ELSEVIER

Computer Networks 39 (2002) 645–660

COMPUTER NETWORKS

www.elsevier.com/locate/comnet

Pushing reactive services to XML repositories using active rules

Angela Bonifati *, Stefano Ceri, Stefano Paraboschi

Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, I-20133 Milano, Italy

Abstract

Push technology, i.e., the ability of sending relevant information to clients in reaction to new events, is a fundamental aspect of modern information systems; XML is rapidly emerging as the widely adopted standard for information exchange and representation and hence, several XML-based protocols have been defined and are the object of investigation at W3C and throughout commercial organizations. In this paper, we propose the new concept of active XML rules for "pushing" reactive services to XML-enabled repositories. Rules operate on XML documents and deliver information to interested remote users in reaction to update events occurring at the repository site.

The proposed mechanism assumes the availability of XML repositories supporting a standard XML query language, such as XQuery that is being developed by the W3C; for the implementation of the reactive components, it capitalizes on the use of standard DOM events and of the SOAP interchange standard to enable the remote installation of active rules. A simple protocol is proposed for subscribing and unsubscribing remote rules. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Push technology; Active rules; XML; SOAP; Document management

1. Introduction

One important aspect of Internet-based information systems is the ability of pushing information to clients, by matching new event occurrences with predefined user's interests. Such ability is embedded within many WEB development products [8,23] and applications [3,49], which support one-to-one information delivery in response to users' current and past interactions. Active rules and database triggers are an important ingredient for supporting this reactive technology [1]. All of the above proposals, however, make use of mechanisms which operate locally, on top of given data sources which are controlled by the organizations delivering the information to users. So far, the possibility of distributing the "pushing logic", and installing it at remote servers, has not been considered.

In this paper, we argue that such a possibility is becoming very concrete with the advent of new technological standards, such as XML [7], and XML query languages [10,14,15,17,21,33,39], and with the

* Corresponding author.
E-mail addresses: bonifati@elet.polimi.it (A. Bonifati), ceri@elet.polimi.it (S. Ceri), parabosc@elet.polimi.it (S. Paraboschi).

1389-1286/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S1389-1286(02)00226-8

parallel development of XML-based repository technology [19,38,43]. Furthermore, the Internet and Web communities are repeatedly proposing the use of XML in network protocols and distributed applications—XML-RPC [48], SOAP [41], XMI [46], ebXML [25], ICE [31], IOTP [32] and XML Protocol [47] are only a few examples.

Our proposed approach falls under the generic framework of e-services; such a paradigm denotes a class of Internet computations and systems which fulfill a given objective with some degree of autonomy, for instance because they search within the Internet the best matchings of given client requests, or are capable of simple forms of negotiations. Along these lines, we propose a class of Internet services that behave, in a remote system, by means of active rules; these rules monitor the events occurring at the remote systems and notify interested information consumers. Each rule acts like an independent e-service; a B2B protocol regulates the remote installation of rules at the server, which is proposed by a rule broker and accepted by the remote repository; this negotiation follows a simple installation contract.

The rules that we propose in this paper are not currently supported by XML repositories; however, the standard bodies, and particularly the W3C, are making the appropriate steps in order to make the implementation of such rules rather simple. In particular, active XML rules capitalize on the existence of events in document object model (DOM) (since the Level 2 Specification [22]) and of a standard XML query language, named XQuery, which has recently been proposed by W3C XML Query Working Group. Our proposal is independent from the choice of a particular XML query language, and is currently based on the XQuery Working Draft [15].

We have already discussed, in [6], the issues that arise in the development of active rules for XML. We have presented two specific instantiations of active rules, relative to Lorel and XSL used as query languages; and we have studied the issues of rule conflicts and of their properties, such as termination, confluence, and edit-script independence. It is worth noticing that many of the problems discussed in [6] relative to a generic XML active rule set are much simplified in the environment proposed in this paper, because the active rules that we use have only the ability of notifying remote users, and therefore cannot trigger each other. Thus, termination is guaranteed; conflict resolution policies may determine different orders of notifications to subscribers.

The submission of a rule for its execution by the server permits locating tasks near to the data, which is innovative for the Web context; this is similar to what happens in distributed databases with stored procedures, that locate applicative code within the server maintaining the data [11]. This integration guarantees the fastest possible notification to subscribers, who come to know of events as they occur; it also improves the global systems' efficiency, because services are executed right where the information resides, without requiring expensive data replication.

For the negotiation protocol, and specifically for the interchange between the rule broker and the XML-based repository, we use simple object access protocols (SOAP) and envelopes. The use of SOAP as generic e-service invocation mechanism makes our solution flexible and portable. SOAP was chosen among the several available XML-based protocols, due to its increasing popularity as a lightweight protocol for exchange of information in a decentralized environment. The XML Protocol Working Group at W3C is addressing the specification of requirements of XML Protocol, which condenses and extends the experience of previously defined lightweight protocols, including SOAP. Such a protocol, once deployed, can be easily adopted in our framework.

This paper is organized as follows. After an overview of related work in Section 2, Section 3 briefly presents the syntax and semantics of XML active rules. Section 4 describes the application scenario for rule brokering. Section 5 describes the B2B protocol for submitting a rule to the XML Server. Section 6 describes the steps that are needed in order to implement a reactive engine on top of an XML Server. Finally, Section 7 draws the conclusions.

2. Related work

Event-based computation has been studied in several diversified communities, spanning from software engineering and networking to databases. Within these communities, many event-based distributed architectures have been defined implementing the mechanisms of remote event subscription, filtering and management. Among these systems, we cite OMG Event and Notification Service [36,37], Yeast [35], Ready [29] and Smarts [42]. OMG event service and notification supports asynchronous exchange of events among clients, which can play the roles of event consumers or suppliers and use event channels to communicate. The publishing mechanism of events is based on the pull/push model and a filter facility is provided. In Yeast, event patterns are descriptors, actions are sequences of commands and remote invocations; the underlying communication layer is based on a traditional client/server paradigm. Yeast is extended by Ready, which introduces a specification language for matching of events and quality of service directions. Smarts deploys a distributed system architecture for the purpose of detecting and handling system problems. Event streams occurring in a network are elaborated in real-time and ad-hoc policies are adopted to solve problems. We have taken into account the experience of such event-based computation systems, particularly for what concerns our proposal of a rule subscription mechanism.

Within the database community, a lot of attention has been paid to reactive mechanisms broadened to a distributed environment [44]. Beyond the traditional applications of centralized active database systems, such as support of integrity constraints, materialized views, and derived data, reactive mechanisms can be used to implement services required for network management, e.g., mail services and firewalls. Prominent works concern the maintenance of materialized views in data warehousing systems [45,50], or the constraint maintenance in a distributed environment [12] by means of distributed triggers. However, the application of active rules to the development of reactive push services has not yet been described in the literature.

The unbundling trend in the database field advocates the modularization of monolithic databases into smaller and autonomous services to promote more flexibility and functionality. According to this trend, an event service and a rule service can be enucleated from conventional databases and offered to the nodes of a network, guaranteeing portability and heterogeneity [13,28,34]. In our approach, unbundling is helpful to identify the rule components, but it is not strictly required, as we assume that XML rules will be part of the XML repository.

We discussed the application of XML active rules to implement suitable e-services in [6]; here, active rules implement business applications (such as alerters, personalizers and classifiers) as well as document maintenance. An event detector based on DOM [22] is responsible for capturing the data mutating events on the document and an XML query (expressed in Lorel [33] or in XSL [17]) implements the conditions and actions of the rules. In the present work, XQuery [15] has been chosen to encode XML active rules.

Other works deal with the definition of triggers for XML data, more or less concerned with e-commerce applications. A novel view specification language, equipped with active capabilities has been defined in [1]. The actors involved in an electronic commerce application might need different views of the repository data, and these are encoded through a set of activity specifications, methods and triggers. Enhanced mechanisms for notification, access control and logging/tracing of user activities are provided. Here active rules are application-specific and use a set of proprietary method calls, defined within the views.

Database reactive technology can be considered as complementary to agent technology in implementing event-based computation. Among agent-based systems, Jedi [20] is relevant because it is based on reactive objects, i.e., autonomous computational units performing application specific tasks. Each reactive object has its own thread of control, and interacts with other reactive objects producing and consuming events. They are based on mobile pieces of software (i.e., agents) and use a subscription facility to declare their interest to events. Compared to agents, rules can be more easily implemented, since they are plugged into

the XML repository without any need of mobility. Rules are less sophisticated than agents and offer less flexibility, but in our application context (i.e., push technology), they can be considered both light-weight and powerful enough.

A language-specific and proprietary architecture to provide sophisticated e-services is the purpose of many high-tech industries research. Among the many products available on the market, e-speak [24] introduces several innovative aspects. The e-speak architecture leverages on a message passing mechanism between the e-speak logical machine (or core) and the resources, which provide the services. The communication is obtained through a specific session layer security protocol, and a mailbox metaphor is used to describe the interactions between the clients and the core. E-speak does not support reactive mechanisms.

The use of rules to perform e-services may produce some scalability issues. In fact, as the application of triggers moves from databases towards the Internet, the number of potentially expensive triggers becomes larger and efficiency becomes increasingly desirable. Recent works [2,16,30] discussed the theme of scalability applied to triggers and to continuous query systems. Their common idea is to share common computations among large sets of similar triggers or queries. These techniques could be adapted to the delivery of efficient and scalable XML rule engines. In our framework, rules do not interfere with each other, therefore scalability issues are less relevant than in traditional rule engines.

3. Active rules for XML

The event-condition-action paradigm for active rules has demonstrated in the database context its flexibility and expressive power; each rule is characterized by the events that can "trigger" it; once a rule is triggered, the condition is "considered"; if the evaluation of the condition is successful, the rule action is executed. This model is proposed for XML active rules. A simple rule can be represented by the following XML fragment:

```
<event>insert(//cd)</event>
<condition> FOR $a IN //cd
            WHERE $a=$new AND
            $a/price < 20 AND
            contains($a/author, "Milli Vanilli")
</condition>
<action>
    <soap-header>
    <uri>/notification</uri>
    <host>131.175.16.105</host>
    <soap-action>notify</soap-action>
    </soap-header>
    <SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
      <SOAP-ENV:Body>
    <m:Notify xmlns:m="http://131.175.16.105/methods">
     <cdfound>
       $a//*
     </cdfound>
    </m:Notify>
```

```
    </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
  </action>
```

The rule is triggered by an insertion of a <cd> element. The rule verifies if the sub-element <price> of the new element has a value less than 20 and if the element <author> contains the string "Milli Vanilli". If the condition is verified, the rule invokes the SOAP method Notify on the server 131.175.16.105, passing as parameter <cdfound> the inserted <cd> node and its content. We now give a more detailed description of each part of the rule.

3.1. Syntax

Syntactically, XML active rules for a distributed context need to be enclosed within the SOAP mandatory tags (for simplicity, in the following syntax, these are indicated by the SOAPHeaderTags). Moreover, events, conditions and actions are surrounded by explicitly named XML tags to make them SOAP-embedded. The triggering operations are *insert*, *delete*, *update* possibly issued by a Web user; the events can be multiple, and, in this case, they are separated by a comma (if there are multiple events, either of them can trigger the rule). [1] The event is associated with an XPath expression that indicates the involved elements. The condition is expressed by means of a self-contained XQuery expression, augmented (!) with the variables $new and $old, which express the transition values bound to the occurred events. The condition may be lacking in case of an event-action rule. Finally, the action can be one or more self-standing SOAP methods.

SOAPHeaderOpeningTags
<event> (*TriggeringOperation*'('*XPathExpression*')'['|','])+ </>
<condition> *((XQueryExpression(!))** </>
<action> *(SOAPMethod)+* </>
SOAPHeaderClosingTags

3.2. Event

The *Event* part of the rule specifies the event responsible for rule triggering; it is enclosed into XML <event> tags. A *mutating* event is generated when the XML content is modified; we assume three types of mutating events: *insert*, *delete*, and *update*. The definition of a mutating event declaratively describes the nodes (elements or attributes) whose modifications need to be monitored; every time a monitored modification occurs, the corresponding event instance is generated and associated with the modified node. e.g., an event definition *insert(//house)* monitors the insertion of <house> elements in the repository; an event instance for the event is generated whenever a <house> element is introduced.

The DOM is an API defined by the W3C to access and manipulate XML information; the DOM consists of an object oriented model defined in IDL, that associates a set of methods with the nodes of an XML document. The component of the DOM interface most relevant to the event part of rules is the *Event Model*, introduced in the DOM Level 2 specification. The DOM Event Model assumes that the visualization and manipulation of XML information generates events on the nodes. For the realization of our services, we are interested into the *mutating events*, which are generated when a node of an XML document is modified (either because it is inserted, deleted, or its textual content is modified). Event detection is realized by *event listeners* associated with DOM nodes, which detect events occurring on the nodes to which

[1] The notation uses both the italic font and quotation marks for representing non-terminals.

they are associated or on their descendants. The detection of events on node descendants is based on a bi-directional propagation of the events, which requires that every event navigates downwards from the root of a document to the node instance on which the event occurred; when the event reaches its target, it can propagate back (*bubble* up) to the document root; events may be *captured* by event listeners; event listeners may choose to stop the propagation of events.

The availability of this sophisticated Event Model in DOM offers most of the services required for an implementation of a reactive mechanism for XML. The main problem that this event model presents is that event listeners are associated with node instances, whereas the event part of rules is defined in terms of a declarative specification of the schema element (e.g., a rule should be defined as triggered by an update on the `price` attribute of a `car` element). Thus, the implementation of events on top of the DOM Event Model requires the introduction of a conversion mechanism able to map each declarative event in an adequate set of associations of DOM event listeners with nodes.

3.3. Condition

The *Condition* part of the rule specifies the predicate that must be satisfied to execute the rule's action, expressed through a query which is interpreted as a truth value if it returns a nonempty answer. An important feature is the presence of a communication mechanism between the condition and the event part, so that the condition has a way to refer to the nodes on which the events occurred. This communication is based on predefined variables `$new` and `$old` that represent the nodes on which the events occurred with their current and past values, in a way similar to transition variables of database triggers.

An XML query for the condition can be expressed by using one of the available query languages for XML. The two most interesting alternatives are XPath and XQuery (in [6] we considered Lorel in place of XQuery). XPath is the language that permits the identification of nodes on which XSL templates must operate their transformation; XPath is a readable and intuitive language and has already gained an extensive support (e.g., the Xalan tool produced by the Apache Software Foundation).

The XML Query Algebra [27] is being developed by the W3C XML Query Working Group and is a compact procedural and strong-typed language for XML. Nonetheless, with respect to XPath, the XML Query Algebra is more a semantic and formal specification rather than an optimized target language for XML.

XQuery [15] is the first step towards the definition of a standard XML query language, based on the experience of Quilt [14]. XQuery uses as one of its components XPath, and enriches it with services that permit the construction of an arbitrary XML structure as the result of a query. Therefore, XQuery is a good prototype of an expressive query language that is needed for the design of complex rules, used throughout this paper.

The implementation of the predefined variables `$new` and `$old` that represent the communication channel between the event and the condition depends on the choice of *row* or *set* semantics for the rules. The discussion on this aspect appears in [6], where we describe the two alternatives. Since the row semantics is easier to implement and to use and suited to the "pushing service" application, we assume a row-level semantics; thus, each event generates a rule consideration, in the context of which the variables `$new` and `$old` offer a reference to the node involved in the event. The implementation of these transition variables can take advantage of the `target` attribute of the `MutationEvent` interface and specifically of the `prevValue` and `newValue` attributes.

3.4. Action

The *Action* part of a rule specifies a SOAP method to invoke when the rule condition is evaluated true. We restrict the SOAP method to implement the call to a message delivery system, that will transfer information

to specific recipients. In this way, the action part is much simpler than the general case, as discussed in [6]; in particular, there are no mechanisms for updating the content of the repository, or problems related to rule termination. In this proposal we assume that rules are not prioritized; therefore, if the same event may be serviced by several rules we cannot assume a rule execution ordering, and rule execution is not confluent; however, the addition of a simple prioritization mechanism could take place easily.

The implementation of the communication channel between condition and action is realized by permitting the reuse of condition variables in the action. The system will then replace at run-time each variable with the XML structure created by the query evaluation.

We assume that complex SOAP parameters can be passed to the method being invoked; these parameters are constructed in the condition and passed to the action, thus keeping the action very simple. It may be useful to define constraints limiting the recipients of the SOAP call appearing in the action of a rule to be authorized addresses, thereby introducing security requirements. It is possible to use mechanisms similar to those present in Java applets, for restricting the applet to the invocation of services available on the site from which the applet was downloaded. In general, the work done in the implementation of Java and in the CORBA middleware can offer many valuable suggestions to the design of a distributed execution system like the one we propose, particularly with respect to security aspects.

4. Application scenario

The generic architecture of reactive e-services architectural framework consists of three main actors (see Fig. 1): Service Reseller, Service Supplier, and Rule Broker. The *Service Supplier* delivers goods and services

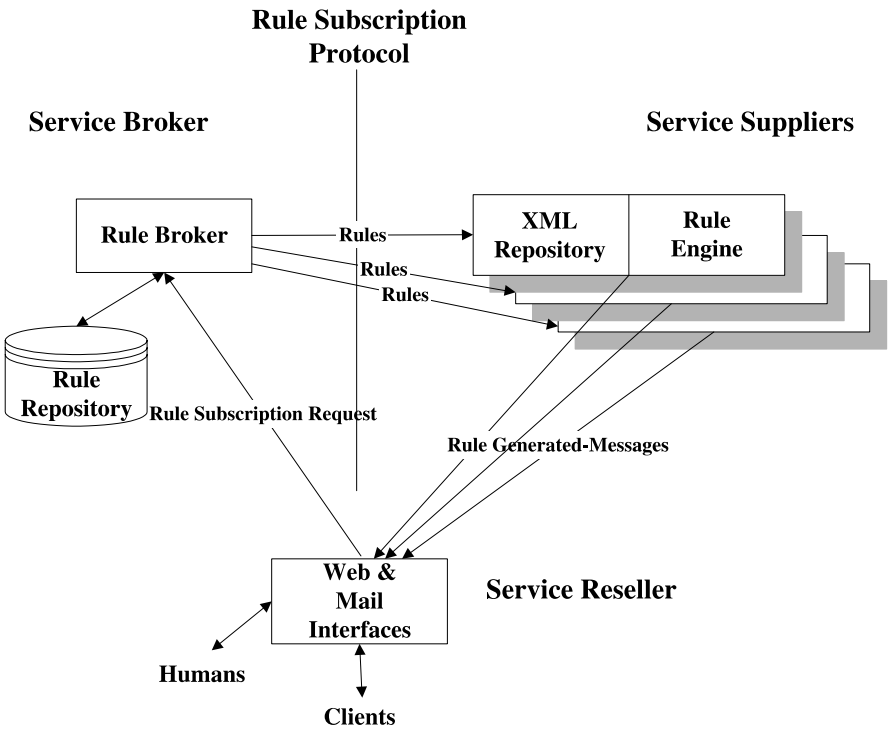


Fig. 1. Reactive e-service application scenario.

described by an XML Server which internally supports an XML rule execution engine. Rules monitor events—such as the new availability of a service—and then notify the service resellers. The *Service Reseller* is, in turn, the recipient of messages concerning new services; they typically interact with clients which are interested in purchasing specific goods or services. The *Rule Broker* acts as an intermediary; it receives information about the services being searched over the Internet, and installs rules at the service supplier sites. To achieve that, it needs to compose the rules in a suitable format and then to install them remotely.

There is no need of a standard protocol offered by the Rule Broker to the Service Reseller; simple WEB-based interfaces may be used in order to acquire information concerning the service being searched. Similarly, the Service Reseller receives a simple message informing of the relevant occurred events, without requiring any special purpose interface. However, the interface between the Rule Broker and the Service Supplier is a classical B2B interface, as the two systems need to establish a protocol which is both technical (yielding to the installation or removal of XML rules at the servers) and business-oriented (yielding to the establishment of mechanisms by means of which the rule broker sees its efforts being repaid). For these reasons, we propose a B2B interface between the Rule Broker and the Service Supplier, based upon four SOAP primitives (*Connect*, *Subscribe*, *Unsubscribe* and *Disconnect*) that are invoked by the Rule Broker and that are supported by the XML Server. We denominate this interface the Rule Submission Protocol and describe it in Section 5.1.

The XML Server needs to be augmented in order to support the protocol mentioned above and to execute XML rules; Section 6 is dedicated to describe the adaptations which are needed in order to upgrade the XML server technology.

The Rule Broker has a double role in the architecture: it performs the task of rule creation and it is responsible for the submission of the rules to all the XML Servers that can contribute to the service. Rule creation requires that the Rule Broker knows how to write rules that satisfy a given applicative need and how to submit them to XML Servers using the Rule Submission Protocol. Rule submission requires that the Rule Broker knows which are the XML Servers that store the pieces of information that are relevant for the application, possibly through interaction with the Service Reseller. In summary, the Rule Broker is the *mediator* of the business transaction that is realized by the reactive services.

The Service Reseller may be directly the user of the notification server, or a mediator which presents the service to the final user with a user-friendly interface.

4.1. Example of application

The architectural framework will be shown at work with a concrete example, in the field of real estate agent applications. An example of request can be: *a furnished four-bedroom (or more), two-bathroom (or more) Victorian house, which costs $1,500,000 or less, located in the Marina area in San Francisco*. A request is immediately issued on a list of house agency XML Servers, but no matching is found. Then, the reactive service is invoked; as a result, a set of rules are submitted to the XML Servers of several house agencies with the task of promptly reacting when a house, which satisfies the requirements, becomes available on the market. To achieve such result, a request is sent from the reseller to the rule broker, a contract is defined upon the rule and the proper authorization for the use and installation of the rule is marshalled, passing through the Rule Subscription Protocol commands.

The message exchange between the Rule Broker and the Reseller is a Rule Subscription Request. The communication and coordination between the Rule Broker and the Reseller can be managed as well through a web interface. After the negotiation, a response can be sent back to the reseller to confirm or reject the subscription. In the following, an example of Rule implementing the request is reported and a Rule-Generated Message delivered by the XML Server to the Reseller, is shown. The information that is sent out to the XML Server is encoded in SOAP and XML; it encloses the rule, the type of contract and the identification of the reseller. The event part is based on any update of relevant elements in the XML re-

sources of the real estate agency servers. The condition part represents the query issued by the final user and is expressed in XQuery; the action part is the invocation of a SOAP method which takes care of alerting the reseller with the news about the houses appearing on the market.

```
POST/Soap/Rules/Subscribe HTTP/1.1
Host: www.expensivehousesincalifornia.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnnSOAPAction: "/Subscribe"

<SOAP-ENV: Envelope
   xmlns: SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV: encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV: Body>
  <m: Subscribe xmlns: m="/http://schemas.xmlsoap.org/soap/rules/">
    <openConnection>343</openConnection>
    <ruleToSubmit>
      <event>update(//housetobuy/house) OR insert(//houseto-
      buy/house)</event>
      <condition>
        FOR $a IN//housestobuy/house,
        WHERE $a//cost < 1500000 AND
              contains($a//style,"victorian") AND
              contains($a//description,"furnished") AND
              $a//nr_of_bedrooms >= 4 AND
              $a//nr_of_bathrooms >= 2 AND
              $a//city="San Francisco" AND
              $a//area="Marina" AND
              empty($a//sold_to)
      </condition>
      <action>
        <soap-header>
          <uri>/NotifyHouse</uri>
          <host>housemediator.com</host>
          <soap-action>notifyHouse</soap-action>
        </soap-header>
        <SOAP-ENV: Envelope
          xmlns: SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
          SOAP-ENV: encodingStyle="http://
                    schemas.xmlsoap.org/soap/encoding">
        <SOAP-ENV: Body>
           <m: DeliverHouseNews xmlns: m="http://
               housemediator.com/soap/methods">
          <foundthehouse>
            $a//*
          </foundthehouse>
          <server>
              www.expensivehousesincalifornia.com
          </server>
```

```
            <localHouseId>
              $a/@Id
            </localHouseId>
           </m:DeliverHouseNews>
          </SOAP-ENV:Body>
        </SOAP-ENV:Envelope>
      </action>
    </ruleToSubmit>
  <contractProposed>
     <cost>0</cost>
     <guarantee>none</guarantee>
    </contractProposed>
  </m:Subscribe>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

When a house satisfying the search is retrieved, a SOAP invocation is produced on the XML server and sent back to the Service Reseller under the format of a Rule-Generated Message.

```
POST /NotifyHouse HTTP/1.1
Host:housemediator.com
Content-Type:text/xml;charset="utf-8"
Content-Length:nnnnSOAPAction:"notifyHouse"

<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding">
<SOAP-ENV:Body>
  <m:DeliverHouseNews xmlns:m="http://housemediator.com/soap/methods">
    <foundthehouse>
      <house Id="1745">
        <address>
          <street>A Street</street>
          <area>Marina</area>
          <city>San Francisco</city>
          <zip>94120</zip>
        </address>
        <cost>1450000</cost>
        <squarefeet>1600</squarefeet>
        <year_construction>1925</year_construction>
        <year_refurbished>1980<year_refurbished>
        <nr_of_bedrooms>6<nr_of_bedrooms>
        <nr_of_bathrooms>2<nr_of_bathrooms>
        <nr_of_balconies>2<nr_of_balconies>
        <miscellaneous>
          <style>modern and victorian</style>
          <view>Ocean view</view>
          <description>
```

```
          Possibly furnished, wide kitchen,
          no smoking policy
          </description>
        </miscellaneous>
      </house>
    </foundthehouse>
      <server>
        www.expensivehousesincalifornia.com
      </server>
      <localHouseId>
        1745
      </localHouseId>
  </m:DeliverHouseNews>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5. Architectural framework

In this section we introduce a proposal for the protocol used to submit rules to the repository. We are interested in presenting the features that are needed for a successful deployment of these services in the Internet context. This is the low-level protocol for the definition of e-service requests.

5.1. E-service request protocol

A minimal interface for the submission of rules is represented by the following primitives: `Connect`, `Subscribe`, `Unsubscribe`, and `Disconnect`. Our goal is to present the fundamental primitives and parameters required for a rule submission service; actual implementations will use additional support primitives and a more complex API. For readability, we represent the primitives in IDL, even if they will be implemented as SOAP invocations to the XML server. We describe separately each of the primitives.

```
ConnectionId Connect(in AuthenticatedUser user,
          in ServerProfile requestedProfile)
```

The `Connect` primitive creates a connection for the submission of rules to a remote XML repository. A connection is established, instead of using a state-less approach as in HTTP, because the submission of rules typically requires that users are authenticated and the capabilities of the server are verified. Instead of repeating these steps for every rule submission, it is preferable to do them once and separately for every set of rule submissions. Once a connection is established, all future requests originating from the same location can refer to the same connection.

The parameters of the request are an instance of *AuthenticatedUser*, which represents the user with the password or the credentials that are needed to verify his identity and the corresponding privilege to submit rules. The second parameter is an instance of *ServerProfile*, which contains a list of features that the submitter expects to be supported by the server. For instance, the requester may present a list of rule languages and ask which are the ones accepted by the system, it may query if the server stores XML information conforming with a given DTD, or it may request if SOAP calls in the rule action should only

return to the submitter; any other parameter that may be relevant in this context may be part of the server profile. If the user privileges are sufficient to open a connection with the server, the response of the call returns a valid connection identifier, combined with the answer to each of the components of the server profile.

```
SubmissionId Subscribe(in ConnectionId openConnection,
                       in Rule ruleToSubmit,
                       in Contract contractProposed)
```

The `Subscribe` primitive permits the submission of a rule to a server. Its parameters are the connection which the request refers to (*openConnection*), the rule (*ruleToSubmit*), and the contract on which the service is realized (*contractProposed*). The role of the last parameter is the focus of Section 5.2. If the request is successful, it will return a valid *SubmissionId*; if the connection is not open, the rule is not understood, the contract is not accepted or the query is not satisfied, the request will not be accepted and it will return an error code, with an explanation of the motivation for the refusal in the SOAP body of the response.

```
void Unsubscribe(in ConnectionId openConnection,
                 in SubmissionId subId)
```

The `Unsubscribe` primitive is invoked when a submitted rule must be removed from a server. The request must originate from the same user who submitted the rule. The parameter *subId* is a rule identifier internal to the XML server that has been returned by the `Subscribe` request.

```
void Disconnect(in ConnectionId openConnection)
```

The `Disconnect` primitive closes the connection created by the `Connect` primitive and frees the resources required for its management on the server. A timeout mechanism can automatically invoke the primitive for connections that remain idle beyond a predefined duration.

5.2. Rule packaging

The primitive *Subscribe* used to submit active rules to an XML Server presents a *contractProposed* parameter. This parameter specifies the contract that the rule submitter offers to the XML Server. A contract describes a set of obligations agreed by each party in the transaction. In this context, the Rule Broker will typically describe the remuneration requested to the Server for the acceptance of the rule and for its execution. Typically, a Broker will be guaranteed either a fixed rate for each installed rule, or a variable rate for each delivered message, or both. In addition, the XML Server will typically offer guarantees regarding the responsiveness of the rule, its robustness, and its availability.

Experience in the construction of B2B applications demonstrates that one of the most critical aspects for the success of a B2B initiative is the definition of adequate contracts. Even for trading commodities, where the problem of defining a market seems relatively simple, several systems failed because they were unable to represent precisely the "contract", that is, the assumption of responsibilities that a commercial transaction implies.

It is indeed quite difficult to prescribe a solution that will satisfy the requirements of every context. The main prescription that we present is the need of this component in the interface, whose exact structure depends on the characteristics of the application and requires a specific study.

6. Implementation

In this section we show the guidelines that can be used for the implementation of an active rule system for XML. An interesting feature of our proposal is the relatively easy implementation and integration with existing Web solutions, based on the reuse of several current Web standards and of their robust implementations.

The run-time behavior of a single rule execution is modeled by a process that binds the three parts of the rule, interacting with the subsystems responsible for each of them: the DOM Event Model, the XQuery engine, and the generator of SOAP calls.

The DOM Event Model permits the definition of arbitrary event listeners, which consist of generic procedures, written in the language implementing the interface (e.g., Java), that need to present the implementation of a set of predefined methods. The event listener can be dynamically associated with the document nodes, for a certain event, and its methods are invoked when an event of the specified kind on the specified node is generated. As we discuss below, the Event Model of the DOM generates processes for rule execution according to two different alternatives. In either case, the rule process first extracts from the event the information that may be required to complete the XQuery query that represents the condition. Then, the condition is executed by a XQuery processor, which may return an empty result or an XML fragment. When the result is empty, the condition is not satisfied, and the execution of the rule process terminates. If the query returns an XML fragment, then the rule action is executed, by extracting from the query result the information necessary for the construction of the SOAP call; then, the call is submitted to the server appearing in one of the action elements.

The rule engine is the software component at the core of an active rule system. In database trigger systems, its responsibilities include the determination of the triggered rules, the selection of the order of rule execution among rules triggered at the same time, and in general the coordination among separate rules [5]. This role is greatly simplified by the assumptions on rules that we made in this paper.

We have seen the detection of events and the triggering of rules is directly managed by the Event Model of the DOM. We identify two main alternatives for the realization of this task.

- The *centralized* solution associates a single event listener with the root of the document. This alternative requires the creation of a complex event listener, which can be considered as the main component of the rule engine. The event listener classifies each event and detects if the occurred event triggers one of the active rules. This task requires an efficient description of the rule repository, with a structure similar to that required for the implementation of trigger engines. Once a rule is triggered, the process managing it is activated.
- The *fragmented* solution creates a set of event listeners, associated with every node instance on which events have to be monitored. This solution introduces an event listener for every rule, and associates the event listener with every node instance on which the triggering events can occur.

In the database field, previous research has analyzed the impact that various design parameters could have on system performance [4]. Based on those results and on the characteristics of this context, it is possible to estimate that the main criteria to use in the choice between the centralized and the fragmented solution are the cost of the association between event listeners and nodes, the number of monitored nodes compared with the number of document nodes, and the frequency of irrelevant events. The fragmented solution is preferable when the association between an event listener and a node requires limited resources, when the monitored nodes are a small fraction of the document nodes, and when the events that trigger rules are very few compared with the produced events.

A particularly sophisticated implementation of the fragmented solution may be optimized to restrict the creation of event listeners only to the nodes that have the potential to satisfy the rule condition. This strategy has a great potential and is the one that can offer the best performance in many applications.

7. Conclusions

In this paper, we propose the use of active rules for pushing reactive services; we have shown that such services satisfy the needs of many important applications. Below, we list some of the possible obstacles that could limit the applicability of our solution, and explain why each of them is not critical.

The first observation is that in order to write an efficient rule it may be necessary to know the schema of XML resources; this is a serious obstacle for a wide-scale rule deployment, where the Rule Broker submits the rule to many different and autonomous sites. Indeed, this problem is faced by most of the wide-scale, inter-business applications based on XML. Fortunately, many ongoing initiatives (RosettaNet [40], eCo Framework [26], OMG CORBA [18], ebXML [25]) are dedicated to the definition of DTDs and schemas for specific industrial sectors, that should permit interoperability among systems. These efforts will offer considerable benefits to our application as well.

Another observation is that a rule is an external program coming from an external system, whose execution may present unacceptable risks (such as: access to protected resources, malicious code, etc.). This is an intrinsic characteristic of any agent-based mechanism. In our context, however, rules are installed only by trusted sites (the Rule Brokers) and they are (on purpose) severely restricted in the scope of their actions. They cannot modify the state of XML resources, and SOAP calls can be addressed only to given, certified methods. Thus, we believe that the application scenario considered in this paper is sufficiently protected and secure.

A final observation is that most relational trigger systems and applications do not exhibit high scalability: when the number of triggers becomes large, applications often become inefficient and unmanageable. However, from the description of the implementation given in Section 6, it is evident that the rule system hereby proposed is much simpler than a generic trigger engine. In particular, interactions among rules are excluded, thereby eliminating one of the main causes of inefficiency and mismanagement. In addition, knowing in advance the structure of rules gives great potential for optimization strategies. In any case, the transition from generic, trigger-based implementations of services (enabling their rapid prototyping) to efficient embedded solutions has already occurred in many active rule applications [9], and could take place in our context as well.

In summary, services based on reactive push technology have great potential applicability; their flexibility and ease of implementation can make such services one of the key ingredients of future Web-based infrastructures.

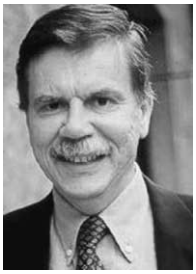
References

[1] S. Abiteboul, C. Cluet, L. Mignet, B. Amann, T. Milo, A. Eyal, Active views for electronic commerce, in: 25th Very Large Data Bases Conference Proceedings, September 1999, pp. 138–149.
[2] M. Altinel, M. Franklin, Efficient filtering of XML documents for selective dissemination of information, in: 26th Very Large Data Bases Conference Proceedings, September 2000, pp. 53–64.
[3] Amazon Web Site, 2001, http://www.amazon.com.
[4] E. Baralis, A. Bianco, Performance evaluation of rule semantics in active databases, in: 13th ICDE Conference Proceedings, April 1997, pp. 365–374.
[5] E. Baralis, S. Ceri, S. Paraboschi, Compile-time and runtime analysis of active behaviors, IEEE Trans. Knowl. Data Eng. 10 (3) (1998) 353–370.

[6] A. Bonifati, S. Ceri, S. Paraboschi, Active rules for XML: a new paradigm for e-services, in: 1st Workshop on e-Services (co-held with the 26th Very Large Data Bases Conference) Proceedings, September 2000.
[7] T. Bray, J. Paoli, C.M. Sperberg-McQueen (Eds.), Extensible Markup Language (XML) 1.0–2nd Edition, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006.
[8] Broadvision Web Site, 2001, http://www.broadvision.com.
[9] S.Ceri, R.J. Cochrane, J. Widom. Practical applications of triggers and constraints: success stories and lingering issues, in: 26th Very Large Data Bases Conference Proceedings, September 2000, pp. 254–262.
[10] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, L. Tanca, XML-GL: a graphical language for querying and restructuring WWW Data, in: WWW8 Conference Proceedings, May 1999.
[11] S. Ceri, G. Pelagatti, Distributed Databases: Principles and Systems, McGraw-Hill, New York, 1984.
[12] S. Ceri, J. Widom, Production rules in parallel and distributed database environments, in: 18th Very Large Data Bases Conference Proceedings, August 1992.
[13] S. Chakravarthy, R. Le, R. Dasari, ECA rule processing in distributed and heterogeneous environments, in: International Symposium on Distributed Objects and Applications, 1999.
[14] D. Chamberlin, J. Robie, D. Florescu, Quilt: an XML query language for heterogeneous data sources, in: Webdb Conference Proceedings, Dallas, TX, May 2000.
[15] D. Chamberlin et al. (Eds.), XQuery: a query language for XML, W3C working draft, 15 February 2001, http://www.w3.org/TR/2001/WD-xquery-20010215/.
[16] J. Chen, D. DeWitt, F. Tian, Y. Wang, NiagaraCQ: a scalable continuous query system for internet databases, in: ACM Sigmod Conference Proceedings, May 2000.
[17] J. Clark (Eds.), XML Transformations (XSLT) Version 1.0, November 1999, http://www.w3.org/TR/xslt.
[18] Corba Web Site, 2001, http://www.omg.org.
[19] Crossgain Web Site, 2001, http://www.crossgain.com.
[20] G. Cugola, E.D. Nitto, A. Fuggetta, Exploiting an event-based infrastructure to develop complex distributed systems, in: 20th International Conference on Software Engineering Proceedings, April 1998.
[21] A. Deutsch, M.F. Fernandez, D. Florescu, A.Y. Levy, D. Suciu, A query language for XML, in: WWW8 Conference Proceedings, Toronto, Canada, May 1999, http://www8.org/fullpaper.html.
[22] Document Object Model (DOM) Level 2 Core Specification Version 1.0 W3C Proposed Recommendation, September 2000, http://www.w3.org/TR/2000/PR-DOM-Level-2-Core-20000927/.
[23] Dynamo Web Site, 2001, http://www.dynamo-ny.com/flash_index.html.
[24] E-Speak Web Site, 2001, http://www.e-speak.hp.com/map.shtm.
[25] ebXML Web Site, 2001, http://www.ebXML.org.
[26] The eCo Framework, 2001, http://www.commerce.net.
[27] P. Fankhauser et al. (Eds.), The XML Query Algebra, December 2000, http://www.w3.org/TR/query-algebra/.
[28] S. Gatziu, A. Koschel, G. von Bultzingsloewen, H. Fritschi, Unbundling active functionality, ACM Sigmod Record 27 (1) (1998) 35–40.
[29] R. Gruber, B. Krishnamurthy, E. Panagos, The architecture of the READY event notification service, in: Workshop on Middleware (co-held with the International Conference on Distributed Computing Systems) Proceedings, May 1999.
[30] E.N. Hanson, C. Carnes, L. Huang, M. Konyola, L. Norohna, S. Parthasarathy, J.B. Park, Scalable trigger processing, in: 15th ICDE Conference Proceedings, 1999, pp. 266–275.
[31] ICE Protocol Specification (CGA), 1999, http://www.idealliance.org/ice/icenote-ice-19990519.asp.
[32] Internet Object Transfer Protocol Specification (Commerce One), 2001, http://search.ietf.org/internet-drafts/draft-ietf-trade-iotp-v1.0-protocol-07.txt.
[33] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom, Lore: a database management system for semistructured data, ACM Sigmod Record 26 (3) (1997) 54–66.
[34] A. Koschel, S. Gatziu, H. Fritschi, G. von Bultzingsloewen, Applying the unbundling process to active database systems, in: International Workshop on Issues and Applications of Database Technology Proceedings, Toronto, Canada, July 1998.
[35] B. Krishnamurthy, D. Rosemblum, Yeast: a general purpose event-action system, IEEE Trans. Software Eng. 21 (10) (1995).
[36] Object Management Group 2000, Event Service Specification, 2000, http://www.omg.org/technology/documents/formal/event_service.htm.
[37] Object Management Group 2000, Notification Service Specification, 2000, http://www.omg.org/technology/documents/formal/notification_service.htm.
[38] Propel Web Site, 2001, http://www.propel.com.
[39] J. Robie (Ed.), XQL (XML Query Language), August 1999, http://www.ibiblio.org/xql/xql-proposal.html.
[40] RosettaNet Framework, 2001, http://www.rosettanet.org.
[41] Simple Object Access Protocol (SOAP) 1.1 (W3C Note), 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[42] System Management Arts, Root Cause Analysis and its Role in Event Correlation (White Paper), 2001, http://www.smarts.com.
[43] Tamino Product Web Site, 2000, http://www.softwareag.com/tamino/.
[44] G. von Bultzingsloewen, A. Koschel, P. Lockemann, H. Walter, ECA functionality in a distributed environment, in: N.W. Paton (Ed.), Active Rules in Database Systems, 1999, pp. 147–175.
[45] J. Widom, Research problems in data warehousing, in: Information and Knowledge Management Conference Proceedings, August 1995, pp. 583–595.
[46] Unysis et al., XML Metadata Interchange, Updated 2 October 1999, http://www.omg.org/cgi-bin/doc?ad/99-10-02.
[47] XML Protocol Requirements, W3C Working Draft 19 December 2000, http://www.w3.org/TR/2000/WD-xp-reqs-20001219/.
[48] XML-RPC Specification (Userland), Updated 16 October 1999, http://www.xmlrpc.com/spec.
[49] Yahoo Web Site, 2000, http://www.yahoo.com.
[50] G. Zhou, R. Hull, R. King, J. Franchitti, Supporting data integration and warehousing using H20, IEEE Data Eng. Bull. 18 (2) (1995) 29–40.

Angela Bonifati received her Laurea Degree in Ingegneria Informatica in 1997 from University of Calabria. Currently she is approaching the completion of her Ph.D. in Computer Science at Politecnico di Milano. Her main research interests focus on XML data management, XML query and update languages, data warehouses and databases for software engineering.



Stefano Ceri is full professor of database systems at the Dipartimento di Elettronica e Informazione of Politecnico di Milano. His research interests focus on distributed databases, deductive and active rules, and object-orientation. He is author of several books, most recently of "Database Systems: Concepts, Languages, and Architecture" (McGraw-Hill, 1999). He is a member of ACM-SIGMOD steering committee, VLDB endowment and EDBT foundation. He won the 10 year VLDB Award in 2000 and was the General Chair of VLDB 2001.

Stefano Paraboschi is an associate professor at the Dipartimento di Elettronica e Informazione of Politecnico di Milano. He received the Laurea Degree in Ingegneria Elettronica in 1990, and a Ph.D. in Ingegneria Informatica in 1994, both from Politecnico di Milano. His main research interests are in the area of databases, with a focus on active databases, data warehouses, and the construction of data-intensive Web sites. He is the author, together with Paolo Atzeni, Stefano Ceri, and Riccardo Torlone, of the book "Database Systems: Concepts, Languages and Architectures" (McGraw-Hill 1999).