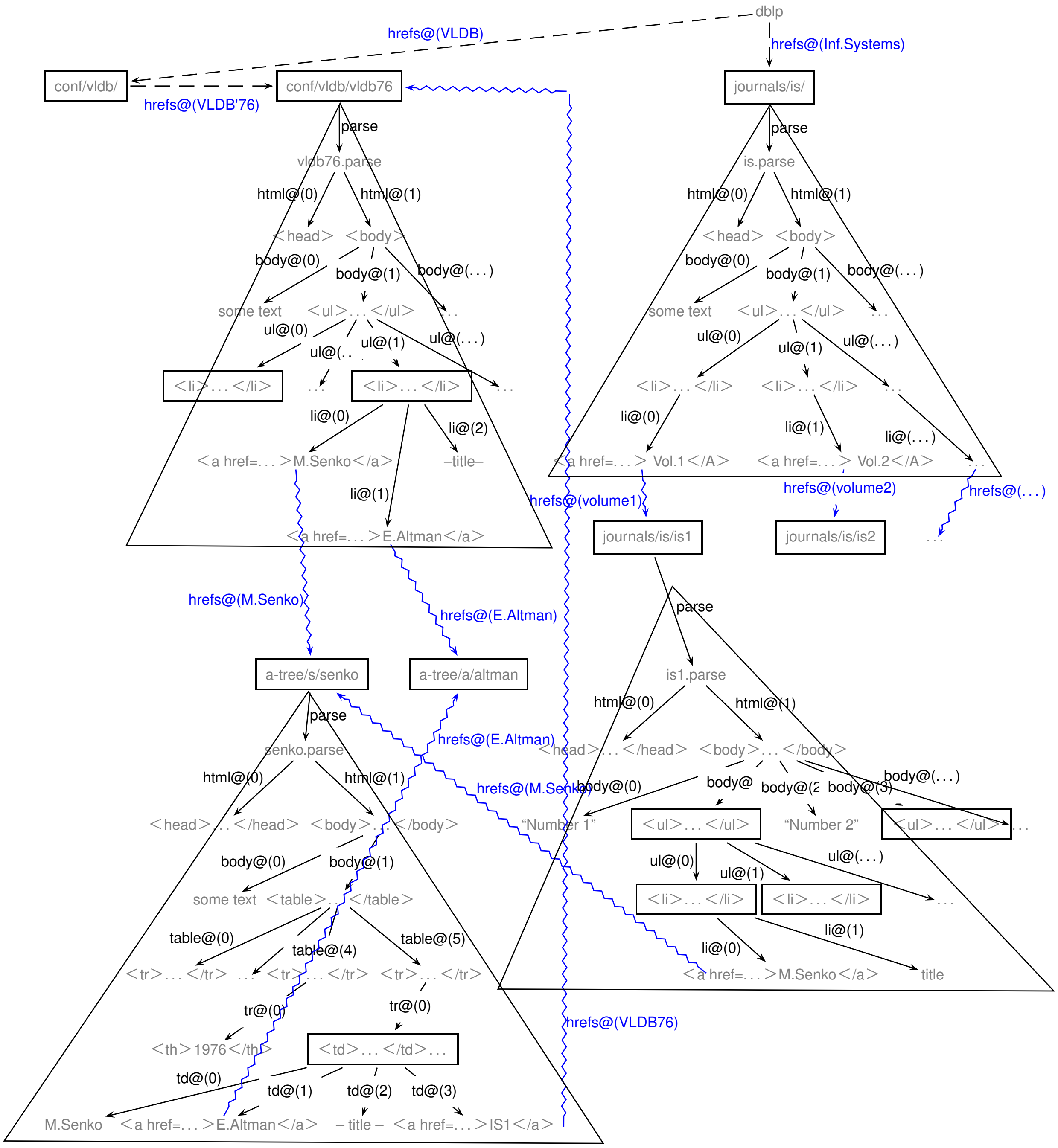


- skeleton: Web pages and hyperlinks
- corresponds to real world objects:
  journals, conferences, books, and authors

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.

$\Rightarrow$ base for (structural) *Web querying*

- (semantical) Web data extraction:
  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 on the Implementation

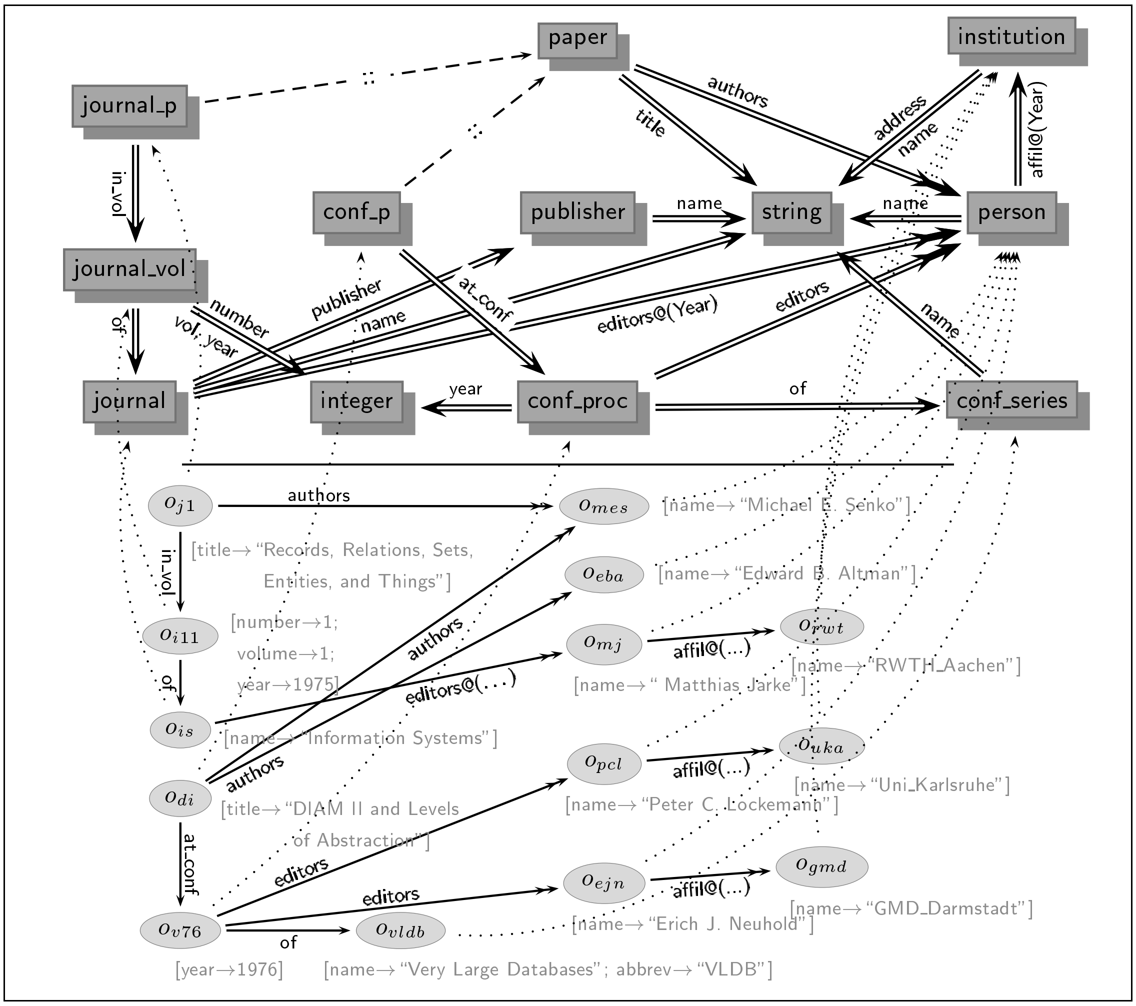
- Logical issues:
- Suitable modeling (classes)
- Object creation on demand
- Object fusion
- Navigation in the model
- Powerful, flexible reasoning

- non-logical features ⇒ built-ins:
- Web Access via http-protocol,
- Parsing of HTML/SGML/XML,
- Matching with Perl Regular Expressions,

Formal Framework: F-Logic

- *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$\rightarrow$v] (scalar)
  o[m$\twoheadrightarrow$v] (multivalued)
  analogous with arguments: o[m@(x$_1$,...,x$_n$)$\rightarrow$v].
  inheritable:
  c[m$\bullet\!\!\rightarrow$v]
  c[m$\bullet\!\!\twoheadrightarrow$v]
- Signatures of methods:
  c[m$\Rightarrow$v] (scalar)
  c[m$\Rightarrow\!\!\!\!\Rightarrow$v] (multivalued)
- Variables allowed at all positions
- Entities can act at the same time 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$\Rightarrow\!\!\!\!\Rightarrow$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$\twoheadrightarrow$$\{o_{mes}\}$; in_vol→$o_{i11}$].

o_{di} : conf_p[title→ "DIAM II and Levels of Abstraction"; authors$\twoheadrightarrow$$\{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@(...)$\twoheadrightarrow$$\{o_{mj}\}$].

o_{v76} : conf_proc[of→o_{vldb}; year→1976; editors$\twoheadrightarrow$$\{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"]. ...

Further F-Logic Features

- 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[...] $\leftarrow$ ...

- Derived equality via object fusion:
  $o_1 = o_2 \leftarrow \ldots$
  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]}.

- nonmonotonic inheritance, equivalent to Default Logic:

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

- FLORID: bottom-up inflationary semantics with user-defined stratification
  (deductive + procedural elements)

Exploration of the Web

- classes url and webdoc
  (subclasses of string).
- class url: url.get implemented as an *active* method (C++):

  u.get :- ...

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

u.get[hrefs@(ℓ) $\twoheadrightarrow$ u'] $\Leftrightarrow$
u.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 $u.\mathsf{get}$ and $u.\mathsf{parse}$) if

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

... integrated into the T_P-operator:

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

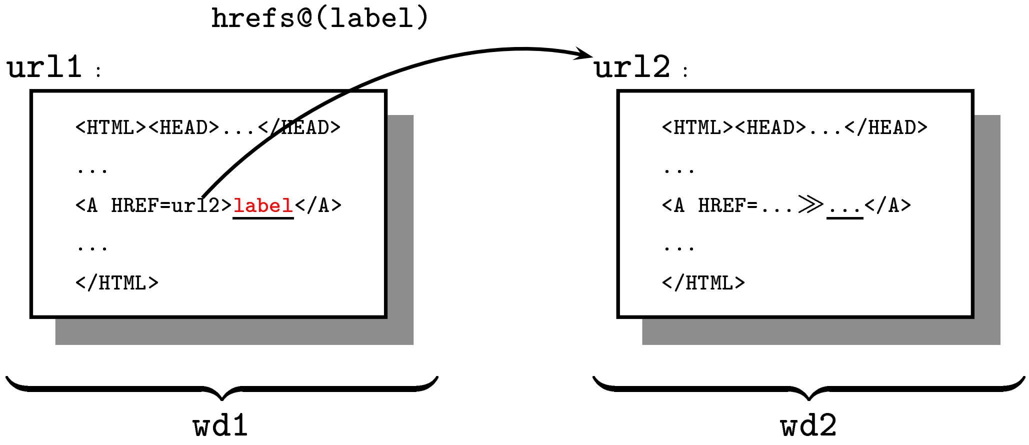
$$\begin{aligned}
T_P(\mathcal{H}) \quad &:= \mathcal{H} \cup \{h \mid (h \leftarrow body) \in ground(P), \mathcal{H} \models body\} \ , \\
T_P^{\mathcal{W},0}(\mathcal{H}) \quad &:= \mathcal{H} \ , \\
T_P^{\mathcal{W},i+1}(\mathcal{H}) &:= \mathcal{C}\ell(T_P(T_P^{\mathcal{W},i}(\mathcal{H})) \\
&\qquad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

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

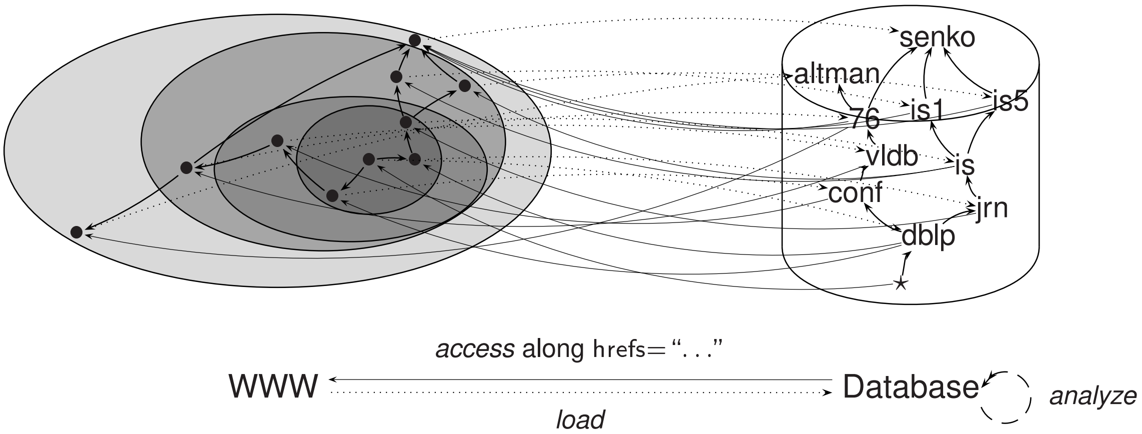
Data-Driven Web Exploration

- 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



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 $<tag>\ \ldots\ </tag>$ 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].

Wrapping

- url.get, url.parse: *raw*, uninterpreted data

$\Rightarrow$ (Extended) Web Skeleton
allows for simple syntax-based and structure-based Web querying

$+$ semantic 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

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 ...

Example: DBLP Server

constructing the application model:

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

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

dblp.journals_page.get.
dblp.conf_page.get

% conferences

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

(S.year@(Year) : conf) [series→S; year→Y; url→(U:url)],
U.get ← % “VLDB”@(1976)
(S : conf_series).url.get[hrefs@(“Contents”)↠U],
pmatch(U,“/[A-z]*([0-9]*).html/”,“19$1”,Year).

% ... 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 and pages
P[title→T] ← P : paper, (P.parsenode.li@(_):b)[b@(0)→T], string(T).
P[pages→N] ← P : paper, P.parsenode[li@(_)→N], string(N).

% papers: authors
N : author[name→Name; url↠(U:url)], P[authors↠N] ←
    (((P : paper).parsenode.li@(_)):a)[href→U, a@(0)→Name].

Example: DBLP Server

% authors pages

U.get $\leftarrow$ A:author[url$\twoheadrightarrow$U]

% authors homepages

A[homepage$\rightarrow$(U:url)], U.get $\leftarrow$
    A:author.url.get[hrefs@(“Homepage”)$\twoheadrightarrow$U].

$\Rightarrow$ Data-driven Web exploration

Example: DBLP Server

- single-site source
- “best case”-example
- well-structured HTML/SGML
  (generated from an underlying database)
- parser-based wrapping
- Model contains Web skeleton, parse-trees, and application

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

Generic Wrapping Rules

Case Study: The MONDIAL Geographic Database

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

⇒ 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)$\rightarrow$"…";
name@(cia)$\twoheadrightarrow$"Belgium";
continent$\rightarrow$$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:

TEXT PATTERNS +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:

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] :-                                COMMALISTS
  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],      DECOMPOSING
            pmatch(S,"/([^,]*)/","$1",X).   COMMALISTS
```

$o_{belgiumCIA}$
[borders↠{ "France 620 km", "Germany 167 km", ... }].

Countries: Name-Value-Pairs

DECOMPOSING PAIRS + REFINEMENT

```
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]).

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})$\rightarrow$620;   .
borders@($o_{germany}$)$\rightarrow$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:

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

LIST OF PROPERTIES REFINED

```
O[MethName:Type->Value2] :-
    O:org[data->D],
    property(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 36):

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

Seat:

```
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”;
establ@(date)→ “07 02 1992”].

Organizations: Memberlists

generic rule pattern for commalists (cf. slide 37)

DECOMPOSING COMMALISTS

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

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

Flexibility

- autonomous sources: layout of an information source can change

CIA Factbook 1997: layout of the organizations page has been changed from HTML lists to HTML tables (without table row/column semantics, only for optical markup)

Changes:

- the first, source-specific rule which decomposes the outermost <UL> has to be redesigned,
- instead of LIST OF PROPERTIES, another rule TABLE OF PROPERTIES has to be employed:

```
O[MethName:Type->Value] :-          TABLE OF PROPERTIES
  O:org[data->D],
  property(org,MethName,MethString,Type),
  D.td@(2)[p@(X)->MethString],
  D.td@(2)[p@(Y)->Value],      Y = X + 1.
```

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].
```

Tables

Mapping Tables to a matrix (+ Refinements):

TABLES: ELEMENTS

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

$o_{belgiumGS}$.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"].

Extracting Column Headers

Identifying the header row(s)

TABLES: HEADER ROWS

```
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:

TABLES: 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].
```

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

header@(1,1)$\rightarrow$ "province";...;

header@(1,4)$\rightarrow$ "pop.Dec.31,1991"].

Identifying Tables

```
admdivtab:countrytable[
  tabheader->>"administrative divisions";
  method->adm_divs; resultclass->province;
  columns->>{prov_name_col,prov_pop_col,
             prov_area_col,prov_cap_col}].
maincitytab:countrytable[
  tabheader->>{"municipalities", "main cities"};
  method->main_cities; resultclass->city;
  columns->>{city_name_col,city_pop_col,city_prov_col}].
```

IDENTIFYING TABLES

```
C[TabName -> T[col(name)->O]] :-
   C:country[url@(gs)->U], T:(U.parse.table),
   element(T,_,_)[contents->I],
   TabName:countrytable[tabheader->TH], substr(I,TH).
```

$o_{belgiumGS}$[admdivtab→$o_{belgiumGS}$.url@(gs).parse.body@(2)]
and
$o_{belgiumGS}$[maincitytab→$o_{belgiumGS}$.url@(gs).parse.body@(3)]

Identifying Table Columns

```
prov_area_col[attr->area; header->>{"area"}].
prov_pop_col[attr->population; header->>{"population"}].
prov_cap_col[attr->capital_name;
             header->>{"capital","hauptstadt"}].
city_pop_col[attr->pop; header->>{"pop"}].
city_prov_col[attr->prov_name; header->>{"prov"}].
```

IDENTIFYING TABLE COLUMNS

```
C.TabName[col@(Attr)->Col] :-
  (C:country).(TabName:table)[header@(HR,Col)->S],
  TabName[columns->>TC[attr->Attr; header->ColHdr]],
  substr(S,ColHdr).
```

$o_{belgiumGS}$.admdivtab[col@(area)→2;
col@(population)→3;
col@(capital_name)→1]

$o_{belgiumGS}$.maincitytab[col@(pop)→1]

Extracting Table Contents

TABLE: DATA EXTRACTION

```
C[Method ->> obj(T,DRow):ResultClass
                         [country->C; Attr->Val]] :-
   C:country[(Whichtable:countrytable)->T]
   Whichtable[method->Method;
              resultclass->ResultClass],
   T[col@(Attr)->Num],
   element(T,DRow,Num)[contents->Val; type->td].
```

Example Bindings:

C = $o_{belgium}$

Whichtable = admdivtab

Method = adm_divs

ResultClass = prov

T = $o_{belgiumGS}$.admdivtab

(Attr, Num) $\in$ {(name,0), (capital_name,1), (area,2), (population,3)}

DRow: all table rows which contain td data.

$o_{belgiumGS}$[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"].

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$\rightarrow\{o_{p_antwerp}\}$; $\ldots$]

$o_{p\_antwerp}$[name$\rightarrow$ "Antwerp"; country$\rightarrow o_{belgiumGS}$;
population$\rightarrow$1610695; area$\rightarrow$3358;
capital$\rightarrow 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, modular design).
- (semi-)automatical approaches for wrapper-generation: not (yet) provide a sufficiently fine granularity
- XML: Meta data (can be) included in DTD

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].
```

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

... additional to “standard” HTML pages:

Querying Forms

- Search engines,
- HTML interfaces to database Web servers (e.g., Oracle)
- calls to CGI scripts
- <form>-tag
- query transmission:
  - GET method: customized url with parameters
    $\Rightarrow$ feasible inside the current framework
  - POST method: parameters are received by the cgi script via standard input.
    $\Rightarrow$ requires special internal functionality; perl-methods [To Do]

Generic Wrapper for Forms

<form method = GET action = *cgi-url*>
<input name=p>
<input type=hidden name=p value=v>
</form>

Query url: `cgi-url?`p_1`=`v_1`&...&`p_n`=`v_n

- input values to the form: object *query*; e.g., query[author→"M.E.Senko"; year→"1975"].

% initialize query url:
F[createurl@(0)→U1]:- F:(X:url.parse.form),
F[method→"GET"; action→BaseURL],
strcat(BaseURL,"?",U1).

% incrementally working on the tags inside the <form> tag:
% next tag is a non-hidden input:
F[completeurl@(M)→U0; createurl@(M)→U1] :-
F:(X:url.parse.form)[createurl@(N)→U], M = N + 1,
F.form@(N):(X.parse.input) [type→"text"; name→N],
query[name→V],
perl(strings.concatenate,[U,N,"=",V],U0), strcat(U0,"&",U1).

% next tag is a hidden input
% skip: next tag is not an input tag

% no more elements inside <form>: access generated url.
U.get :- F:(X:url.parse.form)[completeurl@(N)→U],
not F.form@(N).

Example: Wrapping Alta Vista

The AltaVista form:

```
<form method="GET" action="/cgi-bin/query" name=mfrm>
<input type=text name=q value="">
</form>
```

Submitting the query:

```
altavista[bin_url->
          "http://www.altavista.com/cgi-bin/query?q="].
altavista[searchtext->"Freiburg"].
U:resulturl :-
    strcat(altavista.bin_url, altavista.searchtext, U).
```

“http://www.altavista.com/cgi-bin/query?q=Freiburg”

```
resulturl::url.
U.parse :- U:resulturl.
```

Example: Wrapping Alta Vista

Extracing hits from result pages

- hits are presented in <dl> tags:

```
<dl><dt><b>1. </b>
  <a href="http://web.mathematik.uni-freiburg.de/...">
  <b> Faculty of Mathematics,
      Freiburg University</b></a><dd>
  Albert-Ludwigs-Universitaet Freiburg. ...
  <b>URL:</b>
  <font color=gray>web.mathematik.uni-freiburg.de/...<br>
  Last modified 11-May-99 - page size 4K -
    in English</font></dl>
```

```
 X:hit[url -> HitURL:url; title -> T; mod -> M;
       year -> Y; size -> S; lang -> L] :-
   X:(U:resulturl.parse.dl),
   X.dl@(0).dt@(1):a[href -> HitURL].a@(0)[b@(0)-> T],
   X.dl(0).dt@(4)[font@(3) -> Info:string],
   pmatch(Info,"/Last modified ([0-9]*-[A-z]*-)([0-9]*) -
                page size ([0-9]*)K - in ([A-z]*)/",
              "[$1$2,$2,$3,$4]", "[M,Y0,S,L]"),
   string2real(Y0,Y).
```

Example: Wrapping Alta Vista

Extracing hits from result pages

- every result page contains a “Next” hyperlink to the subsequent result page:

```
<a href="/cgi-bin/query?pg=q&q=Freiburg&stq=10&c9k">
   [<b>Next</b>]</a>
```

```
NextURL:resulturl :-
   X:(U:resulturl.parse.a),
   X[href->LocalNextURL].a@(1)[b@(0)->B],
   substr("Next",B),
   strcat("http://www.altavista.com",
          LocalNextURL, NextURL).
```

... “continue” with

```
resulturl::url.
U.parse :- U:resulturl.
```

Note that this wrapper does not contain any stratification!

Example: Wrapping Alta Vista

The hits can further be filtered:

```
HitUrl.get:relevant_page :-
   X:hit[url->HitURL; year->M; lang->"English"],
   M > 1997, not pmatch(HitURL,"/.com\Z/","$1",X).
```

Then the selected pages can be accessed and processed as usual.

Resulting pages:

- structure and layout of the resulting pages not known
- only rough heuristics, based on text search
- intelligent wrapping techniques

Forms: Applications and Perspectives

- Data-driven Web exploration in combination with Web search engines:
- Refining Search Results from Web search engines (see above),
- Combining the results of different Web search engines,
- Data-Driven queries on Web Indexes:
  query Web indexes depending on previously extracted data.

  e.g., query AltaVista for tourist information pages of all capitals.

Demonstration

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

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

  ⇒ powerful generic rules
  - object creation (wrapping)
  - object fusion (integration)
- Further Work:
  - “intelligent” wrapping (analyzing of tables)
  - Mapping XML to F-Logic