A Unified Framework for Wrapping, Mediating and Restructuring Information from the Web

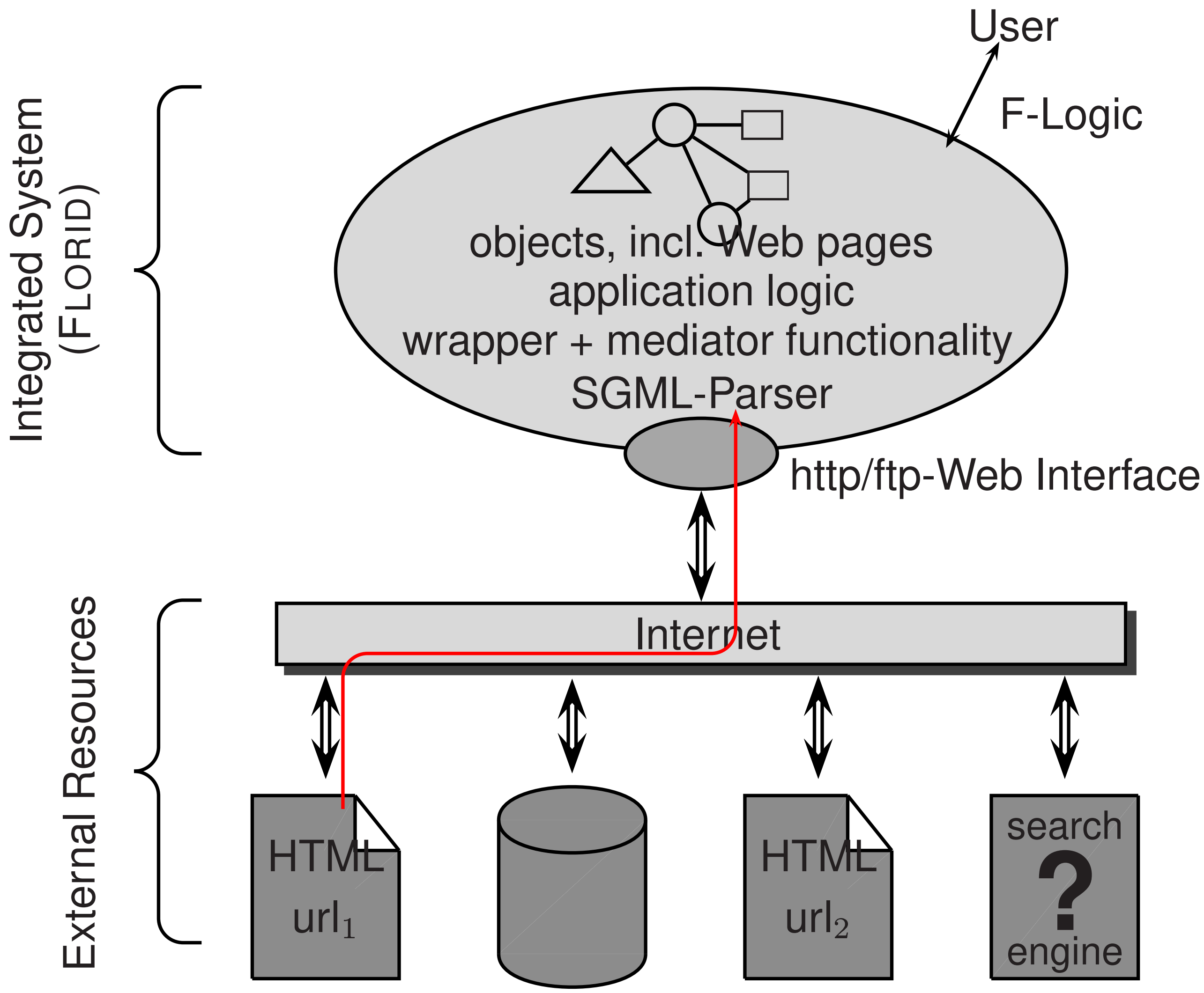
Wolfgang May Rainer Himmeröder
Georg Lausen Bertram Ludäscher
Institut für Informatik
Universität Freiburg
Germany

International Workshop on the World-Wide Web and Conceptual Modeling (WWWCM'99)
November 17th, 1999

Overview

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

Architecture of Web Access with FLORID

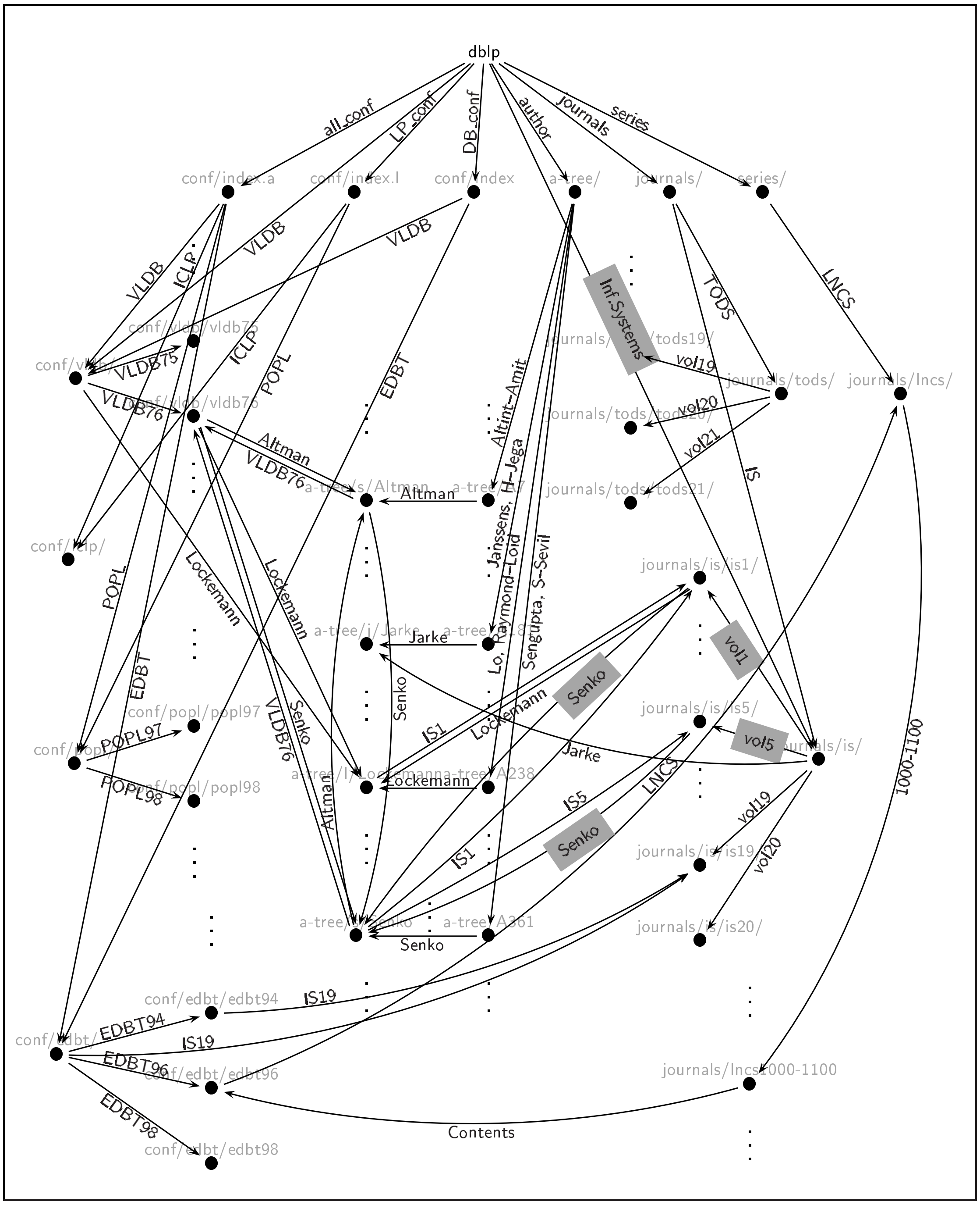


- Unified, integrated framework for wrappers and mediators
- F-Logic: unified data model, wrapper, mediator, and querying language
- Data Model: Representation of the Web fragment and application-level representation.
  Structure + Contents of the Web as a unit

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.

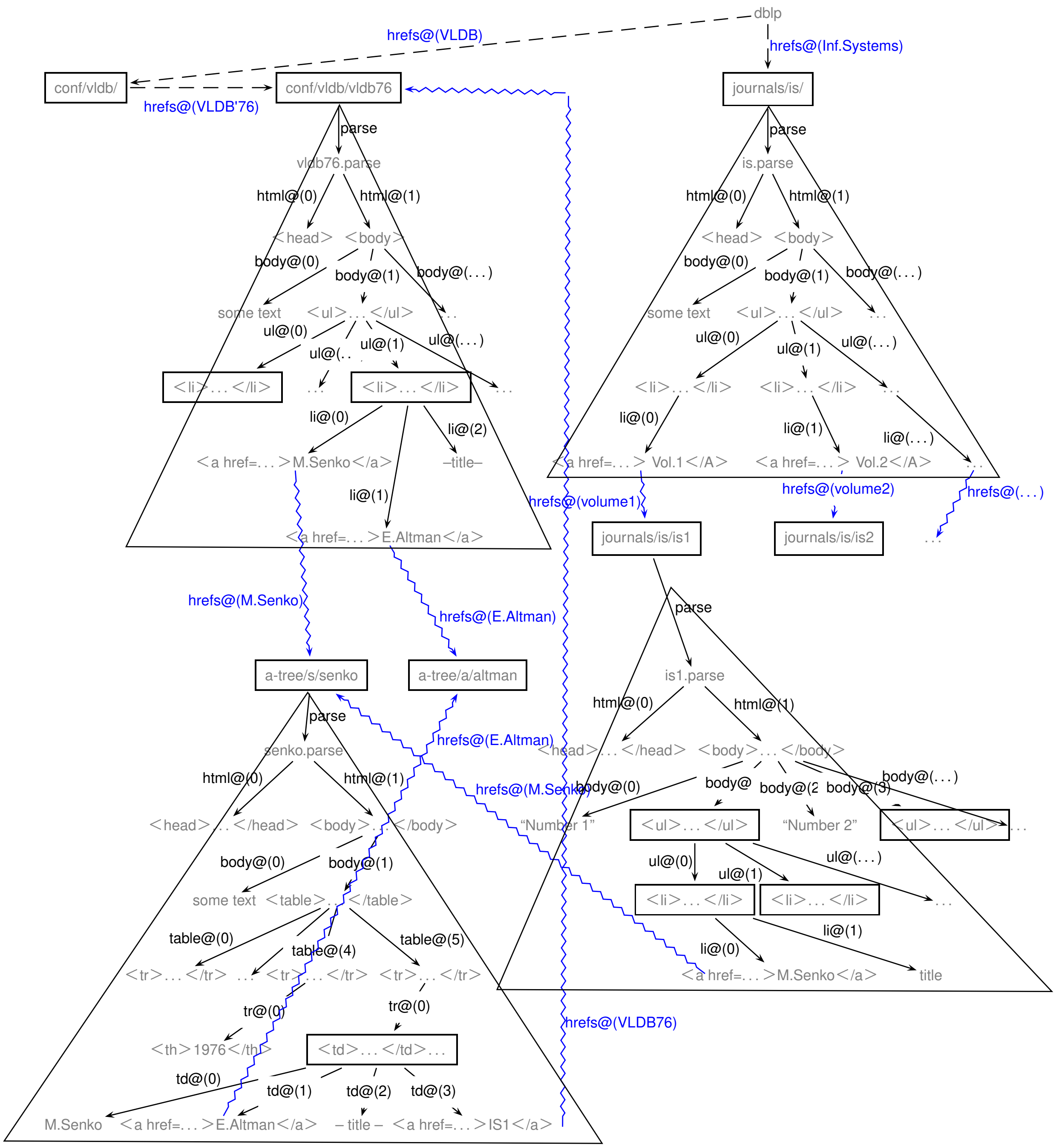
Example: The DBLP Server



- skeleton: Web pages and hyperlinks

Extending the Web Skeleton: Parse-trees

- integration of *parse-trees*



- Details: Modeling and Querying Structure and Contents of the Web, Workshop on Internet Data Modeling (IDM'99)

Extended Web Skeleton

- *extended Web skeleton*: unified – but still based on the Web representation, not on the application semantics.

$\Rightarrow$ (semantical) Web data extraction

- 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?

Formal Framework and Implementation: 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 properties, signatures of methods)
- 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: Geographic Database

o_{belg} : country[name→ "Belgium"; capital→$o_{brussels}$;
total_area→30510; population→10170241; continent@(o_{eur})→100;
indep@(date)→ "04 10 1830"; pop_growth→#0.33;
adm_divs→→ $\{o_{p\_antwerp}, o_{p\_westfl}, \ldots\}$;
main_cities→→ $\{o_{brussels}, o_{antwerp}, \ldots\}$;
religions@("Roman Catholic")→75; religions@("Protestant")→25;
borders@(o_{france})→620; borders@($o_{germany}$)→167;
borders@($o_{luxembourg}$)→148; borders@($o_{netherlands}$)→450].

$o_{brussels}$: city[name→ "Brussels"; country→o_{belg}; province→o_{p_westfl};
longitude→#4.35; latitude→#50.8; population@(95)→951580].

$o_{antwerp}$: city[name→ "Antwerp"; country→o_{belg}; province→$o_{p\_antwerp}$;
population@(95)→459072; longitude→#4.23; latitude→#51.1].

$o_{p\_antwerp}$: prov[name→ "Antwerp"; country→o_{belg};
capital→$o_{antwerp}$; area→2867; population→1610695].

o_{p_westfl} : prov[name→ "West Flanders"; country→o_{belg};
capital→$o_{brussels}$; area→3358; population→2253794].

o_{eu} : org[abbrev→ "EU";name→ "European Union";
establ→ "07 02 1992"; seat→$o_{brussels}$;
members@("member")→→ $\{o_{belg}, o_{france}, \ldots\}$;
members@("membership applicant")→→ $\{o_{hungary}, o_{slovakia}, \ldots\}$].

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']

  o_{eu}.seat $= o_{brussels}$, o_{eu}.seat.province $= o_{p\_westfl}$.

  ?- C : country[name$\rightarrow$"Belgium"], C..main_cities[name$\rightarrow$N].

- 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.

- Aggregation sum, count

- 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++):

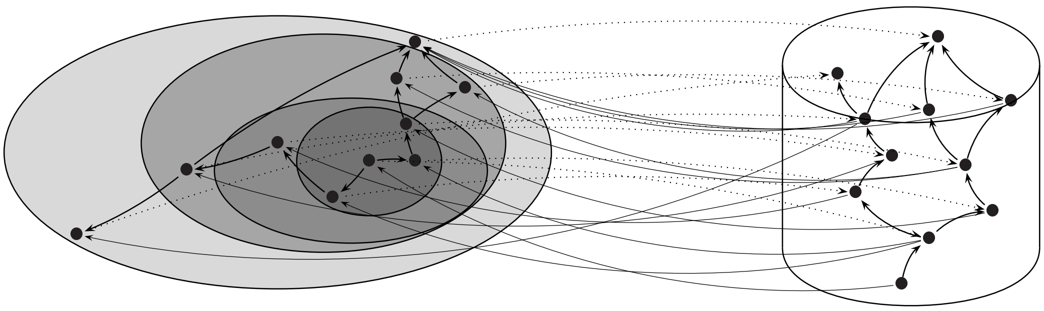
  u.get :- ...

  – 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, among others, the hyperlinks (Web skeleton):

$$u.\text{get}[\text{hrefs}@(\ell) \twoheadrightarrow u'] \Leftrightarrow$$
$$u.\text{get} \;\; \text{contains} \text{ ``}\texttt{<a href} = u' > \ell\, \texttt{}\text{''}\,.$$

Data-Driven Web Exploration

- *data-driven* Web access and wrapping



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: $o.<tag>$@(0), $\ldots$, $o.<tag>$@(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).

Wrapping

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

⇒ (Extended) Web Skeleton
allows for simple syntax-based and structure-based Web querying

\+ semantic wrapping by F-Logic Rules

- Logical Markup:
  Parser-based

\- direct correspondence between HTML-nodes and objects (extended Web skeleton).

- Optical and Syntactical Markup:
  pattern matching via regular expressions

More than Parsing

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

Pattern Matching in FLORID

Perl regular expressions by the built-in predicate

pmatch(<string>,"/<regexp>/", [<fmt-list>], [$X_1, \ldots, X_n$])

Perl-Interface

perl(<perl-routine-as-string>,

[<variable-list$_1$>], [<variable-list$_2$>])

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

- String Manipulation
- Prototyping of built-ins

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

$\Rightarrow$ Integration

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

CIA World Factbook: Countries

Country Pages: Optical Markup

Belgium

Geography

Location: Western Europe, bordering the North Sea, between France and the Netherlands
Area: *total area:* 30,510 sq km
border countries: France 620 km, Germany 167 km, Luxembourg 148 km, Netherlands 450 km

People

Population: 10,170,241 (July 1996 est.)
Ethnic divisions: Fleming 55%, Walloon 33%, mixed or other 12%
Religions: Roman Catholic 75%, Protestant or other 25%
Languages: Dutch 56%, French 32%, German 1%, legally bilingual 11% (divided along ethnic lines)

Government

Capital: Brussels
Independence: 4 October 1830 (from the Netherlands)

Economy

GDP: purchasing power parity – $197 billion (1995 est.)

Processing: By Matching

CIA World Factbook: Countries

```
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,"/(.*) \([0-9]+ KB\)/","$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).
```

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).
```

COMMALISTS

```
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, . . .”]

DECOMPOSING COMMALISTS

```
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

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].
```

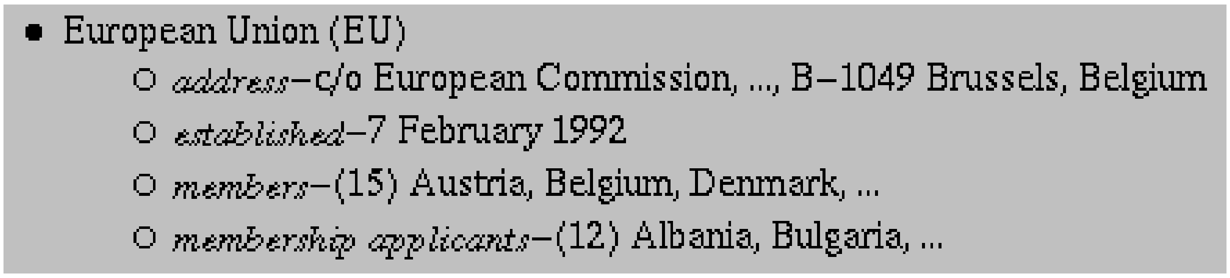
$o_{belgiumCIA}$[borders@(o_{france})$\rightarrow$620; .
borders@($o_{germany}$)$\rightarrow$167; ...].

CIA World Factbook: Organizations

Organizations Pages:

Logical Markup (lists) + Optical Markup (commalists)

Processing: Parsing + Matching



- European Union (EU)
    - *address*–c/o European Commission, ..., B–1049 Brussels, Belgium
    - *established*–7 February 1992
    - *members*–(15) Austria, Belgium, Denmark, ...
    - *membership applicants*–(12) Albania, Bulgaria, ...

```
<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>
```

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

A list for every letter:

```
L:org_list :-
   ciaAppC.parse.root[html@(X):-L:(ciaAppC.parse.ul)].
```

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 25):

```
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 26)

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

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)

administrative divisions of Belgium

province	capital	area (km^2)	pop. Dec. 31, 1991
Antwerp	Antwerp	2,867	1,610,695
Brabant	Brussels	3,358	2,253,794

main cities of Belgium

cities	pop. 1995 est.
Brussels (Agglomeration)	951,580
Antwerp	459,072

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$\rightarrow$"area (km^2)";type$\rightarrow$"th"].

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

⇒ 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}[seat$\rightarrow o_{brussels}$].

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
- Further Work:
  - “intelligent” wrapping (analyzing of tables)
  - Mapping XML to F-Logic