The VLDB Journal 10: 39–47 (2001) / Digital Object Identifier (DOI) 10.1007/s007780100039

Active rules for XML: A new paradigm for E-services

Angela Bonifati, Stefano Ceri, Stefano Paraboschi

Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milan, Italy;
E-mail: {bonifati,ceri,parabosc}@elet.polimi.it

Edited by F. Casati, M.-C. Shan, D. Georgakopoulos. Received: 30 October 2000 / Accepted: 19 December 2000
Published online: 27 April 2001 –

Abstract. XML is rapidly becoming one of the most widely adopted technologies for information exchange and representation. As the use of XML becomes more widespread, we foresee the development of active XML rules, i.e., rules explicitly designed for the management of XML information. In particular, we argue that active rules for XML offer a natural paradigm for the rapid development of innovative e-services. In the paper, we show how active rules can be specified in the context of XSLT, a pattern-based language for publishing XML documents (promoted by the W3C) which is receiving strong commercial support, and Lorel, a query language for XML documents that is quite popular in the research world. We demonstrate, through simple examples of active rules for XSLT and Lorel, that active rules can be effective for the implementation of e-commerce services. We also discuss the various issues that need to be considered in adapting the notion of relational triggers to the XML context.

Key words: Active databases – Document management – XML – XSLT – Query languages for XML

1 Introduction

E-services are emerging as a new paradigm to build Web applications. XML (*eXtended Markup Language* [XML98]) is one of the main technological ingredients in this scenario; XML repositories have an important role, since they can be used both as direct data sources and as an interoperability layer to conventional data sources.

Research on e-services is currently focusing on the definition of mechanisms for interaction among existing services, normally with the assumption that e-services have already been implemented with traditional technologies. Our work instead focuses on the rapid development of e-services and their support mechanisms, based on the specification of active rules for active XML management. By means of active rules, it becomes possible to generate or manipulate the content of an XML repository as a reaction to changes which occur to XML information, offering an infrastructure that is particularly relevant to the definition of e-services. We list the main benefits that active rules are able to offer in this context.

- First, e-services defined by active rules are autonomously executed after events on XML data. This guarantees that the e-service is invoked automatically as soon as the relevant XML information is modified, without the need to introduce an explicit coordination mechanism or a workflow description responsible for the invocation.
- With respect to traditional technology used to implement e-services, the active paradigm, due to its simplicity, permits an easier design of e-services, and, in addition, permits them to be rapidly prototyped and tested. Services implemented by active rules can be incrementally created and dropped, and this fits particularly well with e-services, which exhibit considerable dynamicity and should flexibly adapt to changes in the environment.
- In addition, the action of rules may invoke procedures that are executed remotely and pass them XML-formatted parameters (e.g., according to the SOAP protocol). This permits the rapid integration of an e-service based on active rules with other e-services implemented with traditional technologies.
- Finally, negotiation among e-services can be modeled by conflict resolution policies for rules, or additional negotiation policies can be introduced if needed. Thus, it is possible to develop or modify negotiation policies by simply altering rule priorities, and to enforce in the rule set different negotiation policies, once they are required.

Active rules for XML enable a rapid development of a number of rather conventional applications, such as the automatic synthesis of XML content, the incremental maintenance of related information, the development of refresh services for views, or the checking of integrity constraints. In addition, rules guarantee support to a variety of new and important e-services; in this paper, we demonstrate the development of e-services for asynchronous selective notification, for e-mail classification, and for personalized delivery of information (i.e., presenting different information on a web site, given the user's profile). Additionally, active rules may be used to check the correctness of e-service invocation and parameter passing.

Correspondence to: A. Bonifati

Even if rules are a very powerful and flexible tool, we do not consider them as the final tool to use for every e-service. Indeed, triggers have been used for the rapid prototyping of several new applications in the relational DBMS context: many new DBMS services (view materialization, data replication, integrity maintenance, etc.) have been implemented by first using triggers and eventually by using specialized mechanisms, offering higher performance [CCW00]. Similarly, the main advantage that active rules will be able to offer in XML repositories is the possibility of a rapid design and evaluation of many new e-services.

This paper provides an example-driven presentation of the features that we expect to be present in active rules for XML. We introduce two active variants of existing query languages for XML that we consider to be particularly representative. We first extend XSLT (*Extensible Stylesheet Language Transformations* [XSLT99]), a pattern-based language providing transformation rules for XML, which is well-known as a standard and has wide commercial support. We next concentrate on Lorel, a query language for XML that is quite popular in the research world, whose features are likely to influence the language that will be proposed by the W3C (*World Wide Web Consortium*) as standard XML query language. Our approach is first to present examples, so as to familiarize the reader with the extensions that are required in order to support active XML management. Our main observation is that such extensions are quite similar and can be precisely characterized. We next discuss the issues that need to be solved in order to support active rules on top of XML repositories, and describe the research agenda that must be addressed for a concrete development of active rule services.

2 Active rules for XML

The increasing interest in XML for document representation has given rise to novel languages [BC00] for extracting information from XML content, much in the same way as traditional query languages for databases (e.g., SQL); among these are XML-QL [DF*98], Lorel [AQ*97, GMW99], XSLT [XSLT99],
XQuery [De98], XML-GL [CC*99], XQL [RLS98], and Quilt [CRF00]. Languages for content-based updates or for extracting views of XML documents are also emerging [AQ*97, DOM00, GMW99], and other proposals permit the declaration of the content of a document and force semantic integrity constraints upon them [XSc00]. All these solutions offer the base components required for the design and implementation of an active rule system for XML.

An active rule for XML follows the conventional ECA paradigm and its components are events, conditions, and actions.

- Events are changes to an XML document, caused by the direct invocation of a data manipulation primitive within an XML repository, or by higher-level operations, such as the publishing or updating of a Web page, the creation of a new document, the closing of an editing session, or the receipt of an XML document by e-mail.
- Conditions are queries on the XML repository, expressed by using an XML query language; the query inspects the content of the changed XML document and compares it with its original state or the content of other XML information available in the repository. As in many database systems, queries are interpreted as true predicates when they return a non-empty XML result.
- Actions are executed only when the condition, interpreted as a predicate, returns a true value. An action consists of building new XML documents and/or modifying existing XML documents; these may or may not be associated with a DTD.

Although conditions were kept separate from actions in the early development of active rule systems, (e.g., see [WC96]), in the industrial development of trigger systems the condition and action part of an active rule have been kept integrated. This is perhaps less declarative, but enables efficient binding of the data which causes the condition to be true and of the data to which the action should be applied; these are coincident in most meaningful applications. Based on this trend, we have decided to keep XSLT and Lorel unchanged concerning their ability to express as a single computation both the condition (query on the XML resource) and action (either retrieving another XML resource, or changing the content of the present one). We therefore designed a language extension to cover the event specification, which is absent from both languages. As a result, our XML active document language consists of an event part and of a condition-action part, with data bindings being generated by the event part and used by the condition-action part.

An XML document is a tree structure of tag-delimited elements and attributes. Elements and attributes are collectively called *nodes*. Changes to XML documents may add a node, delete a node, or update the node contents; an update occurs when a node keeps the same name (i.e., the tag string for elements and the name for attributes) and becomes associated with a different value (i.e., the CDATA or PCDATA content for elements and the value for attributes).

More formally, we consider events the insertion and deletion of elements as well as the insertion, deletion, and update of attribute values and of CDATA or PCDATA contents; with reference to the DOM (*Document Object Model*) Level 2 event model [DOM00], these events are classified as *mutation events*. *Event listeners*, associated with DOM nodes, are responsible for event detection both on the nodes on which they are located or on their descendents. In both our active extensions of XSLT and Lorel, events are represented as update primitives (insert, delete, update, change) relative to a path expression; the affected nodes of the document are denoted by a variable which can be used within the XSLT and Lorel queries. In this way, event bindings are “passed” from the event part to the condition-action part of the rule, which are written in standard XSLT and Lorel. In addition, we enable in XSLT and Lorel the use of the functions `old` and `new` to denote the values of the affected node, respectively, before and after the application of the update primitive.

3 Active rules in XSLT

XSLT is a language designed by the W3C XSL Working Group [SLR98, W*98, XSLT99]. An XSLT program consists of a collection of template rules; each template rule has two parts: a pattern which is matched against nodes in the source

tree and a template which is instantiated to form part of the result tree. XSLT makes use of the expression language defined by XPath [XPa99] for selecting elements for processing, for conditional processing, and for generating text.

XSLT template rules assume a rule processor which considers the nodes of a document and applies to them the specified transformations. Only one transformation can be applied to a node; if a node matches different templates, the XSLT processor employs a conflict resolution policy based on the consideration of template priorities and pattern specificity; if these criteria do not permit the identification of the applicable rule, the template defined last is applied.

Overall, the enrichment of template rules with events permits us to easily obtain a powerful representation of active rules. The critical aspect of this extension is the definition of an adequate mechanism for rule processing, where rules are driven by the detection of events.

3.1 Examples

We show several examples of active rules in XSLT, that demonstrate the types of e-services that can be managed in a uniform and abstract way with the active-rule paradigm.

3.1.1 Alerter

The first example is an alerter set up by an online trading company, that causes an SMS message to be sent to the customer's cellular phone if a monitored quote reaches a given threshold relative to its opening value. The service is based on two XML resources available at a stock trading company. One describes the current quotes of each stock; the other one drives the alerting service by indicating the name of the stock and the percentage of increase (or decrease) of the quote relative to its last opening value. Some XML fragments are shown in the following:

```
<Quotation>
  <Name> Microsoft </>
  <CurrentValue> $100.00 </>
  <Opening> $110.00 </>
</>

<User>
  <FirstName>Stefano</>
  <LastName>Ceri</>
  <e-mail>ceri@elet.polimi.it</>
  <MobilePhone>334-455211</>
  <RaiseFactor>+0.05</>
  <QuotationMonitored>Microsoft</>
</>
```

The rule consists of an event command `<rule:update/>` which is added by us to XSLT, and then a standard XSLT rule, which consists of two nested templates. The event part binds an XSLT variable `Q` to a quotation whose current value is updated. The condition part matches quotations with users' profiles, by matching stock names and then checking that the rise of the quotation since the opening is above the threshold level; if so, the action consists in the call of another XSLT template named `Alerter` which sends an alert message to the owner of the quotations.

```
CREATE RULE CheckQuotation
PRIORITY: 0
EVENT:    <xsl:variable name="Q" select="Quotation">
          <rule:update  select="$Q/CurrentValue">
COND_ACT: <xsl:variable name="U" select="User">
          <xsl:template  match="$U">
            <xsl:if test="QuotationMonitored=$Q/Name and
                          number($Q/CurrentValue)-
                          number($Q/Opening) >
                          (number($U/RaiseFactor) *
                                  number($Q/Opening)]"/>
                <xsl:message>
                   <xsl:call-template name="Alerter"/>
                </xsl:message>
             </xsl:if>
           </xsl:template>
           <xsl:template name="Alerter"
            match="Alerting-Service">
                <sendto>
                  <xsl:value-of select="$U/MobilePhone"/>
                  <subject> Alert from the Stock Market:
                      <xsl:value-of select=
                       "$U/QuotationMonitored"/>
                      has gone up <xsl:value-of select=
                                  "$U/RaiseFactor"/>!
                  </subject>
                </sendto>
           </xsl:template>
```

Note the XSLT instruction `<xsl:message>`, used for writing in an alert box or in a log file; note also how the argument of the command `xsl:if` can be interpreted as the rule condition, and element `xsl:message` as the rule action. In general, however, conditions and actions may freely interleave, as shown in the following examples.

3.1.2 Personalizer

Personalization, or one-to-one Web site delivery, is attracting increasing interest due to its strong potential. Personalization is typically preceded by the acquisition of profile information about the user, and makes use of this information to change the content of the Web page being presented, e.g., by proposing banners or by presenting "favorite goods" matching the user's interest. Sometimes personalization may be implicit, as an effect of simple clickstream analysis. In the sequel, we show a rule that personalizes the "look and feel" of a Web page by rendering it according to three different styles (junior, standard, senior), based on the age of the person who is doing the login.

```
CREATE RULE Personalize
PRIORITY: 0
EVENT:   <xsl:variable name="U"   select="User">
         <rule:login  select="$U"/>
COND_ACT: <xsl:template  match="$U" >
                <xsl:variable name="age_user" select=
                 "$U/age"/>
                <xsl:choose>
                   <xsl:when  test="$age_user &lt;= 27"  >
                      <xsl:include href="junior.xsl"/>
                   </xsl:when>
                   <xsl:when test="$age_user >
                    27 and age_user &lt; 55" >
                     <xsl:include href="standard.xsl"/>
                   </xsl:when>
                   <xsl:otherwise>
                     <xsl:include href="senior.xsl"/>
                   </xsl:otherwise>
                </xsl:choose>
         </xsl:template>
```

We deliberately kept this example simple, for illustration purposes, but personalization can take place in many ways, e.g., giving different interfaces to each individual user, depending explicitly on user preferences or implicitly on the ca-

pabilities of the device used. For instance, the `junior.xsl` style applied to young users is:

```
<xsl:stylesheet version="1.0" xmlns:xsl=
 "http://www.w3.org/1999/XSL/Transform"
 id="junior">
  <xsl:template match="/">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>
  <xsl:template match="font">
    <font color=yellow size=14/>
    <img src="junior.gif" align=left/>
  </xsl:template>
 </xsl:stylesheet>
```

3.1.3 Classifier

Another classic application of active rules is the classification of sets of instances within well-defined subsets. For instance, a classifier can be used to check incoming messages, trashing some of them or putting the other ones in separate input boxes to be read later.

As an example, a simple classifier of incoming mails is provided by the following rule that monitors the insertion of mail messages in the XML node `Default_service_mail`. The classifier selects the ones containing "I love you" in the subject and deletes them; the others are copied (with all their elements and subelements) in the node `toProcess` and become ready to be processed by the user.

```
CREATE RULE MailClassify
PRIORITY: 0
EVENT:   <xsl:variable   name=M
                         select=
                         "Default_service_mail/mail" />
         <rule:insert  select="$M" />
COND_ACT:<xsl:template  match="$M" >
           <xsl:variable name=subject select="subject" />
           <xsl:choose>
            <xsl:when  test=
             "contains($subject, "I love you")/>
                <xsl:call-template  name="DropIt" >
                   <xsl:with-param name="MDrop"
                    select="$M"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:call-template   name="ProcessIt" />
                 <xsl:with-param name="MProcess"
                  select="$M"/>
            </xsl:otherwise>
           </xsl:choose>
        <xsl:template>
        <xsl:template name="DropIt" >
            <xsl:param name="MDrop" />
        </xsl:template>
        <xsl:template name="ProcessIt"  >
           <xsl:param name="MProcess" />
           <toProcess>
              <xsl:copy-of   select="$MProcess"/>
           </toProcess>
        </xsl:template>
```

4 Active rules in Lorel

Lorel was originally designed for querying semistructured data and has now been extended to XML data; it was conceived and implemented at Stanford University [AQ*97]. It is a user-friendly language in the SQL/OQL style, includes a strong mechanism for type coercion, and permits very powerful path expressions, extremely useful when the structure of a document is not known in advance [GMW99]. Lorel represents well the requirements of rich XML query languages, such as XML-QL [DF*98], XML-GL [CC*99], and Quilt [CRF00].

4.1 Example: view maintenance system

We show Lorel at work on view maintenance, a classic application of active rules that could also be very relevant for document management and hence for e-service applications.

A view is simply a derived document, whose content can be assembled by querying other documents. The view maintenance service is used to maintain the view aligned with the original document when they are changed.

We describe several rules implementing a very useful service, targeting the automatic conversion and maintenance of a short order form (that represents the view) with respect to an extended order form (that embodies the original document).

The document base includes *orders*, containing full orders, and their view *short orders*, containing summaries of the main features of the orders.

```
<order>
 <ordenNr>234587</>
 <seller>
   <seller-url>www.Amico.com</>
   <seller-mail>Amico@shops.com</>
 </seller>
 <customer>
  <first-name>Angela</>
  <last-name>Bonifati</>
  <age>27</>
  <address>Via Bazzini, 3 - 20131 Milano</>
  <e-mail>bonifati@elet.polimi.it</>
 </customer>
 <items>
   <item>
     <item-type>Toys</>
     <name>Sound 'n Lights Monitor</>
     <description>Walky Talky devices</>
     <brand>Italian Toys</>
     <size>20x17x34</>
     <time-to-ship>3 days</>
     <shipping-weight>3 pounds</>
     <quantity>1</>
     <list-price>$29.99</>
     <our-price>$19.99</>
     <you-save>$10</>
   </item>
   .....
   .....
  </items>
 <total>$200</>
</order>

<short-order>
 <orderNr>234587</>
 <items>
   <item>
     <item-type>Toys</>
     <name>Sound 'n Lights Monitor</>
     <quantity>1</>
     <price>$19.99</>
   </item>
   .....
   .....
  </items>
 <total>$200</>
</short-order>
```

We assume that orders may change due to the insertion, deletion, or update of newly bought items. Then, rule *R1* reacts to an insert event on the `<order>` element of the order; the `<orderNr>` element and the `<item>` elements of the order are extracted and enclosed in the `<short-order>` element.

```
CREATE RULE R1
PRIORITY: 0
EVENT:    insert (order O)
COND_ACT: select xml(short-order: select OR, OI, OT
                                  from   O.orderNr OR,
                                         O.#.item OI,
                                         O.total OT
                                  with   OI.#)
```

Rule *R2* reacts to the arbitrary change of any element contained in the <items> element, by selecting the corresponding element in the short order and then substituting it with the new XML tree rooted in <items> in the order document.

```
CREATE RULE R2
PRIORITY: 0
EVENT:     change (order.#.items  I)
COND_ACT:  update SI:=I
           from short-order S, order O, S.#.items  SI,
           where O.#.items = I  and
                 O.orderNr = S.orderNr
```

As a variant of this rule, rule *R2'* reacts to an update event on the specific <brand>, <size>, and <time-to-ship> attributes of the <item> element of the order; the short-order document is updated by replacing the <item> element of the corresponding short order with the XML tree rooted in the <item> element of the order.

```
CREATE RULE R2'
PRIORITY: 0
EVENT:     update (order.#.item{I}.brand)
           or update (order.#.item{I}.size)
           or update (order.#.item{I}.time-to-ship)
COND_ACT:  update SI:=I
           from short-order S, order O, S.#.item SI
           where S.#.item = I and
                 O.orderNr = S.orderNr and
                 I.name =  SI.name
```

Rule *R3* reacts to a delete event on the <item> element of the order; the deleted items are deleted from short orders as well.

```
CREATE RULE R3
PRIORITY: 0
EVENT:     delete (order.#.item I)
COND_ACT:  update SI-=I
           from short-order S, order O, S.#.item  SI
           where O.#.item = I  and
                 O.orderNr = S.orderNr  and
                 I.name = SI.name
```

A suitable collection of these rules, built according to well-known rule generation algorithms (e.g., [CW91]), is capable of maintaining the view; when view maintenance is too complex, a radical approach consists of recomputing invalidated views from scratch. This is also typically managed by active rules, which schedule the refresh program.

5 Issues of active rules management

Several issues emerge from the above examples and from the early study of the solutions that are required in order to support them. Most of these issues have already been considered in the context of active rules for database systems but take a different flavor in the context of XML. In this section, we describe succinctly the main problems and in Sect. 6 we will analyze the impact that each of these aspects has on the design of an active rule system.

5.1 Perceiving changes

Monitoring the events, i.e., perceiving the changes to XML nodes that trigger the rules and the portions of XML resources that are affected by the changes, is one of the main difficulties of building an active XML repository.

Two cases are possible. The first alternative is to assume a tight integration of the event monitor with the DOM system and then let event detection be associated with the monitoring of method calls, as proposed in [AC*99]. The recent DOM Level-2 specification [DOM00] includes the definition of an event model which is adequate for our needs. Tight integration is definitely the simplest solution.

However, in the context of e-services, XML manipulation is performed by means of a heterogeneous collection of software systems (e.g., document editors, mailers), and often occurs outside of the XML repository. This means that it may not be available an internal component monitoring all the updates. The second alternative is to assume a loose integration between the event detector and the XML resources. The identification of the events must then be based on a comparison between the initial and current version of the target XML resource, and not on the constant monitoring of state changes.

Detecting changes on semi-structured information is a general problem, considered in [CR*96,CG97,CAW98,Ch99]. The problem can be solved by executing an *XML-diff* algorithm, which produces an optional (i.e., undefined on certain nodes) one-to-one identity relationship between nodes of the old and new version of the XML resource, so that any two related nodes must be considered as two versions of the same node. Base events can be immediately derived from this relationship: if a node of the new version is not related to a node of the old version, then the node is inserted; if a node of the old document is not related to a node of the new version, then the node is deleted; finally, if two related nodes of the new and old version have identical label and different value, then the node is updated.

The use of node identifiers for storing XML documents, available in all the DOM implementations of XML, provides the identity relationship between nodes as discussed above. However, if we assume that XML resources may be changed as an effect of the interaction with arbitrary e-services, then, to be coherent we must assume that such an identity relationship may not be supported. Quite luckily, references [CR*96, CG97] address exactly this problem, and present algorithms for the optimal matching of two hierarchical structures without assuming the matching of object identifiers.

Algorithms in [CR*96,CG97] produce their outputs in the form of *edit scripts*, i.e., of sequences of records (similar to logs) representing insertions and deletions of XML elements and insertions, deletions or updates of attribute and CDATA values. Edit script detection algorithms assume that, when a fragment of XML graph is inserted or deleted, insertions are done top-down (starting from the root of the fragment) and deletions are done bottom-up (starting from leaves). For instance, the edit script describing the change to orders as illustrated in Sect. 4.1 first associates an identifier with nodes <order>, <seller>, <customer>, <item> and then describes the change of an order item by means of the edit script. Once edit scripts are available, event detection is performed by simple lookup programs on the edit script, that

detect whether the changes on the XML document include the event being monitored by a given rule.

Some implementations of XML-diff algorithms are already available. For instance, IBM Alphaworks is offering a *XMLTreeDiff* package (*http://www.alphaworks.ibm.com*), which consists of a set of Java beans computing a difference between two DOM trees and returning the result (the edit script) in an XML format.

5.2 Phases of rule execution

Prior to discussing the various options for rule execution, we introduce a preliminary terminology. When one of the events of a rule occurs, we say that the rule is *triggered*; when the condition is being evaluated, we say that the rule is *considered*; when the rule consideration succeeds, we say that the rule is *executed*, meaning that the action part of the rule is executed. Note that a rule can become triggered and then not be executed (because consideration fails); normally, this is the semantics of rules even if conditions and actions are merged, as in the proposed examples. However, a triggered rule should always be considered.

5.3 Conflicts

In a generic active rule system, several rules may be triggered at the same time; all the triggered rules are the elements of the *conflict set* and are called *conflicting* rules. There are several situations where conflicts may emerge. The simplest case occurs when there are several rules associated with the same event: every time the event is generated, all the rules sharing the event will become part of the conflict set. Conflicts may also arise when an operation produces different elementary actions, whose events are associated with different rules. Indeed, when events are detected by a comparison between the current and the previous state of the XML resource, all the rules triggered by the events in the edit script will be conflicting.

To solve conflicts, an order may be defined on rules; the order may be partial or total, and it may depend on an explicit definition of priorities among rules or on an implicit ordering mechanism (e.g., based on the time of rule definition); the rule order is used by the rule engine to select the rule to consider among all those contained in the conflict set. By analogy with the SQL3 standard, we assume that one rule among the highest priority ones should be extracted from the conflict set and should be considered; if the rule execution generates events triggering higher priority rules, the rule should be suspended and resumed only when there will be no more higher priority triggered rules.

The conflict resolution policy defined for XSLT is quite intricate; in addition, each conflict resolution step is managed by selecting one rule and discarding all the other ones. This is in conflict with the general requirement that all triggered rules should be considered. Therefore, although syntactically XSLT is very suitable to express reactive computations, semantically the execution of rules should take place under the control of a rule engine with its own conflict resolution policies.

5.4 Phantoms

A critical problem is the impact that the execution of a rule may have on the events that have yet to be considered by other rules; two rules, for instance, may be triggered on the insertion of a certain node and the action of the first rule may delete that node: the second rule is then triggered by an event which has "disappeared". The solution we propose exploits the fact that each rule execution is bound to an event; therefore, it is possible to check that the corresponding XML element or attribute is still part of the database, before requesting an update on it.

5.5 Rule granularity

Two different granularities can be used for the execution of rules: set-oriented and node-oriented.

- Set-oriented granularity assumes that each rule considers at once all the event instances that are relevant for the rule: the variable possibly appearing in the event part denotes a set of events, and the consideration of the rule evaluates all the different event instances caused by the application of the same set-oriented primitive in a single consideration and execution. In the case in which events are extracted from an edit script, all the events of the same kind should be assembled.
- Node-oriented granularity instead considers each event instance separately, and the variable in the event part is associated with a single event. In the case in which events are extracted from an edit script, a rule should consider the events in the order in which they appear in the edit script.

Set-oriented rules can typically be processed in a more efficient way, because the costs incurred in the setup of a rule are shared among all the event instances treated by the rule execution. The performance advantage of the set-oriented solution can be considerable [AGW97,AKG96]. Node-oriented rules typically are easier to define, particularly when the rule action requires the access to the transition values (introduced by the functions `old` and `new`).

The SQL3 standard for relational databases admits both set-oriented and node-oriented (called tuple-oriented) triggers, which can arbitrarily intermix. The definition of an active rule system for XML information should consider this dimension and has to clarify if it permits only one of the two granularities, fixed for the rule engine, or if the rule designer should specify the granularity to use at the level of the single rule; in the latter case, the choice between set-oriented and node-oriented granularity should take into consideration the characteristics of the application.

5.6 Support for transition values

Active relational systems typically permit the retrieval, in the context of a rule, of the database state before and after the event which triggered the rule. This service is typically offered giving access to a pair of variables called `old` and `new`, which return the state of the object on which the event occurred, before and after the event, respectively. It is important

to observe that `old` and `new` variables will have a single value associated when rules have a node-oriented granularity, and a set of values with a set-oriented granularity.

5.7 Rule execution

Similar to the SQL3 standard [CKM99], we assume the most general situation with both set-oriented and node-oriented rules. Rules with a different granularity can be mixed and the ordering specified by priorities applies to set-oriented and node-oriented rules indistinctly. The rule engine performs the following iterative behavior:

1. While there are rules in the conflict set do:
   (a) Extract one of the highest priority rules from the conflict set (we say that the rule is *de-triggered*).
   (b) Execute that rule according to its granularity.
   (c) Whenever a rule executes an action, add to the conflict set the rules which are triggered by the action. If the currently running rule triggers some rules at higher priority, its execution is suspended and the highest priority rule among the triggered ones is executed.

Eventually, the algorithm may terminate in a state where no rule is triggered; we call this a *quiescent state* of the rule engine. However, as discussed next, there can be situations where the rules trigger each other indefinitely, without producing a quiescent state.

5.8 Rule interaction

Two rules are called *cascading rules* when the action of the former produces an event that triggers the latter. As an example of cascading rules, consider again the rule *R3* and then consider the rule *R4* that reacts to a delete event on the `<item>` element of the short order, and evaluates the new total in the short order. Clearly, the two rules cascade; their combined execution removes deleted items from the short orders and then recomputes the requested total amount.

Note the use of the predicate `old` to denote the transition value required in order to perform the row-level computation. Obviously, such a transition value is currently unavailable in Lorel and should be supported by a Lorel rule engine.

```
CREATE RULE R4
PRIORITY: 0
EVENT:     delete (short-order.#.item I)
COND_ACT:  update S.total:= S1.#.total - SI.total
           from  short-order S, short-order S1
                 old(S1.#.item) SI
           where S1.orderNr = S.orderNr and
                 S.#.item = I and
                 I.name =  SI.name
```

5.9 Termination, confluence, observable determinism

The properties of termination, confluence, and observable determinism of a set of rules were originally defined in [AWH95] for active databases; these definitions are naturally extended to XML active systems. Let us define an *XML repository state* as the set of XML resources and their organization in a given time instant. Then:

- A set of active rules *terminates* if, for any activity and initial XML repository state, the execution of the rule engine always produces a quiescent state, i.e., one in which no rule is triggered.
- A terminating set of active rules is *confluent* if, for any activity and initial XML repository state, the execution of the rule engine always produces the same quiescent state.
- A confluent set of active rules is *observably deterministic* if, for any activity and initial XML repository state, the execution produces identical observable outputs, presented in the same order.

The problems of termination and confluence have been widely studied in the field of databases [AWH95, BCP98, BW94, KU96, vS93, Wi96]; in general, the above results apply also to active rules for XML. In particular, acyclicity of the triggering graph of active rules for databases is a sufficient condition for termination [AWH95], where the triggering graph is defined as a graph whose nodes are the rules and whose arcs are drawn from rule R_i to rule R_j when rule R_i triggers rule R_j.

We also note that confluence is often compromised when instance-oriented rules are present, because the final result usually depends on the order of binding extraction from edit scripts, which in turn depends on the order of execution of element updates. However, many applications require termination but do not require confluence.

5.10 Edit-script independence

The above properties are defined regardless of the notion of edit script, but the edit script considered by a given rule engine and in a given rule execution is one of the many equivalent edit scripts that can be produced by an XML-diff algorithm, which operates with a given optimization strategy for choosing among equivalent edit scripts. For instance, an edit script consisting of a single update record of a given node could be equivalently described by a sequence of updates to that node, or by a sequence of a delete and an insert of that node. Therefore, another important property, called *edit-script independence*, complements the properties of termination, confluence, and observable nondeterminism.

- A set of active rules is *edit-script independent* if, for any activity and initial XML repository state, the behavior of the rules does not depend on the choice of the edit script describing the change of the XML resources.

The property of edit-script independence is orthogonal to termination and confluence. For instance, it is possible to envision a system of identical rule pairs, where each pair is associated to any event that could possibly be part of an edit script. Let us assume that the rules in the pair would produce different states and the execution of any one rule in the pair would produce a state where the other rule would not be executed. Such a system is edit-script independent, because the choice of edit script has no impact on the rule behavior (every edit script will trigger at least a pair of rules), but the system is not confluent. Similarly, the system could be edit-script independent and not terminating (e.g., due to a single looping rule). This property is novel and requires to be understood in all its subtleties and consequences; we are currently studying it in depth.

6 Implementation of active document systems

We briefly discuss the XSLT-based or Lorel-based implementation of active rule systems, touching on some of the issues that were introduced in the previous section.

6.1 Implementation of active rules in XSLT

XSLT technology is an important part of XML solutions, as it is responsible of the presentation of XML information delivered by Web servers. Even if the language has been very recently defined, many XSLT-based products are already on the market, and many are going to be launched in the near future. Several open-source implementations are also available, in different languages. An XSLT-based realization of active services can be greatly facilitated if it effectively exploits the potential of industrial quality software.

A minimal architecture for an XSLT-based active system should include the following components: an offline event detector, a processor of XSLT templates (an *XSLT processor*), and a rule engine. The first two components are already available and do not need to be re-implemented, but rather to be adapted. The rule engine must be built from scratch, but it is a relatively simple component, whose main task is to invoke computations performed by the other two components.

A possible interaction of these modules would consist of the initial invocation of the event detector for building the initial edit-script, followed by repeated invocations of the XSLT processor, one for each rule instance, controlled by the rule engine. The XSLT processor should be integrated with a DOM system for changing the content of the underlying document base; when the DOM system supports Level 2 events [DOM00], event listeners can capture modification events on the document and thus extend the edit script, otherwise the XSLT processor should be modified so as to keep the edit script current. The execution would of course terminate on a quiescent state.

The offline event detector can be introduced into the architecture encapsulating an XML-diff service to make it produce an XML representation of events that could be used by the rule engine to drive rule execution. Conflicts among rules cannot be solved using the standard execution model of XSLT templates, because it is based on a selection mechanism which applies only one rule to a node and then the choice of the rule is based on the specificity of the path identifying the node and on the order of rule definition, without an explicit ordering mechanism. Given these constraints, it is better to delegate to the rule engine the full responsibility for rule consideration and let it invoke the XSLT processor for a single rule instance.

Phantoms may occur, but their treatment does not impact on the rule engine; they need to be carefully considered by the rule designer in order to avoid anomalous behaviors.

Rule granularity can both be set-oriented and node-oriented, depending on the number of event instances that are passed as a parameter to the XSLT processor. When the rule engine invokes the XSLT processor on rule instances associated to single events, the behavior would be node-oriented.

Overall, all the issues described can be managed in an XSLT solution, and given the attention that XSLT is currently receiving, an XSLT-based approach can be a very interesting strategy for the realization of active services for XML.

6.2 Implementation of active rules in Lorel

The Lorel prototype has a more classical architecture, consisting of the typical components of database servers. Therefore, the implementation of active rules in Lorel should follow the typical approach that was used for relational engines, e.g., [Wi96], where the condition-action parts are compiled and executed as any other query. It is, however, necessary to completely design and add those features, like event management and the rule engine, needed to achieve the functionality of an active system. This solution offers the greatest flexibility, but it also requires greater efforts than the XSLT-based solution. The current implementation of Lorel is a nice prototype developed in an academic environment, and it offers innovative features with respect to the data model and the operations that can be used to manipulate the data, but it does not guarantee industrial-level performance and robustness. We have already started the design of an active system on top of Lorel, and have encountered obstacles due to the lack of robustness of the system.

The addition of active services to Lorel can be a representative example of the components that would be required in order to support reactive processing within XML-based data managers.

7 Conclusions

This paper presents a general framework for the introduction of active rules in the context of XML. In this process, particular attention should be paid to the work done in recent years for active database systems, so as to take maximum advantage of those technological features that have already been fully understood and instead concentrate research and development on the features which are novel or need important redesigns.

As experience in active databases has demonstrated, active rules are a powerful but delicate mechanism. Thus, we do not suggest the adoption of active rules as a tool for the final implementation of e-services, but rather we envision their use as rapid prototyping tools enabling a fast deployment of new e-services, in order to evaluate their interest and potential. When a prototype has demonstrated the validity of the solution, a specialized subsystem can be designed which focuses on that particular service. It may still use the active rule engine as the implementation environment, however, offering the automatic generation of active rules for the specific application; or it may use an ad hoc solution designed from scratch, offering greater performance.

This is similar to what has happened to active rules in database systems. For instance, replication services have now become an essential ingredient of data server architectures (e.g., for enabling data warehousing); their initial implementation was realized using triggers and this solution is still adopted by some vendors, while most of them moved to more efficient methods, tightly integrated with the database engine (e.g., based on logs). Similarly, we envision that the introduction of active rules for XML data will enable a fast prototyping of e-services for XML data. In the paper, we have presented e-services such as alerting, personalization, classification, and view management; additional applications include integrity maintenance, workflow management, process interoperability, and many others.

As future work, we are interested in the development of flexible conflict resolution policies that may enable the easy modeling of arbitration among e-services. Such policies should be able to offer a high degree of dynamicity and reconfigurability, and would permit fine-grain resolution of conflicts among e-services.

References

[AC*99] Abiteboul S, Cluet C, Mignet L, Amann B, Milo T, Eyal A (1999) Active views for electronic commerce. In: Proc. 25th VLDB, Edinburgh, UK, September 1999, pp 138–149

[AGW97] Adelberg B, Garcia-Molina H, Widom J (1997) The Strip rule system for efficiently maintaining derived data. In: Peckham J (ed) Proc. ACM SIGMOD Int. Conf. on Management of Data, Tucson, Ariz., June 1997, pp 147–158

[AKG96] Adelberg B, Kao B, Garcia-Molina H (1996) Database support for efficiently maintaining derived data. In: Proc. 5th Int. Conf. on Extending Database Technology, Avignon, France, March 1996, Lecture Notes in Computer Science, vol. 1057. Springer, Berlin Heidelberg New York, pp 223–240

[AQ*97] Abiteboul S, Quass D, McHugh J, Widom J, Wiener J (1997) The Lorel query language for semistructured data. Int J Digital Libr 1(1):66–88

[AWH95] Aiken A, Widom J, Hellerstein JM (1995) Static analysis techniques for predicting the behavior of active database rules. ACM TODS 20(1):3–41

[BC00] Bonifati A, Ceri S (2000) Comparative analysis of five XML query languages. ACM Sigmod Record 29(1):68–79

[BCP96] Baralis E, Ceri S, Paraboschi S (1996) Modularization techniques for active rules design. ACM TODS 21(1):1–29

[BCP98] Baralis E, Ceri S, Paraboschi S (1993) Compile-time and runtime analysis of active behaviors. IEEE TKDE 10(3):353–370

[BW94] Baralis E, Widom J (1994) An algebraic approach to rule analysis in expert database systems. In: Proc. 20th VLDB, Santiago, Chile, September 1994, pp 606–617

[CAW98] Chawathe S, Abiteboul S, Widom J (1998) Representing and querying changes in semistructured data. In: Proc. ICDE98, Orlando, Fla., USA, February 1998, pp 4–13

[Ch99] Chawathe S (1999) Comparing hierarchical data in external memory. In: Proc. 25th VLDB, Edinburgh, UK, September 1999, pp 90–101

[CC*99] Ceri S, Comai S, Damiani E, Fraternali P, Paraboschi S, Tanca L (1999) XML-GL: a graphical language for querying and restructuring WWW Data. In: Proc. 8th Int. World Wide Web Conference, WWW8, Toronto, Canada, May 1999. http://www8.org/fullpaper.html

[CCW00] Ceri S, Cochrane R, Widom F (2000) Practical applications of triggers and constraints: success stories and lingering issues. Invited for the 10-year paper award. In: Proc. 26th VLDB, Cairo, Egypt, September 2000

[CG97] Chawathe SS, Garcia-Molina H (1997) Meaningful change detection in structured data. In: Proc. ACM SIGMOD Conf. 97, Tucson, Ariz., May 1997, pp 26–37

[CKM99] Cochrane R, Kulkarni KG, Mendonça Mattos N (1999) Active database features in SQL3. In: Paton N (ed) Active Rules in Database Systems. Springer, Berlin Heidelberg New York, pp 197–219

[CRF00] Chamberlin D, Robie J, Florescu D (2000). Quilt: an XML query language for heterogeneous data sources. In: Proc. WebDB 2000, Dallas, Tex., May 2000

[CR*96] Chawathe SS, Rajaraman A, Garcia-Molina H, Widom J (1996) Change detection in hierarchically structured information. In: Proc. ACM SIGMOD Conf. 96, Montreal, June 1996, pp 493–504

[CW91] Ceri S, Widom J (1991) Deriving production rules for incremental view maintenance. In: Proc. 17th VLDB, Barcelona, Spain, September 1991, pp 577–589

[De98] DeRose SJ (1998) XQuery: a unified syntax for linking and querying general XML. In: Proc. Query Languages workshop (QL98), Boston, Mass., December 1998. In [Mar98]

[DOM00] The Document Object Model (DOM) Level 2 specification. W3C Candidate Recommendation, 10 May 2000. http://www.w3.org/TR/DOM-Level-2

[DF*98] Deutsch A, Fernandez M, Florescu D, Levy A, Suciu D (1998) XML-QL: a query language for XML. In: Proc. Query Languages workshop (QL98), Boston, Mass., December 1998

[GMW99] Goldman R, McHugh J, Widom J (1999) From semistructured data to XML: migrating the Lore data model and query language. In: Proc. 2nd Int Workshop on Web and Databases (WEDB99), Philadelphia, Penn., June 1999

[KU96] Karadimce AP, Urban SD (1999) Active rule termination analysis: an implementation and evaluation of the refined triggering graph method. In: J Intell Inf Syst, 12(1):27–60

[Mar98] Marchiori M (1998) Proc. Query Languages Workshop (QL'98), Boston, Mass., December 1998. Papers available at: http://www.w3.org/Tands/QL/QL98

[RLS98] Robie J, Lapp J, Schach D (1998) XML Query Language (XQL). In: Proc. Query Languages workshop (QL98), Boston, Mass., December 1998. In [Mar98]

[SLR98] Schach D, Lapp J, Robie J (1998) Querying and transforming XML. Proc. Query Languages workshop, Boston, Mass., December 1998. In: [Mar98]

[vS93] Van der Voort L, Siebes A (1993) Termination and confluence of rule execution. In: Proc. 2nd Int. Conf. on Information and Knowledge Management, Washington D.C., November 1993

[WC96] Widom J, Ceri S (1996) Active Database Systems. Morgan Kaufmann, San Mateo, Calif., USA

[Wi96] Widom J (1996) The Starburst active database rule system. IEEE TKDE 8(4):583–595

[W*98] W3C XSL Working Group (1998) The query language position paper of the XSL working group. In: Proc. Query Languages workshop, Boston, Mass., December 1998. http://www.w3.org/TandS/QL/QL98/ pp/xsl-wg-position.html

[XML98] eXtended Markup Language (XML) Specification. 1.0, February 1998. http://www.w3.org/XML

[XSc00] XML schema definition language – seventh working draft. September 2000

[XPa99] XML path language (XPath) specification. W3C Recommendation, November 1999. http://www.w3.org/TR/xpath

[XSLT99] Extensible stylesheet language transformations (XSLT) specification (Version 1.0). W3C Recommendation, November 1999. http://www.w3.org/TR/WD-xsl