Query Subscription in an XML Webhouse[*]

Benjamin Nguyen[†] Serge Abiteboul[†] Grégory Cobena[†] Laurent Mignet[†]

Abstract

We consider a query subscription system that can provide users with information about web changes that interest them. We present a query subscription language and a system that combines monitoring of page changes and continuous queries, i.e., queries that are evaluated regularly.

1 Introduction

The web is huge and keeps growing at a healthy pace [ABS00]. Most of the data is unstructured, consisting of text (essentially HTML) and multimedia (sound, video, images). Some is structured and usually stored in relational databases. All this data constitutes the largest body of information accessible to any individual in the history of humanity. A major evolution is occurring that will dramatically simplify the task of developing applications with this data, the coming of XML [W3C98]. The current development of the web and the generalization of XML technology provides a major opportunity for changing the face of the web in a fundamental way. Typically, web users are not only interested in the current values of documents but also in their modifications. We consider here query subscriptions that allow users to obtain information about the web changes they are interested in.

This work is part of the Xyleme [Xyl] project aiming at the development of *a dynamic XML warehouse for the web*.The problems we consider are typical warehousing problems [Wid95]. In this presentation, we focus on a particular aspect, namely the management of subscriptions to changes in the warehouse. There are three main aspects in the subscription mechanism of Xyleme (see Figure 1), that are to a certain extent complementary:

- change monitoring, that consists in filtering the flow of documents acquired by Xyleme to detect documents that may interest certain users based on specifications of these users' interest.
- continuous queries, that consist in regularly asking the same query of interest to the user automatically.
- the generation of a report from the changes detected and continuous query answers.

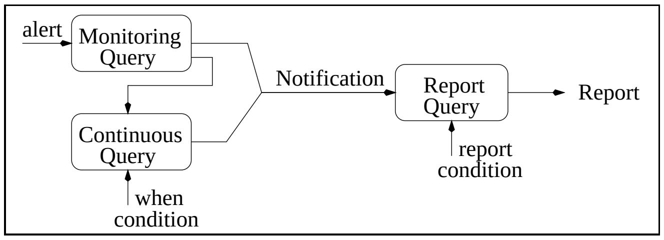

Figure 1: Main Components of a Subscription

We have designed a subscription language to combine these aspects. An example of a subscription is as follows:

```
subscription MyXyleme
monitoring
     select UpdatedDoc(URL)
     where URL extends ‘‘http://www.xyleme.com’’ and modified
monitoring
```

[*]Work supported in part by R.N.R.T.

[†]I.N.R.I.A. VERSO, BP 105 78153 Le Chesnay Cedex, France, Email: firstname.lastname@inria.fr

```
     select NewMember(X)
     from X in self//member
     where URL = ‘‘http://www.xyleme.com/members.xml’’ and new(X)
report
     select...
     % possibly an XML query on
     % the stream of notifications
     when notifications.count > 100
refresh ‘‘http://www.xyleme.com/members.xml’’ biweekly
```

We also present the general architecture of the subscription system we are implementing. The system uses intensively other Xyleme modules such as the acquisition mechanism [MAA$^+$00] or the query processor [ACVW00]. A major issue in this context is scalability. Indeed, most of the problems we consider here would be relatively simple at the level of an Intranet. They are not in the context of the web. XML is still in its infancy, therefore not much XML can be found on the web. However, we believe that XML will mature and that these problems will soon become important. Thus, we discuss technical aspects to allow the scaling of the system to millions of XML and HTML documents per day and the management of millions of subscriptions.

The main contributions are (i) the general architecture tailored to scaling to large number of documents and subscriptions; and (ii) the specification language that combines monitoring and continuous queries, and subscriptions within a flow of several millions per day.

Related works Some of the ideas described here are not new. Some components are inspired by components found in active databases [WC95]. Web subscription systems have already been proposed by some sites, like NetMind [Min] or Northern Light [NL]. Continuous query systems have also been investigated [LPTH00, Con, LPT99, CDTW00, Nia].

2 The Subscription system

In this section, we first the general architecture of Xyleme, and then the subscription system.

2.1 Motivations

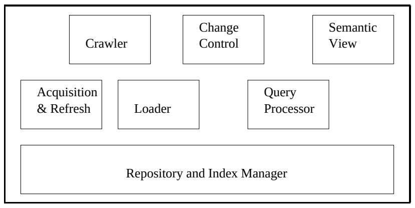

Figure 2: Xyleme Global Architecture

A complete description of the Xyleme system is beyond the scope of the present paper. We only give here a brief and partial description of the main features of the system. The main modules are shown in Figure 2. The lowest layer consists of the XML *repository* and *index manager*. The repository called Natix, that is being developed at Mannheim University [Moe99], is tailored for storing tree-data, e.g., XML pages. Above the repository, on the left hand side, are the three modules in charge of populating and updating the XML warehouse. Since the functionalities of both *crawler* and *loader* are be obvious, let us just say a few words about the *data acquisition* and *refresh* module [MAM00, MAA$^+$00]. Its task is to decide when to (re)read an XML or HTML document. This decision is based on criteria such as the importance of a document, its estimated change rate or subscriptions involving this particular document. On the right-hand side, we find the *query processor* [ACVW00] that is an XML-tailored query processor that supports our own XML query language in the (temporary?) absence of a standard. Above it are the *change control* and *semantic* modules. The main role of the *semantic* module is to classify all the XML resources into semantic domains and provide an integrated view of each domain based on a single *abstract* DTD for this domain. The *change control* module is the topic of the present paper, and will be detailed further.

Xyleme runs on a cluster of Linux PCs [PC]. All modules and in particular the XML loaders and the indexers are distributed between several machines. The repository itself is distributed. Data distribution is based on an automatic semantic classification of all DTDs. The system tries to cluster as many documents as possible from the same domain on a single machine. The entire system is written in C++ and uses Corba [Cor] for internal communication and HTTP for external communication.

2.2 Subscription System

In this section, we present the general architecture of the subscription system shown in Figure 3. This architecture can be broken down into two groups of modules.

- Some generic modules that can be used in a more general change control setting. These are the Monitoring Query Processor, the Subscription Manager, the Trigger Engine, and the Reporter.
- Some application specific modules that are dedicated to the change control in the Xyleme environment. These include the specific Alerters we are using, the Xyleme Query Processor, and some modules to input subscriptions (Xyleme Subscription Manager) and send results (Xyleme Reporter).

In Figure 3, the dotted lines are used for the flow of commands and the filled lines for the dataflow. The generic part of the system is within the thick line rectangle.

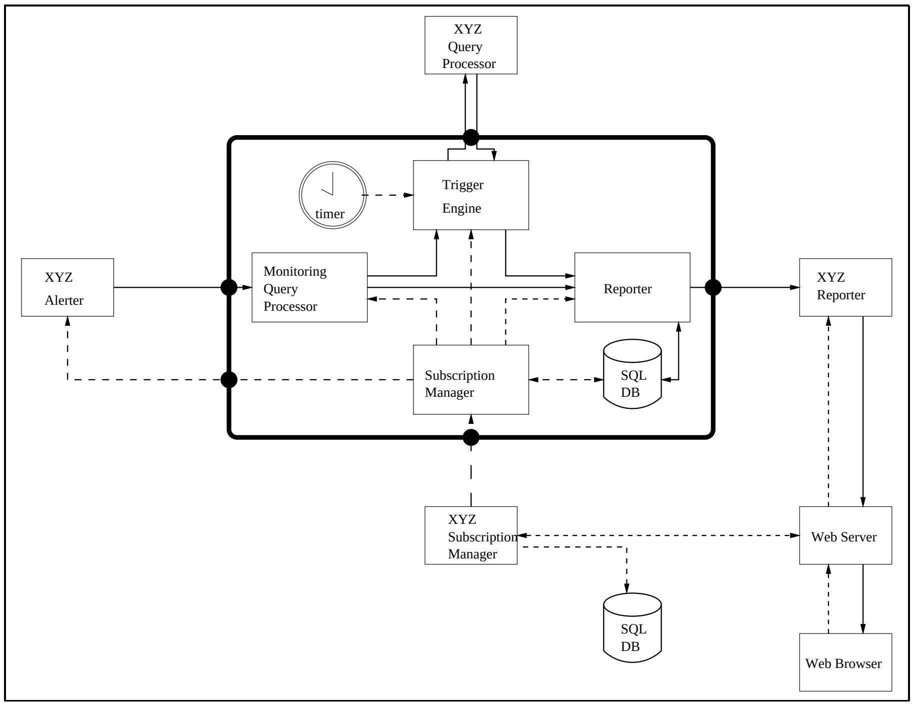

Figure 3: Architecture for the subscription system

The global system For each document, the Alerters detect some atomic events of interest. If the set is nonempty, an alert is sent to the Monitoring Query Processor that consists of the set of atomic events detected together with the requested data. The Monitoring Query Processor determines whether some subscriptions are concerned with these alerts or whether they should trigger some particular processing. For instance, the Trigger Engine may start the evaluation of a query. Subscriptions coming from the Monitoring Query Processor (for monitoring queries) or the Trigger Engine (for continuous queries) of queries are sent to the Reporter. When some condition holds, the Reporter sends the set of subscriptions received so far, an XML document, to the Xyleme Reporter that post-processes by applying an XML query to it. This produces a report that is sent by email, or consulted on the web, with a browser. The Subscription Manager is in charge of controlling the entire process.

Alerters The whole monitoring is based upon the detection of *atomic events*. These depend on the type of documents that are beeing processed. For instance, for HTML documents, typical atomic events considered are the matching of the URL of the document with some string pattern or the fact that the document contains a given keyword. For XML documents, we also consider, for instance, the fact that the DTD of the document is a DTD we are interested in, or that it contains a specific tag, or that a new element with a tag we are monitoring has been inserted in the document.

The role of the Alerters is to detect these events for each document entering the system, and if this is the case, to send an alert for the particular document. Thus we see that these modules are *application dependent*. We distinguish between three kinds of alerters: (i) URL alerters that handle alerts concerning some general information such as the URL of a document or the date of the last update, (ii) XML alerters that are specific to XML documents and (iii) HTML alerters for HTML documents. (Only the first two have been implemented.)

In our implementation Alerters could easily support the rate of the crawlers (eg 4 Million pages/day) with a heavy load of atomic events to monitor (1 Million).

Monitoring Query Processor The system must detect conjunctions of atomic events that correspond to subscriptions. We call *complex event* such a conjunction of *atomic events*. The role of the Monitoring Query Processor is, based on the alerts raised by the each document (i.e. a set of atomic events) the detection of the complex events that the document matches. When such a complex event is detected, the *Monitoring Query Processor* sends a *notification* that consists of the code of the complex event[1] along with some additional data (see the $select$ clause further) to the *Reporter* and/or the *Trigger Engine*. The details of the alogorithm we use for this are beyond the scope of the present paper.

Measures show that the current implementation can process close to half a billion sets of atomic events per day on a standard PC (e.g, the rate of 100 crawlers).

Trigger Engine The *Trigger Engine* can trigger an external action either upon receiving a notification, or at a given date. In our setting, it is in charge of evaluating the continuous queries either when a particular notification is detected or regularly (e.g., biweekly). The query code combined with the result of the query forms a notification that is sent to the Reporter.

The (Xyleme) Reporter The *Reporter* stores the *notifications* that it receives. When a report condition is satisfied, it sends these notifications as an XML document. The Xyleme Reporter post processes this report, basically by applying an XML query to it. Reports are, for the moment, sent by email. We are considering the support of an access to reports via web publication which seems better for very large reports. The main difficulty for the Reporter is the management of a heavy load of emails. In our implementation, the Reporter supports hundreds of thousands of emails a day on a single PC. This limitation is due to the UNIX send-mail daemon implementation.

The (Xyleme) Subscription Manager The *Subscription Manager* receives the user requests and manages the other modules of the subscription system.

3 Subscrition Language

As we have seen in Figure 1, a subscription consists of the following parts: (i) monitoring queries, (ii) continuous queries, (iii) report specification, and (iv) refresh statements, and takes the following form:

```
subscription name
  monitoring...   % (i)
  continuous...   % (ii)
  report when ... % (iii)
  refresh...      %(iv)
```

In this section we next consider (i,ii,iii) in turn; (iv) will be ignored in this paper.

A Monitoring Query has the general form:

```
select  result
(from   from-clause)?
where   condition
```

[1] In fact all the complex events are detected on a document simultaneously and thus are sent to the *Reporter/Trigger Engine* in one batch.

The *from* clause may be omitted because we know the data that is being filtered, i.e., the document that is currently being processed. The current document is denoted *self* in the query. A *from* clause may be used to attach variables to elements of the current document.

The *select* clause describes the data the resulting notification should contain based on constants, $self$ or elements defined in the *from* clause. The notification itself is an XML element. For the moment, we have not implemented this part of the system. Notifications simply return the URL of the document that triggered the monitoring query and basic information about the document.

The where clause is a condition that consists of a conjunction of *atomic conditions*.

An atomic condition may be (i) a condition under the URL (extends or equals a string); (ii) information about the document provided by Alerters or directly by Xyleme, such as :

- (LastAccessed | LastUpdate) <comparator> date;
- (new | unchanged | updated | deleted[2]) *self*;
- self **contains** string.

(iii) element values inside a document following this syntax :

$(< change >)? < element\text{-}name >$ (*contains string*)?
$change :=$ new | updated | unchanged

A Continous Query consists of a standard Xyleme query [ACVW00] plus a condition that specifies when to apply the query. Typically, this condition involves a frequency (e.g., every week). The continuous query may also be triggered by a notification sent by the subscription processor. This part of the system has been specified but not yet implemented.

The Report part of a subscription has the following form:

```
select      ...            % report query
when        ...            % reporting condition
(atmost)?   ...            % limiting conditions
(archive)?  ...            % archiving information
```

The report query is a standard Xyleme query that inputs the current set of notifications, i.e., an XML document, and produces another XML document. The *when* clause indicates when to fill in a new report.

The *when* clause is compulsory whereas the last two clauses are optional. The *atmost* clause sets a limit to the reporting query. For instance, *atmost 500* means that after 500 notifications, we will stop registering the new notifications until the next report. Finally the *archive* clause requests the results of this particular subscription to be archived for some period of time. For instance, *archive monthly* requests the archiving of reports for this particular subscription for a month before garbage collecting them.

Remark Since some subscriptions may be too costly to maintain (e.g. monitoring all the URLs that extend "http://") we have a way of controlling subscriptions that we do not detail here.

Acknowledgments We thank Mihai Preda for the core algorithm of the Monitoring Query Processor, Jeremy Jouglet and David Leniniven for implementing the Reporter, INRIA Postmaster for their support. Finally members of VERSO groups and XYLEME S.A. provided valuable comments.

References

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web*. Morgan Kaufmann, California, 2000.

[ACVW00] Vincent Aguilera, Sophie Cluet, Pierangelo Veltri, and Fanny Watez. Querying xml documents in xyleme. *ACM SIGIR Workshop on XML and information retrieval*, 2000. To appear.

[CDTW00] Jianjun Chen, David DeWitt, Fend Tian, and Yuan Wang. Niagaracq: A scalable continous query system for the internet databases. *ACM SIGMOD*, page 379, 2000.

[2]We will not discuss deletions in this paper. It is not an obvious notion since deletion is rarely explicit on the web.

[Con] Webcq, opencq webpage. http://www.cc.gatech.edu/projects/disl/WebCQ/.

[Cor] Corba web page. http://www.omg.org/.

[LPT99] Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE TKDE*, 11(4):610, 1999.

[LPTH00] Ling Liu, Calton Pu, Wei Tang, and Wei Han. Conquer: A continual query system for update monitoring in the www. *International Journal of Computer Systems, Science and Engineering*, 2000.

[MAA$^+$00] Laurent Mignet, Serge Abiteboul, Sébastien Ailleret, Bernd Amann, Amélie Marian, and Mihai Preda. Acquiring xml pages for a webhouse, October 2000. BDA'00.

[MAM00] Amélie Marian, Serge Abiteboul, and Laurent Mignet. Change-centric management of versions in an xml warehouse, October 2000. BDA'00.

[Min] Mind-it web page. http://mindit.netmind.com/.

[Moe99] Guido Moerkotte. The aodb relational system. U. Mannheim, personal communication, 1999.

[Nia] Niagara webpage. http://www.cs.wisc.edu/niagara/.

[NL] Northern light news search. http://www.northernlight.com/news.html.

[PC] Information on clusters of pcs. http://www.alinka.com/fr/index.htm.

[W3C98] W3C. *eXtensible Markup Language (XML) 1.0*, february 1998.

[WC95] J. Widom and S. Ceri. *Active database systems: Triggers and rules for advanced prcessiong*. Morgan-Kaufmann, California, 1995.

[Wid95] Jennifer Widom. Research problems in data warehousing. *International Conference on Information and Knowledge Management (CIKM)*, 1995.

[Xyl] Xyleme home page. http://www.xyleme.com/.