Filtering of XML Documents[*]

D. Ballis
Dip. Matematica e Informatica
Via delle Scienze 206,
33100 Udine, Italy.
demis@dimi.uniud.it

D. Romero
DSIC, Universidad Politécnica de Valencia
Camino de Vera s/n, Apdo. 22012,
46071 Valencia, Spain.
dromero@dsic.upv.es

Abstract

*In this paper, we present a simple, easy-to-use, rewriting-like methodology for filtering information in an XML document. Essentially, we define a specification language which allows one to extract relevant data (*positive filtering*) as well as to exclude useless and misleading contents (*negative filtering*) from a set of XML documents according to some given criteria. We believe that our methodology achieves the right tradeoff between expressive power and simplicity of use, and thus it may be also fruitfully employed by those users who typically prefer to avoid formal languages.*

1 Introduction

Internet users typically retrieve/receive a lot of information which should be absorbed in a pleasant and/or understandable fashion. Frequently, information flows are not of the desired quality, since data might appear obscure, difficult to interpret or with an unknown format. Besides, most of the time, just a small percentage of the whole amount of the data received is considered interesting by the user.

Filters are one of the possible methodologies which is employed to obtain Web contents which best fit user's needs. They allow one to extract useful data as well as to get rid of meaningless, incorrect information of a Web site. In other words, they are just as useful for selecting the good as they are in weeding out the bad. Arguably, simply selecting the good is a more productive and enjoyable process; although, sometimes, it is more convenient to just remove the unnecessary information. Additionally, filters can transform our information flows into more useful and palatable forms [12].

XML[18] is the format of interchange usually used to transmit information on internet with the aim of standardizing and facilitating data sharing across different systems. A lot of research work has been invested in XML document management, in particular XML filtering is undoubtedly one of the more relevant topic in such an area. The World Wide Web Consortium has defined XQuery[20] and XPath[19] as standard languages to consult and filter information in XML documents, nonetheless a plethora of alternative and worthwhile proposals have been developed independently, e.g. [7, 15, 8].

Our contribution. In this paper, we present a novel XML filtering language which allows the user to easily select the desired information (*positive filtering*) as well as to remove noisy, spurious data (*negative filtering*) from a given Web page. Our language is easy to use and thus can be employed even by those users who are typically not used to express themselves using formal methodologies, since no special expertise is required. We also believe that the proposed methodology is expressive enough to catch all those filtering features one may require.

Basically, in our approach, XML documents are encoded as Herbrand terms of a suitable term algebra, then *homeomorphic embedding* among terms [5, 14] is used to recognize the information that the user wants to select or to strike out. To detect information patterns inside an XML document, similar works make use of tree (or graph) *simulation*, which is a slightly different, less powerful, but efficient to compute notion of embedding. Simulation has been used in a number of works dealing with querying, transformation, and similarity checks of semistructured data (cf. [1, 10, 4, 6]). For instance, the language Xcerpt [6] is a (logic) query language for XML and semistructured documents which implements a sort of unification by exploiting the notion of simulation.

Plan of the paper. The rest of the paper is structured as follows. Section 2 summarizes some preliminary definitions and notations, we also formulate a simple method for translating XHTML/XML documents into Herbrand terms. In

[*] This work has been partially supported by the EU (FEDER) and Spanish MEC TIN-2004-7943-C04-02 project, the Generalitat Valenciana under grant GV06/285, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project. Daniel Romero is also supported by ALFA grant LERNet AML/19.0902/97/0666/II-0472-FA

2nd Int'l. Workshop on Automated
Specification and Verification of Web Systems (WWV'06)
0-7695-2826-0/06 $20.00 © 2006 IEEE

Section 3, we recall the notion of *homeomorphic embedding*, which we utilize to recognize patterns inside XML documents. In Section 4, we define our XML filtering language, while in Section 5, we formalize the positive as well as the negative filtering by means of the notion of *partial matching*. Section 6 concludes.

2 Preliminaries

We call a finite set of symbols *alphabet*. Given the alphabet A, A^* denotes the set of all finite sequences of elements over A. Syntactic equality between objects is represented by $\equiv$. By $\mathcal{V}$ we denote a countably infinite set of variables and Σ denotes a set of function symbols, or *signature*. We consider varyadic signatures as in [9] (i.e., signatures in which symbols have an unbounded arity, that is, they may be followed by an arbitrary number of arguments). $\tau(\Sigma, \mathcal{V})$ and $\tau(\Sigma)$ denote the *non-ground term algebra* and the *term algebra* built on $\Sigma \cup \mathcal{V}$ and Σ. Terms are viewed as labelled trees in the usual way. Positions are represented by sequences of natural numbers denoting an access path in a term. The empty sequence Λ denotes the root position. By notation $w_1.w_2$, we denote the concatenation of position w_1 and position w_2. Positions are ordered by the prefix ordering, that is, given the positions w_1, w_2, $w_1 \leq w_2$ if there exists a position x such that $w_1.x = w_2$. Given $S \subseteq \Sigma \cup \mathcal{V}$, $O_S(t)$ denotes the set of positions of a term t which are rooted by symbols in S. $t_{|u}$ is the subterm at the position u of t. $t[r]_u$ is the term t with the subterm rooted at the position u replaced by r. Given a term t, we say that t is *ground*, if no variables occur in t. Besides, t is a *terminal* term, if it is a 0-ary signature symbol (i.e. constants are terminal terms).

A *substitution* $\sigma \equiv \{X_1/t_1, X_2/t_2, \ldots\}$ is a mapping from the set of variables $\mathcal{V}$ into the set of terms $\tau(\Sigma, \mathcal{V})$ satisfying the following conditions: (i) $X_i \neq X_j$, whenever $i \neq j$, (ii) $X_i\sigma = t_i$, $i = 1, ..n$, and (iii) $X\sigma = X$, for any $X \in \mathcal{V} \setminus \{X_1, \ldots, X_n\}$. By $Var(s)$ we denote the set of variables occurring in the syntactic object s.

Term rewriting systems provide an adequate computational model for functional languages. In the sequel, we follow the standard framework of term rewriting (see [3, 13]). A *term rewriting system* (TRS for short) is a pair (Σ, R), where Σ is a signature and R is a finite set of reduction (or rewrite) rules of the form $\lambda \to \rho$, λ, $\rho \in \tau(\Sigma, \mathcal{V})$, $\lambda \notin \mathcal{V}$ and $Var(\rho) \subseteq Var(\lambda)$. We will often write just R instead of (Σ, R). A rewrite step is the application of a rewrite rule to an expression. A term s *rewrites* to a term t via $r \in R$, $s \to_r t$ (or $s \to_R t$), if there exist a position $u \in O_\Sigma(s)$, $r \equiv \lambda \to \rho$, and a substitution σ such that $s_{|u} \equiv \lambda\sigma$ and $t \equiv s[\rho\sigma]_u$. When no confusion can arise, we will omit any subscript (i.e. $s \to t$). A term s is a *irreducible form* (or *normal form*) w.r.t. R, if there is no term t s.t. $s \to_R t$. t is the irreducible form of s w.r.t. R (in symbols $s \to^!_R t$) if $s \to^*_R t$ and t is irreducible.

```
<books>
   <book>
      <title>El Alquimista</title>
      <author>Coelho</author>
      <year>2002</year>
   </book>
</books>

books(book(title(El Alquimista),
           author(Coelho),year(2002)))
```

Figure 1. An XML document and its corresponding encoding into a Herbrand term.

We say that a TRS R is *terminating*, if there exists no infinite rewrite sequence $t_1 \to_R t_2 \to_R \ldots$ A TRS R is *confluent* if, for all terms s, t_1, t_2, such that $s \to^*_R t_1$ and $s \to^*_R t_2$, there exists a term t s.t. $t_1 \to^*_R t$ and $t_2 \to^*_R t$. When R is terminating and confluent, it is called *canonical*. In canonical TRSs, each input term t can be univocally reduced to a unique *irreducible form*.

Let $s = t$ be an equation, we say that the equation $s = t$ *holds* in a canonical TRS R, if there exists an irreducible form $z \in \tau(\Sigma, \mathcal{V})$ w.r.t. R such that $s \to^!_R z$ and $t \to^!_R z$.

2.1 Denotation of XML Documents

In this work, we assume an *XML document* [18] to be well-formed, since there are plenty of programs and online services which are able to validate XML syntax and perform link checking (e.g. [21],[17]). XML documents can be encoded as Herbrand terms as follows.

Let us consider two alphabets T and $\mathcal{T}ag$. We denote the set T^* by $\mathcal{T}ext$. An object $\mathtt{t} \in \mathcal{T}ag$ is called *tag* element, while an element $\mathtt{w} \in \mathcal{T}ext$ is called *text* element. Since XML documents are provided with a tree-like structure, they can be straightforwardly translated into ordinary terms of a given term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag)$ as shown in Figure 1. Note that XML tag attributes can be considered as common tagged elements, and hence translated in the same way.

In the following, we will also consider terms of the non-ground term algebra $\tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$, which may contain variables. An element $s \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$ is called *XML document template*. In our methodology, XML document templates are used for specifying filtering criteria over XML documents as described in Section 4.

2nd Int'l. Workshop on Automated
Specification and Verification of Web Systems (WWV'06)
0-7695-2826-0/06 $20.00 © 2006 IEEE

3 Homeomorphic embedding

Roughly speaking, *homeomorphic embedding* allows us to verify whether a given XML document template is somehow "included" into another one. We give a definition of homeomorphic embedding, $\trianglelefteq$, which is an adaptation of the one proposed in [14], where (i) a distinct trreatment of the variables is considered, (ii) terms with different arity are allowed, and (iii) the positional ordering among the arguments of terms is ignored (i.e. $f(a, b)$ is "equivalent" to $f(b, a)$).

Definition 3.1 (homeomorphic embedding) *The* homeomorphic embedding *relation*

$$\trianglelefteq \subseteq \tau(\mathcal{T}\mathit{ext} \cup \mathcal{T}\mathit{ag}, \mathcal{V}) \times \tau(\mathcal{T}\mathit{ext} \cup \mathcal{T}\mathit{ag}, \mathcal{V})$$

on XML documents templates is the least relation satisfying the rules:

1. $X \trianglelefteq t$*, for all* $X \in \mathcal{V}$ *and* $t \in \tau(\mathcal{T}\mathit{ext} \cup \mathcal{T}\mathit{ag}, \mathcal{V})$.
2. $s \trianglelefteq f(t_1, \ldots, t_n)$*, if* $s \trianglelefteq t_i$ *for some* i.
3. $f(t_1, \ldots, t_m) \trianglelefteq g(s_1, \ldots, s_n)$ *iff* $f \equiv g$ *and* $t_i \trianglelefteq s_{\pi(i)}$*, for* $i = 1, \ldots, m$*, and some injective function* $\pi : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$.

Whenever $s \trianglelefteq t$, we say that t *embeds* s (or s *is embedded* or *recognized* into t).

The intuition behind the above definition is that $s \trianglelefteq t$ iff s can be obtained from t by striking out certain parts, in other words, the structure of s appears within t.

Note that, in Definition 3.1, when $\mathtt{m}$ is 0 we have $\mathtt{c} \trianglelefteq \mathtt{c}$ for each terminal term $\mathtt{c}$. Also observe that, if we constrain π to be an increasing function w.r.t. the usual ordering over natural numbers $\leq$, Definition 3.1 captures the notion of *ordered* homeomorphic embedding, where the ordering among the arguments of a term does matter. Typically, ordered homeomorphic embeddings can be computed faster [16].

Let us illustrate Definition 3.1 by means of a rather intuitive example.

Example 3.1 *Consider the following XML document templates (called* s *and* t*, respectively)*

```
(s) book(author(X))
(t) book(authorlist(author(name(Borges)),
                     author(Guerrero)),
      year(1957), code(BG57),
      title(Manual de zoología fantástica))
```

We observe that the structure of s *can be recognized inside the structure of* t*, hence* $s \trianglelefteq t$*, while* $t \ntrianglelefteq s$*. More precisely, there are* 3 *possible homeomorphic embeddings between* s *and* t*. In Figure 2, we show two of them: specifically, the ones which bind the variable* X *to some terminal term.*

4 XML filtering language

A *filtering specification* is a set of filtering rules. Basically, a *filtering rule* formalizes the information pattern (also called *filtering criterion*) which has to be detected inside a given XML document p. The information, which is recognized in p, is then selected (*positive* filtering) or removed (*negative* filtering), whenever a given *filtering condition* which refers to the detected instance of the filtering criterion is fulfilled.

We simply model filtering criteria as XML document templates.

A filtering condition consists of a sequence of equations over terms and membership tests (e.g. $X \in rexp$) w.r.t. a given regular language.[1]

More formally, a filtering criterion t belongs to $\tau(\mathcal{T}\mathit{ext} \cup \mathcal{T}\mathit{ag}, \mathcal{V})$. Besides, let (Σ, R) be a canonical TRS. A filtering condition C is a sequence of the form $C \equiv (X_1 \text{ in } rexp_1, \ldots, X_n \text{ in } rexp_n, s_1 = t_1 \ldots s_m = t_m)$, where $Var(C) \subseteq Var(t)$, t is a filtering criterion, $rexp_i$ is a regular expression over $\mathcal{T}\mathit{ext}$, $i = 1, \ldots, n$, and $s_j = t_j$ is an equation over $\tau(\Sigma, \mathcal{V})$, $j = 1, \ldots, m$.

Given a filtering condition

$$\begin{aligned} C \equiv \;& (X_1 \text{ in } rexp_1, \ldots, X_n \text{ in } rexp_n, \\ & s_1 = t_1 \ldots s_m = t_m) \end{aligned}$$

we say that C *holds* for substitution σ, if (i) each structured text $X_i\sigma$, $i = 1, \ldots, n$, is contained in the language of the corresponding regular expression $rexp_i$; (ii) each instantiated equation $(s_i{=}t_i)\sigma$, $i = 1, \ldots, m$, holds in R.

The TRS R contains the definition of some auxiliary functions which the user would like to provide in order to ease some common operations, such as string processing, arithmetic, boolean operators, etc. It is formalized as a term rewriting system, which is handled by standard rewriting [13].

In Figure 3, we present the complete grammar of our filtering language using a BNF-like notation. In the following, we describe the syntax and the features of the language through some intuitive examples. Note that some parts of the rules are not mandatory (e.g. conditions and labels). In particular, when no condition is provided for a filtering rule r, we say that r is *unconditional*.

4.0.1 Positive filtering

Intuitively, given an XML document p, a positive filtering rule extracts every instance $t\sigma$ of a given filtering criterion t from p. Then, if the associated condition C holds for σ, the result is delivered to the user. Positive filtering rules are

[1] Regular languages are represented by means of the usual Unix-like regular expressions syntax.

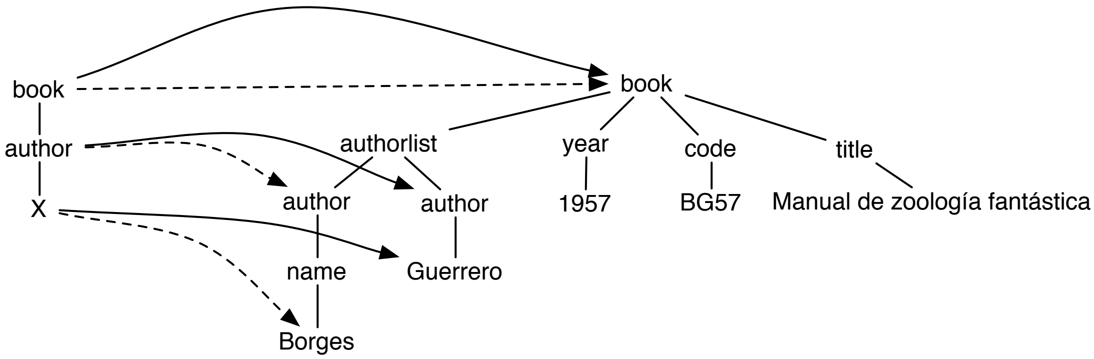

Figure 2. Example of homeomorphic embeddings

```
(1)  <filterRule> ::= <filter> <criterion> in
                <expression> [where <condition>]
                [label]
(2)  <filter> ::= filter | filter*
(3)  <criterion> ::= <xmlDocumentTemplate>
(4)  <expression> ::= <xmlDocument>|(<filterRule>)
(5)  <condition> ::= <sequence of membership tests
                      and equations>
(6)  <label> ::= (P) | (N)
```

Figure 3. BNF-like grammar for the XML filtering language

identified by means of the label `(P)`. However, when no label is provided, the rule is considered positive by default.

Example 4.1 *Consider the following filtering criterion and XML document:*

```
(t) book(title(X),author(name(Y),
           surname(Z)),code(W))
(p) book(title(El Alquimista),
           author(surname(Coelho),
                  name(Paulo)),
           year(2002),abstract(blablabla),
           code(PC))
```

along with a TRS R which models the usual string concatenation function ++ and the function `first`*, which returns the first character of a string. Then, the execution of the positive filtering rule*

```
filter book(title(X),author(name(Y),
            surname(Z)),code(W))
    in book(title(El Alquimista),
            author(surname(Coelho),
            name(Paulo)),year(2002),
            abstract(blablabla),
            code(PC))
    where X in [:Text:]Alquimista[:Text:],
             W = first(Y) ++ first(Z)
   (P)
```

will yield the following slice of the original XML document:

```
book(title(El Alquimista),
     author(surname(Coelho),name(Paulo)),
     code(PC))
```

4.0.2 Negative filtering

Given an XML document p, a negative filtering rule extracts every instance $t\sigma$ of a given filtering criterion t from p. Then, if the associated condition C holds for σ, $t\sigma$ is removed from p and the result is delivered to the user. Negative filtering rules are identified by means of the label `(N)`.

Example 4.2 *Consider the filtering criterion*

```
book(code(X),name(Y))
```

together with the XML document of Example 4.1. Then, the execution of the unconditional negative filtering rule

```
filter book(code(X),name(Y))
       in book(title(El Alquimista),
               author(surname(Coelho),
               name(Paulo)),year(2002),
               abstract(blablabla),
               code(PC))
       (N)
```

will produce the following slice of the original XML document:

```
book(title(El Alquimista),
     author(surname(Coelho)),
     abstract(blablabla),year(2002))
```

4.0.3 Unordered filtering vs ordered filtering.

In Section 3, we state that ordered homeomorphic embeddings are faster to compute than the unordered ones of Definition 3.1. When the user knows the structure of the XML document, is thus more convenient to use the notion of ordered embedding for filtering information. In our language we provide this distinction: the keyword `filter` allows us to implement the filtering mechanism without taking into account the positional ordering of the term arguments, while by means of the keyword `filter*` we force the system to consider such an ordering.

Example 4.3 *Consider the filtering criterion* `book(code(X),title(Y))` *together with the XML document of Example 4.1. Then, the execution of the unconditional, ordered, positive filtering rule*

```
filter* book(code(X),title(Y))
        in book(title(El Alquimista),
                author(surname(Coelho),
                name(Paulo)),year(2002),
                abstract(blablabla),
                code(PC))
```

would not produce any result, while the unordered version would select the code as well as the title of the given book.

4.0.4 Nested filtering.

Our filtering language allows the user to specify complex, compound filtering statements by combining simpler filtering rules through nesting (see the grammar rules `(1)` and `(4)` of Figure 3). We call such rules *compound* filtering rules.

Example 4.4 *Consider the XML document of Example 4.1. The execution of the compound filtering rule*

```
filter name(X) in
 (
  filter book(title(X),author(name(Y),
              surname(Z)),code(W))
    in book(title(El Alquimista),
            author(surname(Coelho),
            name(Paulo)), year(2002),
            abstract(blablabla),code(PC))
    where X in [:Text:]Alquimista[:Text:],
             W = first(Y) ++ first(Z)
   (P)
 )
(N)
```

yields

```
book(title(El Alquimista),
     author(surname(Coelho)),code(PC))
```

In our methodology, *filtering* is carried out by running filtering rules on XML documents. This is mechanized by means of *partial matching*, a technique based on homeomorphic embedding, which is described in the following section.

5 Filtering using Partial Matching

Given an XML document p and an XML document template t, partial matching exploits homeomorphic embedding to recognize instances of t in p. A formal definition follows.

Definition 5.1 (partial matching) *Let* $s_1,\ s_2 \in \tau(\mathcal{T}ext \cup \mathcal{T}ag, \mathcal{V})$. *We say that* s_2 partially matches s_1 *via substitution* σ *iff*

1. $s_1 \trianglelefteq s_2$;
2. *for each* $X \in \mathcal{V}$ *such that* $X \in Var(s_1)$, $X\sigma = s_{2|w}$ *for some* $w \in O_{\mathcal{T}ag}(s_2)$, $X\sigma$ *is a terminal term, and* $s_1\sigma \trianglelefteq s_2$.

Definition 5.1 extends the notion of partial matching via *tree simulation* we formalized in [2] in order to deal with the more powerful notion of homeomorphic embedding.

Let us see an example which illustrates the definition above.

Example 5.1 *Consider the XML document template and the XML document (called* s *and* p*, respectively) of Figure 2. Then, we have that* p *partially matches* s *via* $\{X/Borges\}$ *and* p *partially matches* s *via* $\{X/Guerrero\}$*. Note that there is also a third homeomorphic embedding between* s *and* t*: specifically, the one which binds variable* X *to the term* $name(Borges)$*. However,* p *does not partially match* s *via* $\{X/name(Borges)\}$*, since* $name(Borges)$ *is not a terminal term and hence point* 2 *of Definition 5.1 is not satisfied in this case.*

The semantics of the positive as well as the negative filtering rules is based on the partial matching relation, as we can see in the following sections.

5.1 Positive filtering

Essentially, given a positive filtering rule *filter* t *in* p *where* C (P), the main idea is to compute the set of all substitutions $\{\sigma_1, \ldots, \sigma_n\}$ such that t partially matches s via $\sigma_i, i = 1, \ldots n$. Clearly, each instance $t\sigma_i$ is embedded in p

by Definition 3.1. At this point, we glue together each $t\sigma_i$, such that the corresponding condition C holds for σ_i, into a single XML document, and finally we deliver the resulting slice of p. Summing up, the method works in two steps:

1. Compute the set of substitutions σ_i, such that p partially matches s via σ_i.

2. Assemble the new document.

5.1.1 Step 1.

We define the following operator $\mathcal{R}$ which applies to a given filtering rule *filter t in p where* C (L), where $L \in \{P, N\}$, and allows to compute all the required $t\sigma_i$.

$$\mathcal{R}(\text{filter } t \text{ in } p \text{ where } C\ (L)) = \{t\sigma_i \mid p \text{ partially matches } t \text{ via } \sigma_i, C \text{ holds for } \sigma_i\}$$

Example 5.2 *Consider the XML document*

```
books(book(title(El Alquimista),
           author(Coelho),year(2002))
      book(title(Ficciones),
           author(Borges),year(1944)))
```

together with the filtering criterion

```
books(title(X),year(Y))
```

Then, if we apply the operator $\mathcal{R}$ to the following filtering rule r

```
filter books(title(X),year(Y))
    in books(book(title(El Alquimista),
                  author(Coelho),
                  year(2002))
             book(title(Ficciones),
                  author(Borges),
                  year(1944)))
    (P)
```

we get the following set $\mathcal{R}(r)$

```
{books(title(El Alquimista), year(2002)),
 books(title(Ficciones),year(1944))}
```

5.1.2 Step 2.

Data obtained by applying the operator $\mathcal{R}$ to a given positive filtering rule r are subsequently assembled into a single XML document in the following way.

Let $result \in \mathcal{T}ag$ be a fresh symbol which does not appear in any $t\sigma \in \mathcal{R}(r)$. Then, the execution of r yields the outcome

$$result(t\sigma_1, \ldots, t\sigma_n) \quad \text{where } t\sigma_i \in \mathcal{R}(r),\ i = 1, \ldots, n$$

In other words, we create a new XML document containing the results of the execution of the given positive filtering rule r.

Example 5.3 *Consider the result of the operator $\mathcal{R}$ of Example 5.2. Then, we obtain the final outcome:*

```
result(books(title(El Alquimista),
             year(2002))
       books(title(Ficciones),
             year(1944)))
```

which corresponds to the following XML document

```
<result>
   <books>
      <title>El Alquimista</title>
      <year>2002</year>
   </books>
   <books>
      <title>Ficciones</title>
      <year>1944</year>
   </books>
</result>
```

5.2 Negative filtering

Given a negative filtering rule *filter* t *in* p *where* C (N), we aim at removing each instance $t\sigma$ of t which is embedded in p whenever the associated condition C holds for σ. For this purpose, we introduce the auxiliary function $clean(p_1, p_2)$ which deletes the embedded term p_2 from the entire XML document p_1 and returns the modified document. For instance, $clean(f(g(a, c), a, h(a), d), f(a))$ will return the XML document $f(g(c), h, d)$. Such a function can be easily and efficiently implemented by applying a bottom-up algorithm, which starts removing the p_2's embedded subterms from the leaves of p_1 and proceeds recursively towards the root.

By using the $clean$ function and the operator $\mathcal{R}$, defined in Section 5.1, giving a semantics for the negative filtering rules becomes straightforward. More formally, the execution of a negative filtering rule $r \equiv$ *filter* t *in* p *where* C (N) yields the following XML document as outcome:

$$clean(clean(\ldots(clean(p, t\sigma_1), t\sigma_2)\ldots), t\sigma_n)$$

where $\mathcal{R}(r) = \{t\sigma_1, \ldots, t\sigma_n\}$.

Example 5.4 *Consider the negative filtering rule r*

```
filter book(code(X))
   in books(book(title(El Alquimista),
                 author(Coelho),
                 code(C2002), year(2002))
            book(title(Ficciones),
                 author(Borges),
                 codes(code(B44),
                       ISBN(9514280)),
                 year(1944)))
  (N)
```

Then,

$\mathcal{R}(r) =$ {book(code(C2002)),book(code(B44))},

and the final outcome is

```
 books(book(title(El Alquimista),
            author(Coelho),year(2002))
       book(title(Ficciones),
            author(Borges),
            codes(ISBN(9514280)),
            year(1944)))
```

which corresponds to the XML document

```
<result>
   <book>
      <title> El Alquimista </title>
      <author> Cohelo </author>
      <year> 2002 </year>
   </book>
   <book>
      <title> Ficciones </title>
      <author> Borges </author>
      <codes>
         <isbn> 9514280 </isbn>
      </codes>
      <year> 1944 </year>
   </book>
</result>
```

6 Conclusions

The growing complexity of the World Wide Web demands for tools which are able to tame the so-called information overload. To this respect, filters allow one to extract relevant and meaningful information within the enormous amount of data available on the Web. In this paper, we presented an XML filtering language which has several advantages w.r.t. other approaches. On the one hand, it is easy to use, since it has a simple syntax and a clear, intuitive semantics; on the other hand, it provides the expressive power of functions (modeled as Term Rewriting Systems) and a sophisticated mechanism for detecting information patterns which is based on the notion of homeomorphic embedding. By defining suitable rules, the user can easily establish filtering criteria which are subsequently employed to automatically select the needed information as well as to weed out unwanted contents from a collection of XML documents.

Finally, let us conclude by mentioning some directions for future work. We are currently working on a system which implements the proposed methodology. To increase efficiency, we are taking into account several approximations of the relation of homeomorphic embedding with different degrees of accuracy and time complexity [16]. Moreover, we are also planning to develop a compiler which allows us to translate filtering specifications into XPath queries to take advantage of the efficient XPath implementations [11] which are available nowadays.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufmann, 2000.

[2] M. Alpuente, D. Ballis, and M. Falaschi. Rule-based Verification of Web sites. *Int'l Journal on Software Tools for Technology Transfer*, 8:565–585, 2006.

[3] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[4] E. Bertino, M. Mesiti, and G. Guerrin. A Matching Algorithm for Measuring the Structural Similarity between an XML Document and a DTD and its Applications. *Information Systems*, 29(1):23–46, 2004.

[5] M. Bezem. *TeReSe, Term Rewriting Systems*, chapter Mathematical background (Appendix A). Cambridge University Press, 2003.

[6] F. Bry and S. Schaffert. Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In *Proc. of the Int'l Conference on Logic Programming (ICLP'02)*, volume 2401 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[7] F. Bry and S. Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. Technical report, 2002. Available at: `http://www.xcerpt.org`.

[8] A. Cortesi, A. Dovier, E. Quintarelli, and L. Tanca. Operational and Abstract Semantics of a Graphical Query Language. *Theoretical Computer Science*, 275:521–560, 2002.

[9] N. Dershowitz and D. Plaisted. Rewriting. *Handbook of Automated Reasoning*, 1:535–610, 2001.

[10] M. F. Fernandez and D. Suciu. Optimizing Regular Path Expressions Using Graph Schemas. In *Proc. of Int'l Conference on Data Engineering (ICDE'98)*, pages 14–23, 1998.

[11] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *Proc. of the 28th International Conference on Very Large Data Bases (VLDB'02), Hong Kong, China*, 2002.

[12] B. Gross. Information Filtering, 2006. Available at: `http://bengross.com/filter.html`.

[13] J. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I, pages 1–112. Oxford University Press, 1992.

[14] M. Leuschel. Homeomorphic Embedding for Online Termination of Symbolic Methods. In T. Æ. Mogensen, D. A. Schmidt, and I. H. Sudborough, editors, *The Essence of*

Computation, volume 2566 of *Lecture Notes in Computer Science*, pages 379–403. Springer, 2002.

[15] W. May. XPath-Logic and XPathLog: A Logic-Programming Style XML Data Manipulation Language. *Theory and Practice of Logic Programming*, 2004.

[16] T. Schlieder and F. Naumann. Approximate Tree Embedding for Querying XML Data. In *Proc. of ACM SIGIR Workshop on XML and Information Retrieval, Athens, Greece*, 2000.

[17] T. und Wicke GbR. Validate/Check XML. Available at: `http://www.xmlvalidation.com/`.

[18] W. Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0, second edition, 1999. Available at: `http://www.w3.org`.

[19] W. Wide Web Consortium (W3C). XML Path Language (XPath), 1999. Available at: `http://www.w3.org`.

[20] W. Wide Web Consortium (W3C). XQuery: A Query Language for XML, 2001. Available at: `http://www.w3.org`.

[21] W. Wide Web Consortium (W3C). Validator for XML Schema, 2005. Available at: `http://www.w3.org/2001/03/webdata/xsv`.

2nd Int'l. Workshop on Automated
Specification and Verification of Web Systems (WWV'06)
0-7695-2826-0/06 $20.00 © 2006 IEEE