# Di$\mathcal{X}$eminator: a Profile-based Selective Dissemination System for XML Documents

*Elisa Bertino*[1] *Giovanna Guerrini*[2] *Marco Mesiti*[3]

[1] Computer Sciences Department
Purdue University, USA
bertino@cs.purdue.edu
[2] Dipartimento di Informatica
Università di Pisa, Italy
guerrini@disi.unige.it
[3] Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano, Italy
mesiti@dico.unimi.it

**Abstract.** Current approaches for the selective dissemination of XML documents are not suitable for an automatic adaptation of user profiles to her current preferences because either they rely on user preferences specified by filling up forms or they require to process a high number of documents. In this paper we present the architecture of Di$\mathcal{X}$eminator, a selective dissemination system for XML documents based on profiles. Profiles, represented through XML Schema, concisely represent the kind of documents a user subscribing the service is interested in. Profiles are used for filtering out irrelevant documents relying on user preferences. Moreover, profiles are kept up to date taking into account the documents the user effectively accesses or refuses.

## 1 Introduction

As the amount of XML data available on the Web and the number of pervasive applications making use of these data increase, systems that support *selective dissemination of information* (*SDI systems*) are more and more popular [1, 4, 8, 10, 13]. A selective dissemination system manages user profiles as well as streams of incoming documents. For each incoming document, the system searches for the set of user profiles that match it in order to identify the users to whom the document should be broadcasted. Users can set their preferences when they connect the first time to the system (by filling up a form) or the preferences can be dynamically discovered by monitoring the documents users frequently access. A key capability of an SDI system is the effective filtering of a continuous stream of XML documents according to user preferences. Another key capability is the adaptability of user profiles to new preferences. It is not reasonable, indeed, to assume that user preferences do not change.

In this paper we present the architecture of Di$\mathcal{X}$eminator, a selective dissemination system for XML documents based on *user profiles*. Our system receives a continuous stream of XML documents and, by matching them against the user profiles, filters out the users that are not interested in the documents. Then, documents are broadcasted only

to interested users. Moreover, the system collects user feedbacks to keep the profiles up to date.

A key characteristic of Di$\mathcal{X}$eminator is that user profiles are modeled as XML Schemas [12]. By means of an XML Schema it is possible to concisely represent the set of documents relevant for a user. A document, which is *valid* with respect to the XML Schema, perfectly adheres to the conditions the user specifies to select the documents she is interested in. Moreover, a user profile can specify constraints on the values of data content elements. Since users can be interested in documents of different types, a user profile is often specified as a set of subschemas, each one of them representing a type of documents. Figure 1 reports an example of profile. The profile contains two subschema. The first one represents documents dealing with books, whereas the second one represents documents containing Sigmod record publications.

The user profile can initially be specified by the user or automatically inferred from documents the user previously deemed valuable, by means of document clustering and schema extraction techniques [7, 10]. Actually, the two approaches can be combined in order to obtain more coincise representations. Schema extraction techniques can be adopted to identify patterns/templates of documents and clustering techniques can be employed to group together similar patterns/templates. Finally, from each group an XML Schema can be determined that concisely represents the documents from which it has been extracted.

Incoming streams of documents are matched against the user profiles in order to establish the users to whom the documents should be broadcasted. Whenever a user accepts the document, because she locally stores the received document or sends a positive feedback to Di$\mathcal{X}$eminator, or rejects it, because she discards the received document or sends a negative feedback, structural and content information are extracted from the document and exploited for enhancing the user profile. Moreover, Di$\mathcal{X}$eminator monitors the Web navigation of its users in order to determine the new kinds of documents they are interested in. This information, gathered by the user feedbacks and the monitors, is used for updating the user profiles accordingly.

The presence of a huge amount of users, together with a high number of kinds of documents they could be interested in, introduces however scalability issues. Therefore, a mechanism for grouping together users interested in the same kind of documents is required along with query capabilities for efficiently selecting only the user profiles that can effectively match the document to be broadcasted. Di$\mathcal{X}$eminator addresses these issues by introducing the concept of *profile types* and their hierarchical organization that allows us to easily identify the users that are interested in the same sets of documents.

Di$\mathcal{X}$eminator takes advantage of previous work we have carried out for estimating the structural similarity between an XML document and a DTD [2]. The measure is employed for computing the *degree of relevance* of the document with respect to the DTD. Moreover, it takes also advantage of our work on the evolution of DTD structure relying on documents classified against it [3, 5]. Di$\mathcal{X}$eminator takes finally advantage of other approaches developed for the selective dissemination of text documents [1, 4, 8, 10, 13]. The most similar approach is the one proposed by Stanoi et al. [10]. Such an approach integrates profile inference with data dissemination and exploits the structured content of XML documents. Profiles are inferred by clustering documents previously

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="BookStore">
  <xsd:complexType>
  <xsd:sequence>
     <xsd:element name="Book" maxOccurs="unbounded">
     <xsd:complexType>
     <xsd:sequence>
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Author" type="myAuthor" maxOccurs="5"/>
        <xsd:element name="Date" type="xsd:string"/>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string" minOccurs="0"/>
     </xsd:sequence>
     </xsd:complexType>
     </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:element name="SigmodRecord">
  <xsd:complexType>
  <xsd:sequence>
     <xsd:element name="issue" minOccurs="0" maxOccurs="unbounded">
     <xsd:complexType> <xsd:sequence>
        <xsd:element name="volume" type="xsd:string"/>
        <xsd:element name="number" type="xsd:string"/>
        <xsd:element name="article" maxOccurs="unbounded">
        <xsd:complexType> <xsd:sequence>
           <xsd:element name="title" type="xsd:string"/>
           <xsd:element name="initPage" type="xsd:string"/>
           <xsd:element name="lastPage" type="xsd:string"/>
        </xsd:sequence> </xsd:complexType>
        </xsd:element>
     </xsd:sequence> </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
 <xsd:simpleType name="myAthor">
  <xsd:restriction base="xsd:string">
   <xsd:enumeration value="Serge Abiteboul"/>
   <xsd:enumeration value="Stefano Ceri"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:schema>
```

**Figure 1:** An example of profile

deemed valuable by the user according to a similarity measure that takes into account document structure. However, the similarity is accepted and measured at nodes, and then weights combined taking the structure into account. The document structures are required to coincide exactly, and no partial structural matches are allowed. By contrast, our approach uses an intensional representation of the documents the user is interested in. Moreover, it supports both approximate structural and content matches, and uses a hierarchy of profile types in order to minimize the number of matches to be executed.

The remainder of this paper is organized as follows. Next section introduces the representation of documents, profiles and profile types. Section 3 deals with the main components of the architecture of Di$\mathcal{X}$eminator. Section 4 concludes the paper and outlines future research directions.

## 2 Documents, Profiles and Profile Types

In this section we briefly discuss our tree representation for documents and profiles. Then, we introduce profile types and their hierarchical organization. This hierarchical organization is really relevant in order to face the scalability issues of Di$\mathcal{X}$eminator.

### 2.1 Documents and Profiles

Documents and profiles are represented as labeled trees. The tree representation is exploited when matching a document against a profile and for the evolution of profile structures. Formal definitions and properties of this representation can be found in [2, 5]. In the tree representation of a document, internal nodes represent elements, whereas leaves represent data content elements.

Figure 1 shows an example of profile. Note that, the profile contains the specification of two kinds of documents: `bookStore` and `SigmodRecord` documents. In the tree representation of a profile, internal nodes represent either element types, operators used for binding together elements (sequence, all, choice, ecc..) or operators specifying whether elements or groups of elements are optional or repeatable. Specifically, the operators that can be used for labeling internal nodes of a profile are {`AND`, `ALL`, `OR`, `[n,m]`}. The `AND` operator represents a sequence of elements, the `ALL` operator represents a set of elements for which the order is not relevant, the `OR` operator represents an alternative of elements (exactly one of the alternatives must be selected). The notation `[n,m]` is used for representing the number of repetitions or the optionality of elements. In particular, `[0,1]` denotes optional elements, `[0,n]` denotes repeatable optional elements, `[1,n]` denotes repeatable mandatory elements (that is, at least an occurrence should be present). Leaves of a profile, by contrast, represent basic types (integer, string, boolean,...) or specific constraints on the content of data content elements. The possibility XML Schema offers for restricting the set of valid values for data content elements by means of *facets* [12] is exploited in Di$\mathcal{X}$eminator for specifying simple constraints on the profile about the documents the user is interested in.

*Example 1.* Consider the profile in Figure 1. The profile specifies that the user is interested in any sigmod record publication and only in the books which author is either Serge Abiteboul or Stefano Ceri. As future work, we plan to introduce new kinds of constraints for specifying typical information retrieval operators. △

### 2.2 Hierarchy of Profile Types

Consider the following situation. Two XML Schemas can be used for representing the same kind of information but they do not have the same structure. Therefore, when we have to broadcast a document of such a kind we need to match it against the two schemas. Consider also the situation in which two users present a profile with the same structure. The same degree of relevance is computed for the two users.

In order to deal with the previous situations (that is, group together XML Schemas representing the same kind of information and apply the filtering algorithm only once on the same profile) we introduce *profile types*. A profile type is a triple $(id, profiles, users)$.

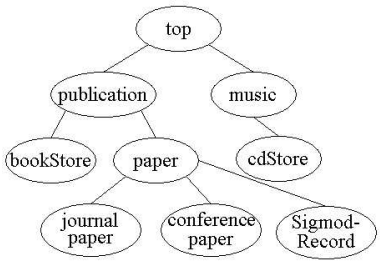



**Figure** 2: Hierarchy of profile types

The *id* component is a meta level description of the profile type. In the current stage of development it corresponds to the name of the outermost element of an XML Schema (e.g. `BookStore`, `cdStore` considering the profile in Figure 1).[1] The $profiles$ component is the set of profiles that belong to the same description. For example, if we wish to model the profile type `cdStore`, the *profile* component contains the XML Schemas that are used for modeling CDs. Finally, the *users* component is a set of Di$\mathcal{X}$eminator users.

Profile types are organized in a hierarchy. Given two profile types, $T_1 = (id_1, P_1, U_1)$ and $T_2 = (id_2, P_2, U_2)$, $T_2$ is a subtype of $T_1$, if and only if $id_2$ is considered as a more specific meta level description of a profile type than $id_1$, and $U_2 \subseteq U_1$. A dummy root node, called `top`, is present. The set of users associated with `top` is the entire set of users subscribing Di$\mathcal{X}$eminator.

*Example 2.* Figure 2 shows an example of hierarchy. Suppose the profile in Figure 1 is associated with user Bob. Since Bob is interested in `bookStore` and `SigmodRecord` profile types, his identifier is contained in the corresponding nodes of the hierarchy and in all the ancestor nodes (till the root `top`). △

## 3 Di$\mathcal{X}$eminator Architecture

Figure 3 reports the main components of the architecture of Di$\mathcal{X}$eminator. Rectangles denote the main functional components of the system, cylinders denote data stores, thick arrows denote the profile flow. Document flow is represented by means of the big arrow that pass through the filtering component. We remark that documents are considered only once by the filtering component. This component is in charge of determining the users interested in a document and of extracting structural and content information that will be exploited in the following phases of profile evolution. The information that is exchanged among components is thus a summary of the document structure and content. This is very important because it makes the system more efficient. In the remainder of the section we provide details of the main components of the architecture.

[1] In a future development the meta level description could be an RDF [11] description that allows us to better describe profiles.

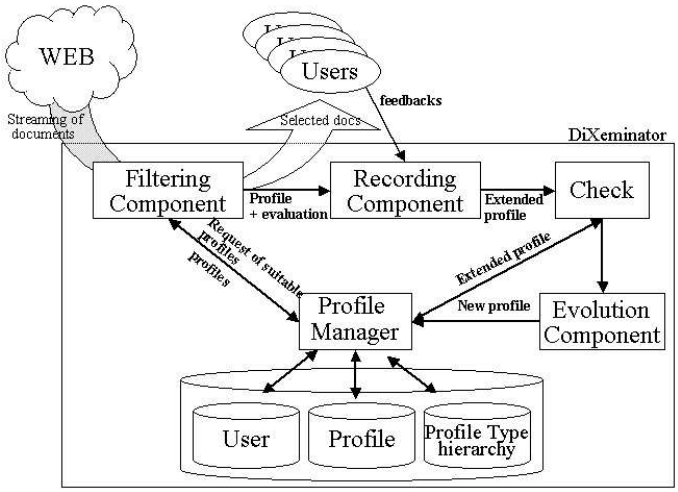



**Figure** 3: Architecture of Di$\mathcal{X}$eminator

### 3.1 Filtering Component

The *filtering* component is the most important component of the system because it is in charge of processing the incoming stream of documents and of broadcasting them to the interested users. This is the only component that works on line, thus it should perform its tasks efficiently.

Figure 4 reports the $Filter$ procedure, which is the main procedure of this component. Given a document $d$ this procedure selects the profiles and broadcasts them to the interested users. Such a procedure exploits different functions. The *ProfileFor* function, which belongs to the ProfileManager component, selects profiles that, once matched against $d$, could return a positive degree of relevance. The function compares the tag $t$ of the root of $d$ with the profile types contained in the hierarchy. The comparison is not a simple syntactic match, but we consider the possibility that $t$ is syntactically (relying on an edit distance function [9]) or semantically (relying on a Thesaurus [6]) similar to the description of the profile type. For all the identified matches the corresponding profiles are grouped together and returned.

The $Match$ function evaluates the relevance degree of document $d$ with respect to a profile $p_i$ (one of those profiles returned by function $ProfileFor$). This function evaluates the degree of relevance of the document with respect to the profile by evaluating the similarity between the two structures. The value $\mathcal{S}_D$ that function $Match$ returns is the ratio between the common features and the common and divergent features identified between the two structures. This evaluation takes into account both the level in which the common and divergent features are detected and the possibility that tags are syntactically or semantically similar. Details of function $Match$ can be found in [2, 5].

Moreover, function $Match$ returns the profile $\bar{p}_i$ which is the profile $p_i$ with some structural and content information extracted from $d$. If the degree of relevance is above

```
Procedure Filter(d : XMLdocument)
begin
    {p_1, ..., p_n} = ProfileManager.ProfileFor(d);
    for i = 1 to n do
    begin
        let (S_D, p̄_i) = Match(d, p_i); /*degree of relevance of d against p_i*/
        if (S_D > σ) begin
            {u_1, ..., u_m} = ProfileManager.UserFor(p_i);
            broadcast(d, {u_1, ..., u_m});
            RecordingComponent.FeedBackWaitingQueue(p̄_i);
        end
    end
end
```

**Figure 4:** The $Filter$ procedure

the fixed threshold, it means that there are users interested in this document, otherwise the document is discarded because it is irrelevant for the user. This threshold, which is a number between 0 and 1, is specified by the user and indicates the strength of desired similarity between her profile and the document to be delivered.

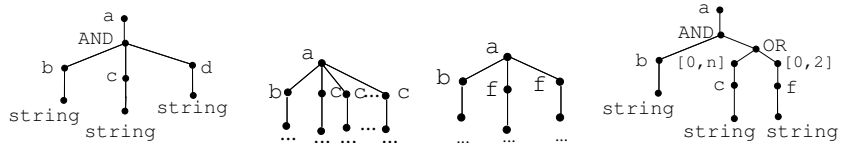
By means of the $UserFor$ function, the users presenting the profile $p_i$ are detected and the document is distributed to them. The $feedbackWaitingQueue$ function is then invoked on profile $\bar{p}_i$. This function inserts $\bar{p}_i$ in a queue waiting for a user feedback. The feedback along with the extracted information are used for subsequent evolutions.

### 3.2 Recording Component

The *recording* component is in charge of receiving a profile with the extracted data from a document $d$ and making it persistent when the user sends a positive feedback. The profile is, thus, extended with auxiliary data structures recording the information made persistent that will be exploited for the evolution phase. Such data structures are associated with each node of the profile. The profile with the auxiliary data structures is called *extended profile*.

In each element of the profile we store information about the elements with the same tag found in that position in the hierarchical structure of the document. In particular, for elements that do not perfectly match the constraints imposed by the profile specification we store the frequency of each subelement, the frequency in which group of subelements appears together, the subelements that are not required in the profile, and the subelement of the profile that are missing. These kinds of information will be considered in the evolution phase to determine the new structure of the profile.

Two approaches can be followed for handling profiles and extracted data when we are waiting for user feedbacks. First, the extracted data are made persistent immediately and, in case of negative feedback, removed from the extended profile. Second, the profile is left unchanged and when the positive feedback arrives it is made persistent. In both cases, a queue of profiles waiting for feedbacks should be maintained. We decided to follow the second approach because it does not require to analyze again the document from which we extracted the information. However, we plan to experimentally validate our choice because if negative feedbacks are rarely returned, the first approach could be more efficient.




(a) (b) (c) (d)

**Figure** 5: (a) Profile before evolution, (b,c) documents matched against (a), (d) Profile after evolution

### 3.3 Check Component

The *check* component is responsible for determining whether the evolution phase should be activated. Specifically, we decided to trigger the evolution phase for a profile when, among the documents matched against that profile, the percentage of non relevant documents, and the percentage of non relevant elements in the documents, are above a fixed threshold (which is initially set by the system and then tuned by user feedbacks). Thus, we compute the sum of the percentage of non relevant elements for each document, normalized by the number of examined documents. Let $Doc_P$ be the set of all documents matched against $P$ and $\tau$ be a fixed activation threshold, the evolution phase for a profile $P$ is triggered when:

$$\frac{\sum_{D \in Doc_P} \frac{\#\{e|e \; element \; in \; D \; non \; relevant \; for \; P\}}{\#\{e|e \; element \; in \; D\}}}{\#Doc_P} > \tau$$

### 3.4 Evolution Component

The *evolution* component is responsible for generating a new set of profiles and is able to work at different granularities, ranging from a very coarse granularity, regenerating the whole profile, to a very fine granularity, regenerating the structure of a single element of a subschema of a profile. Making use of the information recorded in the recording phase, some association rules are extracted that represent relationships between presence/absence of subelements of an element. Based on such rules and on some heuristics and parameters we have identified, the new profile is generated.

The *evolve* function of the evolution component is responsible for generating a new set of profiles and works at different granularities, ranging from a very coarse granularity, regenerating the whole profile, to a very fine granularity, regenerating the structure of a single element in the profile. By making use of the information collected in the recording phase, some association rules are extracted that represent relationships between presence/absence of subelements of an element. Based on such rules and on some heuristics we have identified, the new profile is generated. Details of the approach are reported in [3, 5]. The following example presents the intuition of our approach.

*Example 3.* Suppose the profile $P$ in Figure 5(a) has been specified for a user and documents similar to those reported in Figure 5(b,c) have been matched against it. From such documents the following considerations can be pointed out:

- element `c` in $P$ is specified as mandatory with only one occurrence whereas in the matched documents it can be optional (see Figure 5(c)) or can appear more than once (see Figure 5(b));
- element `f` is not specified in $P$ and when it is present it always appears twice;
- the presence of element `c` implies the presence of element `b`, the presence of element `f` implies the presence of element `b`, the presence of `c` implies the absence of element `f`, the presence of element `f` implies the absence of element `c`;
- element `d` required in $P$ is not present in any documents matched against it.

By means of the policies developed in [3, 5] we can conclude that: element `d` can be eliminated from the specification of $P$; element `c` can be repeated an arbitrary number of times and it is an alternative of element `f` which in turn can be repeated twice. Therefore, the new profile in Figure 5(d) is generated. △

### 3.5 Profile Manager

The *profile manager* plays a central role in our architecture. This component is in charge of handling user information, profiles, the profile types and their hierarchical organization. This manager directly interacts with the database management system in which such data are stored. The profile manager should minimize the interaction with the database management system in order to improve the performance of Di$\mathcal{X}$eminator.

Whenever the filtering component requires profiles to match against an incoming document, the profile manager has to identify the profiles that could return a positive degree of relevance by consulting the hierarchy of profile types. Specifically, the profile manager retrieves the profile types whose identifier match against the document root (by means of function $ProfileFor$ introduced in Section 3.1). This approach allows us to filter out profile types (and thus profiles) that should not match the document with a high degree of relevance. Moreover, by considering the hierarchical organization of the profile types, the profile manager easily identifies only once the users to whom the document should be broadcasted.

Another activity of the profile manager is to handle profile updates. Whenever a profile is updated from the evolution component, the profile manager modifies the corresponding profile stored in the database and also maintains up to date the hierarchy of profile types. Moreover, if a user is not interested anymore in a profile type, the profile manager eliminates her identifier from the profile type and from all the descendant profile types.

## 4 Concluding Remarks

In this paper we presented the architecture of Di$\mathcal{X}$eminator, a new selective dissemination system for XML documents based on user profiles. We wish to remark that users of a Di$\mathcal{X}$eminator system can be other Di$\mathcal{X}$eminators. That is, Di$\mathcal{X}$eminators can be

composed in order to redirect documents from a Di$\mathcal{X}$eminator with high level filters to other Di$\mathcal{X}$eminators with more specific filters.

The proposed system can lead to further research directions. Currently, profiles are XML Schemas representing the document types the user is interested in. We plan to extend the system in order to handle profiles that contain only relevant portions of documents the users are interested in. In this way the presence of certain elements in a document can be exploited as a characteristic of the kind of document, thus improving the performance of the system. Another research direction concerns the development of an index structure for improving the retrieval of user profiles that should be used for filtering. A further research direction concerns the extension of XML Schema formalism in order to represent information retrieval operators for expressing some sophisticated constraints on data content elements. A final research direction concerns the integration of an access control mechanism into the Di$\mathcal{X}$eminator system. In this way, the filtering process takes into account the privileges the users have on the documents.

## References


1. R. A. Baeza-Yates and G. Navarro. Integrating contents and structure in text retrieval. *SIGMOD Record*, 25(1):67–79, 1996.
2. E. Bertino, G. Guerrini, and M. Mesiti. 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.
3. E. Bertino, G. Guerrini, M. Mesiti, and L. Tosetto. Evolving a Set of DTDs According to a Dynamic Set of XML Documents. In *EDBT 2002 Workshop Revised Papers*, LNCS(2490), pages 45–66, 2002.
4. M. Franklin and S. Zdonik. "Data in Your Face": Push Technology in Perspective. In *Proc. of Int'l Conf. on Management of Data*, pages 516–519, 1998.
5. M. Mesiti. *A Structural Similarity Measure for XML Documents: Theory and Applications*. PhD thesis, University of Genova, Italy, 2002.
6. A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, November 1995.
7. A. Nierman and H. Jagadish. Evaluating Structural Similarity in XML Documents. In *Proc. of the 5th Int'l Workshop on the Web and Databases*, 2002.
8. J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient Matching for Web-based Publish/Subscribe Systems. In Proc. of *Int'l Conf. Cooperative Information Systems*, LNCS(1901), pages 162–173, 2000.
9. S. V. Rice, H. Bunke, and T. A. Nartker. Classes of Cost Functions for String Edit Distance. *Algorithmica*, 18(2):271–280, 1997.
10. I. Stanoi, G. Mihaila, and S. Padmanabhan. A Framework for the Selective Dissemination of XML Documents based on Inferred User Profiles. In *Proc. of 19th IEEE Int'l Conf. on Data Engineering*, 2003.
11. W3C. Resource Description Framework, 2004.
12. W3C. XML Schema, 2001.
13. T. Yan and H. Garcia-Molina. The Sift Information Dissemination System. *TODS*, 24(4):529–565, 1999.