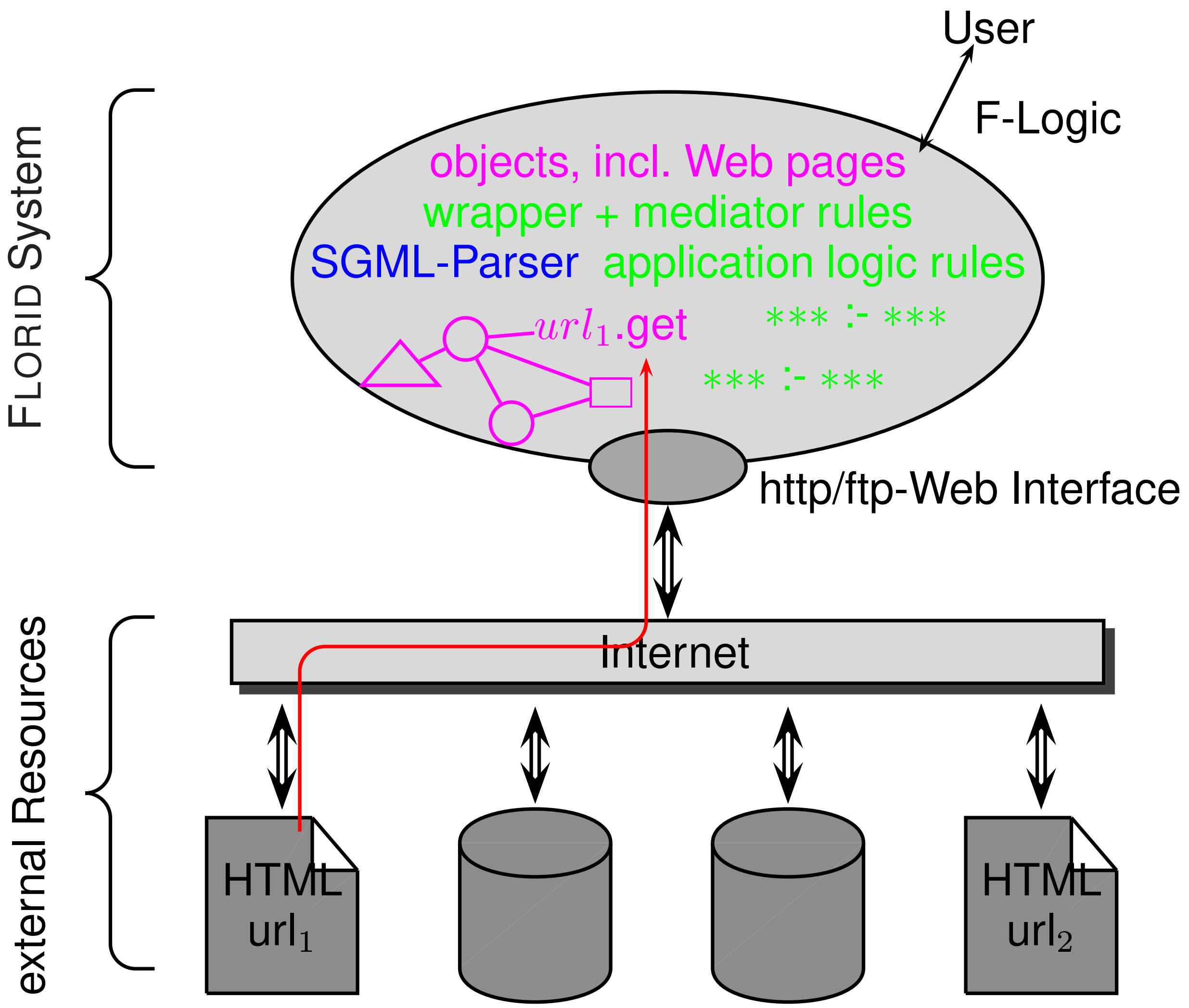
Modeling and Querying Structure and Contents of the Web

Wolfgang May
Institut für Informatik
Universität Freiburg
Germany

Overview

- Integrated Architecture for Web Data Extraction
- Unified World Model
- Implementation: F-Logic/FLORID
- Examples / Case Studies:
  - the DBLP Publications Web Server (single-site)
  - Geographical Information (multi-site)

Integrated Architecture



- Unified, monolithic 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
  - 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: URL's and Web Documents

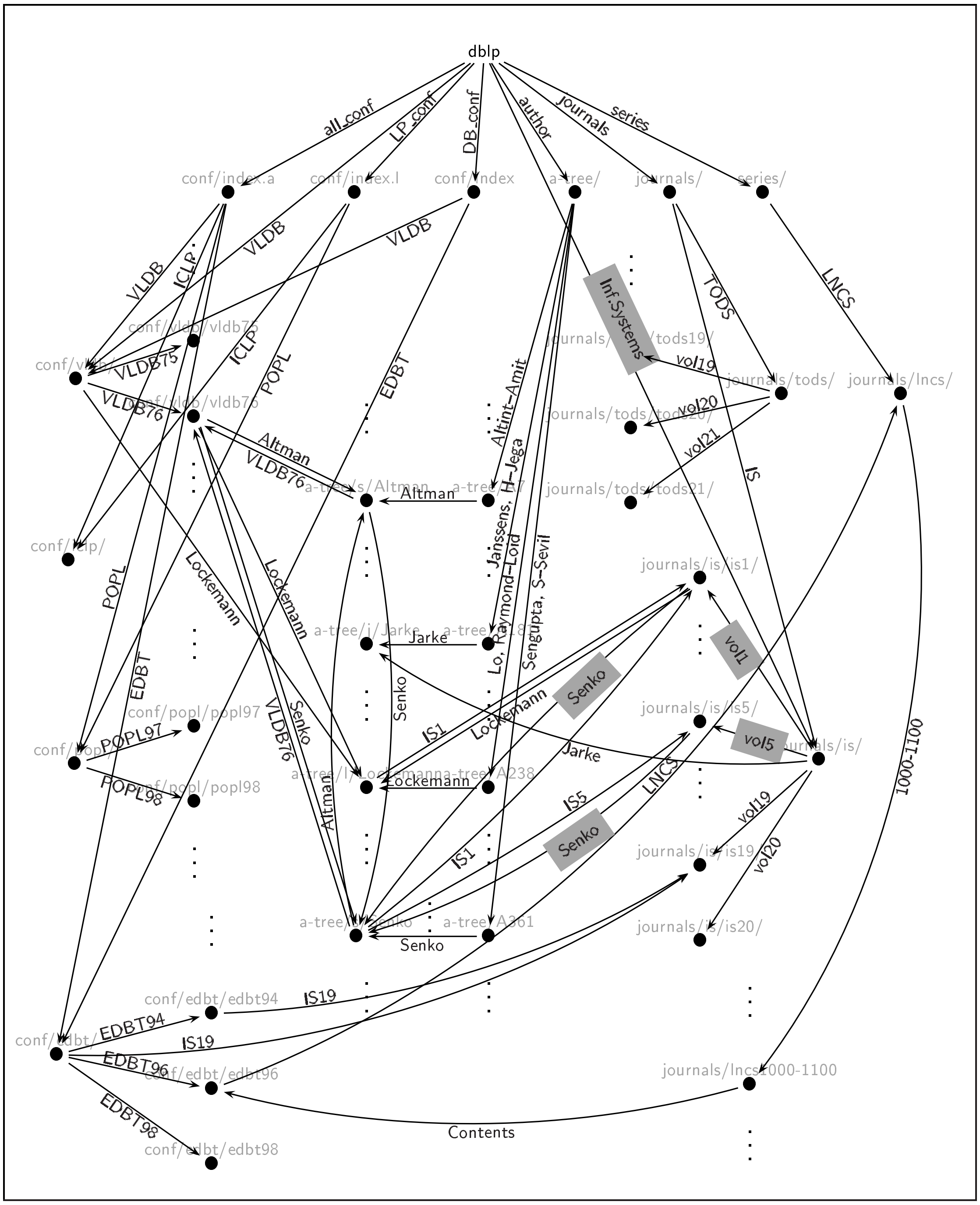
- Every resource in the Web has a unique *url*.
- document associated with a url contains *hyperlinks* to other url's

$(x, \ell, y) \in \mathcal{SK} \Leftrightarrow$ the Web document x contains a hyperlink labeled with ℓ to the Web document y.

(“$\texttt{<a href} = y > \ell \texttt{}$”)

Example: The DBLP Server

Example: The DBLP Server

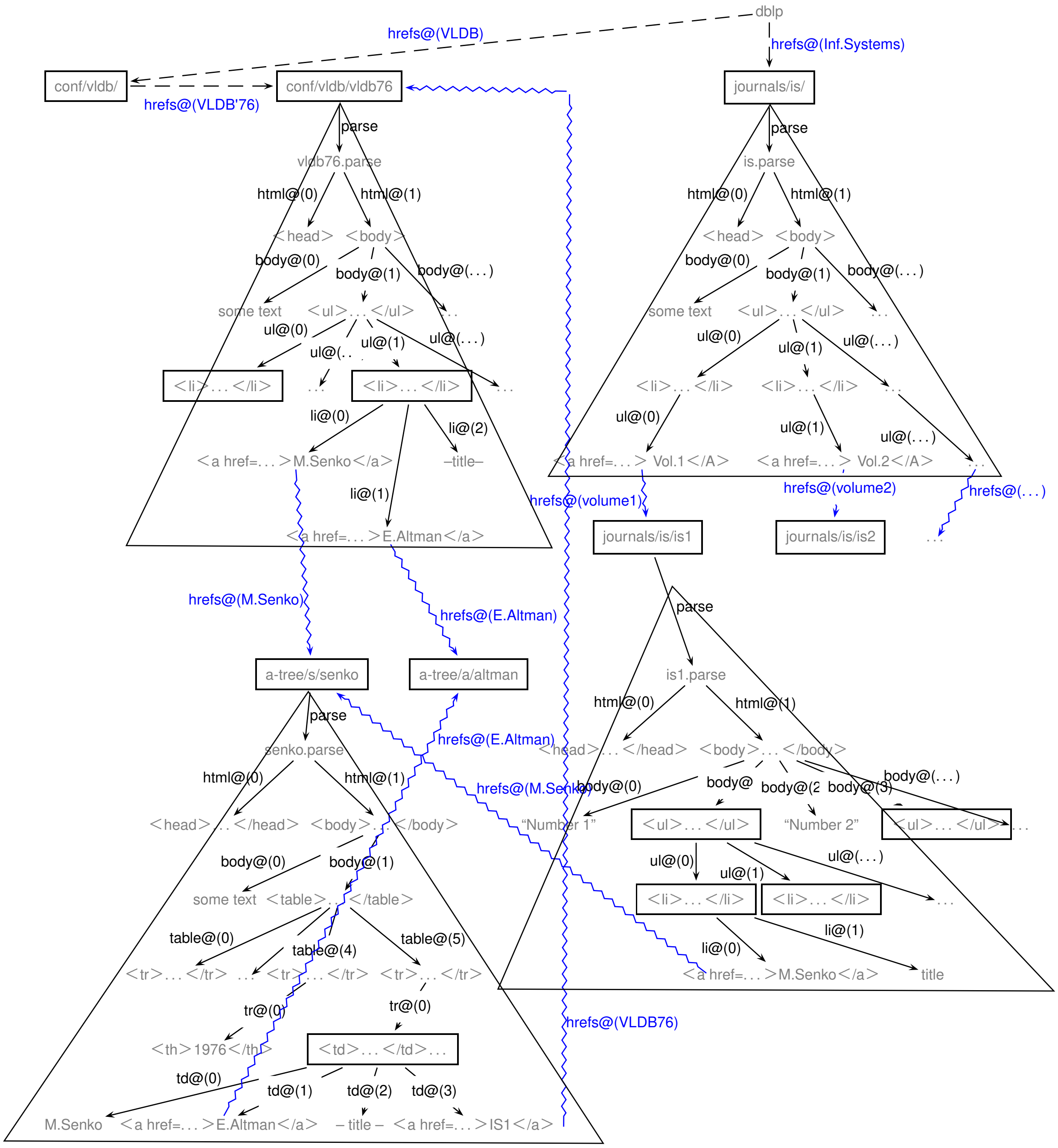


- skeleton: Web pages and hyperlinks
- corresponds to real world objects:
  journals, conferences, books, and authors

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



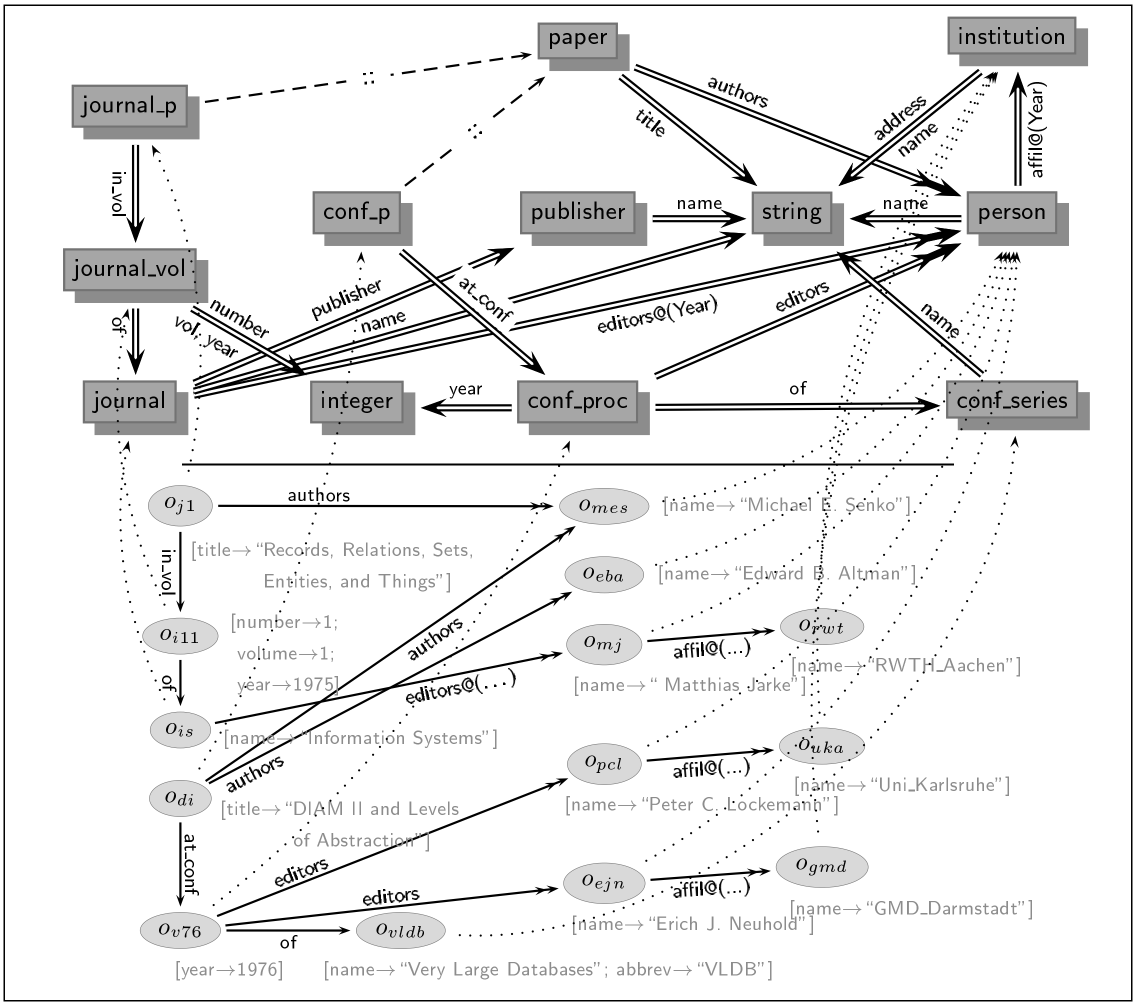
Extended Web Skeleton

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

Formal Framework: F-Logic

- object-oriented database language
- *id-terms* are composed from object constructors and variables (capital letters) 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 at the same 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"]. ...

Formal Framework: F-Logic

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

  ?- P : conf_proc.of[abbrev→“VLDB”], P[year→1976],
  P..editors[affil@(1976)→A].

- object creation by path expressions in the head:
  o.m[. . .] ← . . .

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

- Aggregates: sum, count, ...

- nonmonotonic inheritance

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

Requirements for Implementation of Web Access

- 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

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'] $\Leftrightarrow$

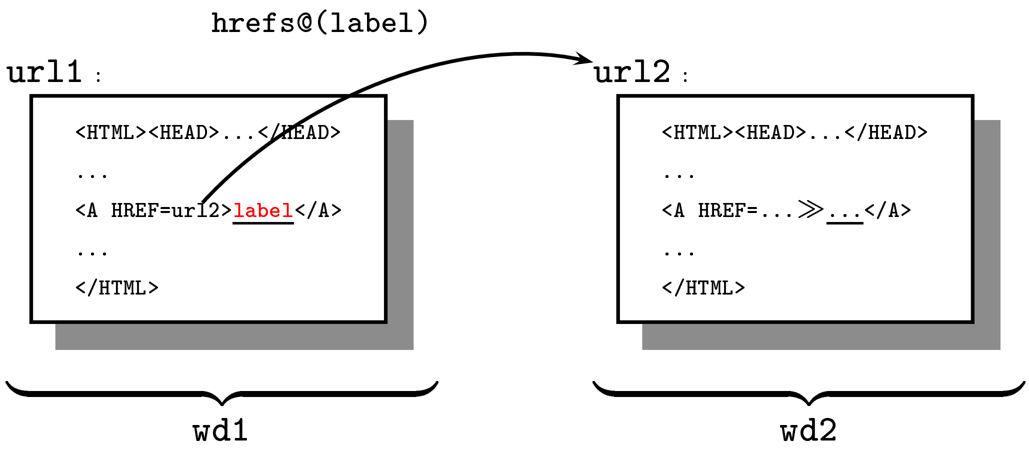
u.get contains "`<a href =` u' `>` ℓ `</a>`" .

```
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; ...]
```

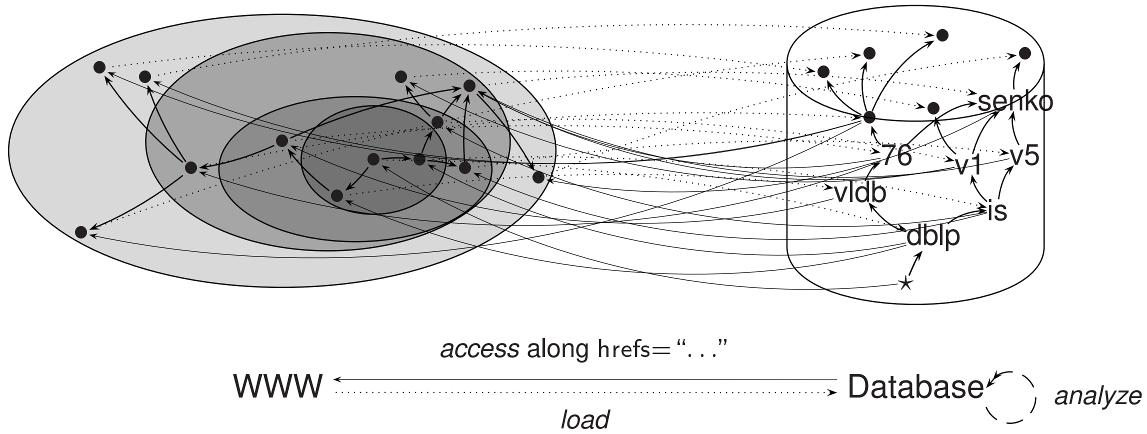
Data-Driven Web Exploration

- 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: active method
- 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 $o.<tag>$
  - 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
- 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-oriented 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
- 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

⇒ Generic Rules for these tasks

- 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

- Inter-Source Links

Conclusion

- practicable approach (multi-source MONDIAL case study).
- unified model for Web representation and model of the application,
- integrated data model/language for wrapping, mediating, and querying.
- Further Work:
  - “intelligent” wrapping (analyzing of tables)
  - Usage with search engines
  - XML

Appendix: 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 $\mathcal{H}$,

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

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