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

Wolfgang May[1] Rainer Himmeröder[1] Georg Lausen[1] Bertram Ludäscher[2]

Abstract. The goal of information extraction from the Web is to provide an integrated view on heterogeneous information sources via a common data model and query language. A main problem with current approaches is that they rely on very different formalisms and tools for wrappers and mediators, thus leading to an "impedance mismatch" between the wrapper and mediator level. In contrast, our approach integrates wrapping and mediation in a *unified framework* based on an object-oriented data model which represents both the Web structure and the data of the application domain. Wrappers *and* mediators are written in a rule-based object-oriented language which is augmented with features for Web access and structured document analysis, i.e., pattern matching by regular expressions and SGML parsing. In this paper, we develop generic, reusable rule patterns for typical extraction, integration, and restructuring tasks using this framework. We show the practicability of our approach by using the FLORID system [HKL+98].

1 Introduction

The Web is now the most popular information repository and there is a strong need for integration of data from different sources. Since Web sites can be seen as *autonomous* databases, there are many exceptions and pitfalls when wrapping such sources. Additionally, data representation can change without notice, and new sources can be added. Thus, a combination of rapid prototyping and refinement is highly desirable for Web data integration. Most approaches comprise *wrappers* for translating data from different local languages into a common format, and *mediators* for providing an integrated view on the data. Existing approaches have a strictly layered architecture and employ different languages for wrappers and mediators.

In [LHL+98], we showed that languages supporting deduction and object-orientation are particularly suited in this context and presented a formal model for querying structure and contents of Web data. *Object-orientation* provides a flexible and rich data model, e.g., for handling partial information. In our framework, a unified model of the Web representation and the application-level semantic representation is generated [May99b], yielding an integrated language for wrapper and mediator rules and for querying. This avoids the impedance mismatch due to separate languages. By *deductive rules*, the process of wrapping, mediating, and restructuring can be defined in a high-level, declarative programming style.

[1] Institut für Informatik, Universität Freiburg, Germany
{may,himmeroe,lausen}@informatik.uni-freiburg.de
[2] San Diego Supercomputing Center, ludaesch@sdsc.edu

In the present paper, we focus on the practical aspects by using the FLORID system [HKL+98], an implementation of the deductive object-oriented database language F-Logic [KLW95]. We present a systematic *methodology* for retrieving and integrating data from the Web by combining generic rule patterns with application-specific rules. The power and flexibility of generic rules can be fully exploited by the language which allows variables ranging also over objects and methods, and path expressions for navigating in the unified object-oriented model of both the Web fragment and the application-level representation.

The paper is structured as follows: We present the formal framework underlying our approach, including its Web model in Section 2. A methodology for extracting data from the Web by generic rule patterns for wrapping and mediating is proposed in Section 3. In Section 4, we show the practicability of our approach by combining generic rule patterns and application-specific rules for generating a database from several Web pages. Finally, we summarize our work and compare it to other approaches.

2 A Formal Framework for Web Access

By combining the rich modeling capabilities (objects, methods, class hierarchy, inheritance, signatures) of the object-oriented data model with the advantages of deductive database languages, F-Logic provides a suitable framework for modeling and handling Web information. The approach makes extensive use of the following facilities which are unique features of our integration approach:

- The power and flexibility of generic rule patterns depends on the usage of variables ranging over objects (including host objects), classes, and methods.
- Integration of information via *object fusion*.

At a short glance, the syntax is as follows (we omit inheritable methods and signature specifications; for the full syntax and semantics, the reader is referred to [KLW95]):

- The language is based on variables, constants, and *object constructors* from which *id-terms* are composed as usual. Id-terms are interpreted as elements of the universe. By convention, object constructors start with lowercase letters whereas variables start with uppercase ones. Ground id-terms play the role of *logical* object identifiers (*oids*).

In the sequel, let O, C, D, Q_i, S, S_i, Sc, and Mv stand for id-terms.

- An *is-a assertion* is an expression of the form $O : C$ (object O is a member of class C), or $C :: D$ (class C is a subclass of class D).
- The following are *object atoms*:
  - $O[Sc@(Q_1, \ldots, Q_k) \rightarrow S]$: applying the *single-valued* method Sc with arguments $Q_1, \ldots, Q_k$ to O results in S,
  - $O[Mv@(Q_1, \ldots, Q_k) \twoheadrightarrow \{S_1, \ldots, S_n\}]$: applying the *multi-valued* method Mv with arguments $Q_1, \ldots, Q_k$ to O results in some S_i.
- A *rule* is a logic rule $\mathsf{h} \leftarrow \mathsf{b}$ over F-Logic's atoms, i.e., is-a assertions and object atoms; a *program* is a set of rules.

Example 1. Below, a fragment of the database which is created by the application described in this paper is shown. For readability, we use mnemonic oid's of the form o_{name}.

o_{belg} : country[name→"Belgium"; car_code→"B"; capital→$o_{brussels}$;
total_area→30510; population→10170241; continent@(o_{eur})→100;
indep@(date)→"04 10 1830"; pop_growth→#0.33; gdp_total→197000;
adm_divs↠{$o_{p\_antwerp}$, o_{p_westfl},...}; main_cities↠{$o_{brussels}$, $o_{antwerp}$,...};
ethnicgroups@("Fleming")→55; ethnicgroups@("Walloon")→33;
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@(date)→"07 02 1992";
seat→$o_{brussels}$; members@("member")↠{o_{belg}, o_{france}, . . . };
members@("membership applicant")↠{$o_{hungary}$, $o_{slovakia}$, . . . }].

In addition to the basic F-Logic syntax, the FLORID system also supports *path expressions* in place of id-terms for navigating in the object-oriented model:

– The path expression O.M is *single-valued* and refers to the unique object S for which $O[M \rightarrow S]$ holds, whereas O..M is *multi-valued* and refers to every S_i such that $O[M \twoheadrightarrow \{S_i\}]$ holds.

Example 2 (Path Expressions). In our example,

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

The following query yields all names N of cities in Belgium:

?- C : country[name→"Belgium"], C..main_cities[name→N].

Since path expressions and F-Logic atoms may be arbitrarily nested, a concise and extremely flexible specification language for object properties is obtained.

Object Creation. Single-valued references can create *anonymous objects* when used in the *head* of rules. A rule of the form

O.M[<properties>] ← <body(O,M)>

creates an object x such that o[m→x] and x[<properties>] hold whenever <body(o,m)> is satisfied.

Negation-free F-Logic programs have a standard logic programming semantics [KLW95]. For programs with negation, FLORID allows *inflationary* and user-defined *stratified* semantics. The FLORID system[1] provides an implementation of F-Logic extended with the Web model described below.

The Web Model. The underlying object-oriented Web model [LHL+98] is based on the concepts of url and webdoc, which represent url's and Web documents, respectively. Every member u of class url provides a special method $u.\mathsf{get}$ which is implemented as an *active* method: When $u.\mathsf{get}$ occurs in the *head* of a rule, the Web document which is accessible via u is accessed, assigned to the newly created object $u.\mathsf{get}$, and made an instance of class webdoc. Typically, the document associated with a url contains *hyperlinks* of the form "<a href $= u' >\ell$ " to other url's which refer to further Web documents. The hyperlink structure (*Web skeleton*) is modeled by the built-in property webdoc.hrefs:

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

By $u.\mathsf{get}$, *"raw"* (i.e., uninterpreted) data from the Web is addressed. This handling reflects only the very basic properties of Web documents, without further exploiting the document type or structure. The contents of the retrieved Web pages has to be analyzed by *wrappers*, using the internal structure of HTML or XML Web documents. The active method $u.\mathsf{parse}$ generates the F-Logic representation of the parse-tree of an SGML document and assigns it to the object $u.\mathsf{parse} : \mathsf{parsetree}$. Together with the Web skeleton, the parse-trees provide a comprehensive model of the relevant Web fragment [May99b]. Via *path expressions*, the user can then navigate through the *extended Web skeleton*, i.e., the link and parse-tree structure of a Web fragment. To our knowledge, FLORID is the only system which is aware of the Web structure itself in its data model. Our approach implements a hybrid concept by embedding *data-driven wrapping* into a *warehouse* approach: An F-Logic database is generated by iteratively accessing Web documents, analyzing them, traversing relevant hyperlinks, etc.

3 Wrapping and Mediation by Generic Rule Patterns

In our approach, the construction of wrappers and mediators is based on *generic rule schemata* for standard tasks which can be complemented by application-specific rules and refinements for handling exceptional cases to obtain the required flexibility. For the wrapping task, depending on the structure of Web pages, *parser*-based, *matching*-based, or *combined* wrapper rules are preferable:

Structural Markup (tagged data): For logical markup such as lists or tables which is subject to a specific HTML "subgrammar", wrapping is based on $u.\mathsf{parse}$, which is an object-oriented representation of the *parse-tree*. For each SGML tag t, the class $u.\mathsf{parse}.t$ contains all structures of type t on u: e.g., $x : u.\mathsf{parse.ul}$ holds for all -lists x on the Web document. Additionally, every tag induces a method for navigation in a parse-tree; e.g., for $x : u.\mathsf{parse.ul}$, the expression $x.\mathsf{ul@}(n)$ yields the n-th element of the *u*nordered *l*ist x.

[1] available at `http://www.informatik.uni-freiburg.de/~dbis/florid/`.

We define generic patterns for decomposing lists into their items, and for analyzing tables by identifying column headers and table entries. The interpretation of the contents is then done by application-specific rules. The usage of parse-trees is illustrated in Sections 4.2 and 4.3.

Optical and Syntactical Markup (untagged data): If the structure is only partially tagged (paragraphs, line breaks, fonts, commas), the wrapper is based on pattern matching via regular expressions. Generic patterns are used, e.g., for decomposing "pseudo-lists" which are structured via emphasizing or bold face fonts, for commalists, and for name-value pairs (cf. Sec. 4.1 and 4.2). Here, matching with regular expressions is provided by the built-in predicate

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

which extracts substrings by Perl's regular expressions into Perl-variables and binds them to F-Logic variables $X_1, \ldots, X_n$ as specified by <fmt-list>.
Additionally, arbitrary predicates (e.g., for string processing) can be defined via FLORID's *Perl* interface: the predicate perl(<perl-routine-as-string>, X,Y) binds Y to the result(s) of calling <perl-routine-as-string> with X as argument.

Often, logical markup and textual formatting are mixed, leading to combined generic rules (cf. Section 4.2).

In contrast to wrapping, the mediation and integration task is usually dominated by application-specific rules. However there are still repeatingly occuring situations where generic rule patterns apply: Objects which result from different sources have to be merged. Similarly, entities can be represented by string-valued attributes in one source, and by objects in another; then, integration will derive a link to the corresponding object.

In general, these generic rule patterns are not sufficient for wrapping and mediating information from Web pages – due to exceptions in the representation and to the complexity inherent to the application semantics. The "skeleton" of the program is assembled from generic rule schemata for the above standard tasks. Then, in a rapid-prototyping process, the program is completed manually:

- Structural and syntactical exceptions are handled by refinements of generic rule patterns,
- application-specific rules interpret the outcome of the generic patterns in a model of the application domain.

The rule-based approach allows for modular design of wrappers and mediators.

4 Patterns in Practice in Florid

We illustrate our methodology by an excerpt of a larger case study which creates a geographical database [May99a]. The main data sources are the *CIA World Factbook* [CIA], which provides information about countries and political organizations, and *Global Statistics* [Glo], a collection of data about administrative divisions and cities. On the wrapper-level, an individual model for each of the data sources is obtained by instantiating generic patterns for wrapping Web documents. The mediation and integration part consists of fusing corresponding objects across sources and creating references between the partial models.

Unlike other systems, FLORID performs all tasks (data access, extraction, integration, and restructuring) in a single unified framework and thus overcomes limitations of the strictly layered approach. For instance, regular expressions for wrapping can be derived at run-time based on previously extracted data from other sources. To our knowledge, FLORID is the only system which allows such a "reflective" programming model. In the following, all instances of generic extraction patterns are labeled with $\boxed{\textsc{rule\_name}}$.

4.1 CIA World Factbook: Wrapping Country Pages

The CIA World Factbook provides political, economic, social, and some geographical information. The pages can be classified as formatted text (see Fig. 1), thus, the wrapper is based on pattern matching via regular expressions. The regular textual format of the pages allows for very generic and easily extendible patterns. This holds for single-valued properties (both numeric and string, such as *capital* and *population*) and for multi-valued properties like border countries, languages, etc., which are given by lists of pairs (*name,value*).

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.)

Fig. 1. Excerpt of the CIA World Factbook: Countries

First, we access the continent pages `europe:continent[url@(cia)->...]` using the built-in active method url.get:

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

Every continent page contains a list of links to the countries, these are automatically stored in the slots u.get.hrefs@(<label>). The country objects are defined with their urls, names, and continents, and the urls are accessed:

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

e.g., for Belgium, this rule derives
$o_{belgCIA}$:country[url@(cia)→"...";name@(cia)↠"Belgium";continent→o_{eur}] .

• **Extracting scalar properties by patterns.** On the CIA country pages, properties of countries are introduced by keywords. In a first step, several single-valued methods are defined using a generic pattern which consists of the method name to be defined, the regular expression which contains the keyword for extracting the answer, and the datatype of the answer:

```
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).
```

The patterns are applied to the country.url.get-objects, representing the Web pages. Note the use of variables ranging over host objects, methods, and classes:

```
Ctry[Method:Type -> X] :-                          TEXT PATTERNS
   p(country,Method, RegEx, Type),                   +REFINEMENT
   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.
```

Here, we see an instance of *refinement*: the (generic) main rule is only valid if it applies to a string or to a number given by digits – but some numbers are given "semi-textually", e.g., "42 million". This is covered by the refined rule. Here, e.g.,

$o_{belgCIA}$[capital→"Brussels"; indep→"4 October 1830";
total_area→30510; population→10170241]

is derived. Similarly, dates are extracted from strings by further matching:

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

yielding, e.g., $o_{belgCIA}$[indep@(date)→"04 10 1830"].

• **Extracting multi-valued complex properties by patterns.** Ethnic groups, religions, languages, and border countries are given by *comma-separated lists* of (*string,value,unit*) triples, introduced by a keyword; unit is, e.g., "%" or "km".

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

For every pattern, the raw commalist is extracted, e.g.,

$o_{belgCIA}$[borders : commalist→"France 620 km, Germany 167 km, ..."] .

Comments (between parentheses) are dropped by a call to the user-defined Perl-predicate dropcomments:

```
C[M:commalist->K] :-  commalist(M,S,_,_),                     COMMALISTS
 pmatch(C:country.url@(cia).get,S,"$1",L), perl(dropcomments,L,K).
```

The comma-lists are decomposed into multi-valued methods (since comments are dropped, e.g., *Muslim(Sunni)* and *Muslim(Shi'a)* both result in *Muslim*):

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

yielding $o_{belgCIA}$[borders↠{"France 620 km", "Germany 167 km", ...}].
In the next step, the values are extracted from the individual list elements. There is a refined rule which preliminarily assigns the null value to elements which contain no value (which is often the case for languages or religions):

```
C[M@(X)->>V] :-  commalist(M,_,S,_),                 DECOMPOSING PAIRS
  strcat("/([^0-9]*)\s* ([0-9\.]+)",S,T), strcat(T,"/", Pattern),
  pmatch(C:country..M, Pattern, ["$1","$2"], [X,V]).
C[M@(X)->>null] :-
  commalist(M,_,S,_), C:country[M->>X], not substr(S,X).
```

yielding $o_{belgCIA}$[borders@("France")↠620; borders@("Germany")↠167; ...].
The multi-valued methods are summed up using FLORID's sum aggregate:

```
C[M@(X)->V] :- commalist(M,_,_,string),                     SUMMATION
               V = sum{W [C,M,X]; C:country[M@(X)->>W]}, V > 0.
```

In a refinement step, non-quantified elements of unary lists are now set to 100%, other entries without percentages remain nulls; here, a count aggregate is used.

```
C[M@(X)->#100] :- commalist(M,_, S, string),            REFINEMENT:
    C:country..M@(X)[], not C.M@(X)[],                  EXCEPTIONS
    N = count{X [C,M]; C:country..M@(X)[]}, N = 1.   OF SUMMATION
C[M@(X)->null] :- commalist(M,_, S, string),
    C:country[M@(X)->>null], not C.M@(X)[],
    N = count{X [C,M]; C:country..M@(X)[]}, N > 1.
```

For country[borders@(country)→number], the country objects are used as arguments (M and Class are instantiated with borders and country, respectively).

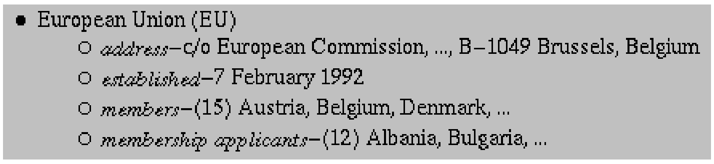
```
C[M@(C2)->V] :- C2:Class[name@(cia)->>X], commalist(M,_, S, Class),
         V = sum{Val [C,X]; C:Class[M@(X)->>Val]}, V > 0.
```

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

4.2 The CIA World Factbook: Wrapping Organizations

Appendix C of the CIA World Factbook provides information about political and economical organizations. The pages are tagged nested lists (see Fig. 2).

Thus, we exploit the structure of the parse-tree: For every organization, a list of properties is given. The list items are then queried by pattern matching, filling the respective methods of the organization. The members are again given as a set of comma-lists (distinguished by the type of membership) of countries.




```
<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>
```

Fig. 2. Excerpt of the CIA World Factbook: Appendix C: Organizations

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

For every letter, there is a list with organizations. The class u.parse.ul contains all lists on u – note that only upper-level lists qualify:

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

The org_lists (<UL>) are then analyzed by navigation in the parse-tree: The k-th element of the <UL> is an organization header, the k+1-th the corresponding data list. In the same step, the abbreviation and the full name are extracted from the header; the inner <UL> contains the properties of the organization:

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

derives, e.g.,

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

For every organization, several properties are listed, e.g., its address, its establishment date, and lists of members of the different member types. For every property, the corresponding x-th item of the (inner) <UL> (which itself is an <LI>) is identified by a keyword (which is <I>-ed in the 0-th item of the <LI>), e.g., "address" or "established", or a member type.

```
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.

Note that the rule is refined to skip the leading “-” and the number of members:

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

derives, e.g.,

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

Again, by an instance of the generic rule pattern (*), the date is formatted:

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

The address is analyzed by an application-specific matching rule:

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

derives, e.g.,

o_{eu}[seatcity$\rightarrow$"Brussels"; seatcountry$\rightarrow$"Belgium"; establ@(date)$\rightarrow$"07 02 1992"].

The commalists of the membertypes are again decomposed using the generic rule pattern (**) (remember that membertype :: commalist):

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

derives, e.g., o_{eu}[member_names:membertype$\twoheadrightarrow${"Austria", "Belgium", ...};
applicant_names:membertype$\twoheadrightarrow${"Albania", "Bulgaria", ...}] .

In a first *integration* step, the memberships are represented by references to the respective country objects:

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

derives, e.g., o_{eu}[member@("member")$\twoheadrightarrow$$\{o_{austriaCIA}, o_{belgCIA}, \ldots\}$].

4.3 Global Statistics: Wrapping Cities and Provinces

The *Global Statistics* [Glo] Pages provide information about administrative divisions (area, population, and capitals) and main cities (population and province). The data is given by HTML tables as shown at the right, but with an irregular structure of columns. Here, generic rules for analyzing tables are applied. Application-

administrative divisions of Belgium

province	capital	area (km²)	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

specific rules map the semantics of the columns to the object-oriented model. Additionally, layout specialties have to be dealt with, such as twocolumn layout and multilevel administrative divisions. After working through the continent pages for getting the country urls, these are accessed and parsed by the built-in SGML parser:

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

The class u.parse.table contains all tables of u. Based on the `<TR>` and `<TD>` structure, the tables are mapped to a matrix (by several refinements of the following rule):

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

Here, e.g., the administrative divisions table is the second element in the page body, i.e.,

o_{belgGS}.url@(gs).parse.body@(2) : o_{belgGS}.url@(gs).parse.table.
elem(o_{belgGS}.url@(gs).parse.body@(2),2,3)[cont→"area (km^2)";type→"th"]
elem(o_{belgGS}.url@(gs).parse.body@(2),4,3)[cont→"3,358";type→"td"] .

In a first step, information about column headers is extracted:

```
T[header_row->>HR] :- U:country_url,                             TABLES:
   T:(U.parse.table), elem(T,HR,_)[type->th].             COLUMN HEADERS
T[header@(HR,Col)->>String] :-
   U:country_url, T:(U.parse.table)[header_row->>HR],
   elem(T,HR,Col)[cont->String;type->th].
```

leading to, e.g.,

o_{belgGS}.url@(gs).parse.body@(2)[header_row↠{1};
header@(3,1)→"province";...;header@(3,4)→"pop. Dec.31,1991"].

Then, the tables are analyzed by application-specific methods, e.g., the table of *administrative divisions* and its columns are identified. For identification of a column, a set of keywords can be given:

```
C[adm_div_tab -> T:admdivtab[name_col->0]] :-
  C:country[url@(gs)->U], T:(U.parse.table), elem(T,_,_)[cont->I],
  pmatch(I,"/administrative divisions of (.*)/","$1",Name).
area_col:admdivtabcol[header->>{"area"}].
pop_col:admdivtabcol[header->>{"population"}].
cap_col:admdivtabcol[header->>{"capital","hauptstadt"}].
```

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

Here, o_{belgGS}[admdivtab→o_{belgGS}.url@(gs).parse.body@(2)], and
o_{belgGS}.admdivtab[name_col→1; area_col=fd 3; pop_col→4; cap_col→2].

Based on the header-entries, the data is extracted and transformed into an object-oriented representation:

```
C[adm_divs ->> prov(C,R):prov
      [country->C; name_str->PN; population->P; area->A]] :-
  C:country[adm_div_tab -> T[name_col->NS;area_col->AS;pop_col->PS]],
  elem(T,R,NS)[cont->PN;type->td], elem(T,R,AS)[cont->A;type->td],
  elem(T,R,PS)[cont->P;type->td].
```

If a *capital* column exists, it is also evaluated:

```
C[adm_divs ->> prov(C,R)[capital_str->CN]] :-
   C:country[adm_div_tab -> T[cap_col->CS]], prov(C,R):prov,
   elem(T,R,CS)[cont->CN;type->td].
```

deriving, e.g., $o_{belgGS}[\mathsf{adm\_divs} \twoheadrightarrow \{o_{p\_antwerp}\}]$ and

$o_{p\_antwerp}[\mathsf{name\_str} \rightarrow \text{"Antwerp"};\ \mathsf{population} \rightarrow 1610695;$
$\mathsf{area} \rightarrow 3358;\ \mathsf{capital\_str} \rightarrow \text{"Antwerp"}]$.

Analogously, the table of *main cities* is evaluated. Then, the references from cities to provinces, and the references from provinces to their capitals are generated:

```
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]].
```

4.4 Mediation

After transforming the contents of the Web pages into F-Logic models by wrapper rules, the integration and restructuring of the data in the object-oriented model can be performed.

The databases generated by the above wrapper rules are integrated into a common object-oriented schema. Here, the countries of CIA and *Global Statistics* have to be matched, either by name

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

(which would derive $o_{belgCIA} \equiv o_{belgGS}$) or (if different names are used) if they belong to the same continent and the capitals' names are the same:

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

For creating intersource links, generic rules using *meta knowledge* about the schema (keys, referential constraints) are employed. Here, we give only an application-specific rule which links seats of organizations (in CIA) to the respective cities (in GlobalStatistics):

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

deriving, e.g., $o_{eu}[\mathsf{seat} \rightarrow o_{brussels}]$.

The rules given are slightly simplified wrt. the original programs [May99a] to illustrate the principles of information extraction from the Web in our approach. In the full program, more exceptions and formatting issues have been faced: these lead to refined rules.

5 Conclusion and Related Work

Our approach for extracting and integrating information from the Web is based on (i) a unified object-oriented model of the Web and the application domain and (ii) an integrated rule-based language for wrapping and mediation which allows reuse of generic rule patterns for different applications. By instantiating generic rule patterns for handling logical markup and textual formatting and typical situations of data integration, a program skeleton is assembled which can be completed by application-specific rules and refining rules.

Existing approaches for information extraction employ different languages for wrappers and mediators. In the Araneus project [AMM97], different languages are used for extracting data and defining views on it in a hypertext-based model. Tsimmis [GMPQ+97] uses OEM (Object Exchange Model) as a common data model. Here, different languages are used for wrapping (WSL), mediation (MSL) and querying (MSL, Lorel). For a recent overview of systems, see [FLM98]. The above approaches use hand-coded wrappers. [HGC+97] describe a grammar-based tool for coding wrappers producing OEM output.

Jedi [HFA+98] is a tool for manually specifying wrappers for HTML pages by a framework combining grammars and rules. W4F [SA99] is a toolkit for interactively generating wrappers for HTML pages using HEL, a DOM-based language operating on the parse-tree of a document. HEL focusses on wrapping single Web pages; Web exploration and information integration is left to the application. Comma-lists and name-value pairs are regarded as atomic – i.e., the result is not completely mapped to the application domain, requiring a manually specified application-specific postprocessing. Another tool for interactive generation of wrappers using the DOM model is presented in *NoDoSE* [Ade98].

[AK97] present a grammar-based approach for semiautomatically wrapping HTML pages. Their approach does not support HTML tables or any textual formatting capabilities such as commalists or comma-pair-lists – which is a common way for structuring data on Web pages. Thus, the extraction of the data as shown in Section 4 is not possible with their wrappers.

[KWD97] present inductive learning methods for wrapping HTML pages with a tabular layout and report a 48% success. In [ECJ+98], higher than 90% success is reported for automatical matching-based wrapper generation for *data-rich and ontological narrow* sources.

Currently, (semi-)automatical approaches for wrapper-generation do not (yet) provide a sufficiently fine granularity for exhaustively extracting information from semistructured sources – especially if these use textual formatting. Indeed, given the many idiosyncrasies of real-world sources, it seems more viable to employ high-level declarative languages for rapid prototyping and refinement of wrappers. Here, a combination of automatically pre-generating wrapper skeletons which are then refined manually could be useful.

To our knowledge, for the above-mentioned tools, no complete case-study comprising wrapping and integration of several sites has been published. We claim that the case study [May99a] would not be possible with the above-mentioned (semi-)automatic approaches. Wrapping would be possible with the wrapping tools *Jedi* [HFA+98], *NoDoSE* [Ade98], and W4F [SA99], but these miss suitable integration functionality – which has then to be done in a different framework.

References

[Ade98] B. Adelberg. NoDoSE – A Tool for Semi-Automatically Extracting Semi-Structured Data from Text Documents. *ACM Intl. Conference on Management of Data (SIGMOD)*, pp. 283–294, 1998.

[AK97] N. Ashish and C. A. Knoblock. Wrapper Generation for Semi-structured Internet Sources. *SIGMOD Record*, 26(4):8–15, 1997.

[AMM97] P. Atzeni, G. Mecca, and P. Merialdo. To Weave the Web. *Intl. Conference on Very Large Data Bases (VLDB)*, 1997.

[CIA] CIA World Factbook. `http://www.odci.gov/cia/publications/nsolo/`.

[ECJ+98] D.W. Embley, D.M. Campbell, Y.S. Yiang, Y.-K. Ng, R.D. Smith. A Conceptual-Modeling Approach to Extracting Data from the Web. *Intl. Conf. on the Entity Relationship Approach (ER)*. Springer LNCS 1507, 1998.

[FLM98] D. Florescu, A. Levy, and A. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, 27(3), 1998.

[Glo] `http://home.worldonline.nl/~quark/index.html`.

[GMPQ+97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2), 1997.

[HGC+97] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting Semistructured Information from the Web. *WebDB '97: International Workshop on the Web and Databases.*, pp. 18–25, 1997.

[HKL+98] R. Himmeröder, P.-T. Kandzia, B. Ludäscher, W. May, and G. Lausen. Search, Analysis, and Integration of Web Documents: A Case Study with FLORID. *Intl. Workshop on Deductive Databases and Logic Programming (DDLP'98)*, pp. 47–57, 1998.

[HFA+98] G. Huck, P. Fankhauser, K. Aberer, E. J. Neuhold. Jedi: Extracting and Synthesizing Information from the Web. *Intl. Conference on Cooperative Information Systems (CoopIS'98)*, pp. 32–43, 1998.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.

[KWD97] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *IJCAI*, 1997.

[LHL+98] B. Ludäscher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective. *Information Systems*, 23(8):589–612, 1998.

[May99a] W. May. Information Extraction and Integration with FLORID: The MONDIAL Case Study. `http://www.informatik.uni-freiburg.de/~may/Mondial/`, 1999.

[May99b] W. May. Modeling and Querying Structure and Contents of the Web. To appear in *Workshop on Internet Data Modeling (IDM'99), Proc. DEXA'99*; IEEE Computer Science Press, 1999.

[SA99] A. Sahuguet, F. Azavant. Looking at the Web through XML glasses. To appear in *CoopIS'99*, IEEE Computer Science Press.