Event-Condition-Action Rule Languages for the Semantic Web

George Papamarkos, Alexandra Poulovassilis, Peter T. Wood

School of Computer Science and Information Systems, Birkbeck College, University of London, London WC1E 7HX
email: {gpapa05,ap,ptw}@dcs.bbk.ac.uk

Abstract. The Semantic Web is based on XML and RDF as its fundamental standards for exchanging and storing information on the World Wide Web. Event-condition-action (ECA) rules are a natural candidate for supporting reactive functionality on XML or RDF repositories. In this paper we describe a language for ECA rules on XML and a prototype implementation of this language. We also discuss some preliminary ideas regarding a language for ECA rules operating on a graph/triple representation of RDF, and we describe the architecture of a distributed deployment of such RDF ECA rules.

1 Introduction

XML and RDF are becoming dominant standards for storing and exchanging information on the World Wide Web. With their increasing use in dynamic applications such as e-commerce and e-learning [9, 10, 14, 15, 1, 19, 16, 22], there is a need for the support of reactive functionality on XML and RDF repositories. Event-condition-action (ECA) rules are a natural candidate for this. ECA rules automatically perform actions in response to events provided that stated conditions hold. They allow an application's reactive functionality to be defined and managed within a single rule base rather than being encoded in diverse programs, thus enhancing the modularity and maintainability of the application. Also, ECA rules have a high-level, declarative syntax and so are amenable to analysis and optimisation techniques which could not be applied if the same functionality were expressed directly in programming language code.

ECA rules have been used in many settings, including active databases [25, 20], personalisation and publish/subscribe technology [4, 9, 10, 12, 21], and specifying and implementing business processes [3, 11, 15]. An ECA rule has the general syntax

`on` *event* `if` *condition* `do` *actions*

The event part specifies when the rule should be triggered, the condition part is a query which determines if the database is in particular state, and the action part states the actions to be performed automatically if the condition holds. Executing a rule's actions may in turn trigger further ECA rules, and

the rule execution proceeds until no more rules are triggered. Non-termination of rule execution is generally a possibility and thus much research has focussed on the development of static rule analysis techniques for detecting possibly non-terminating rule sets; a practical solution to this problem adopted by commercial DBMS is to set a predefined upper limit on the number of recursive rule firings, and to abort a transaction if this is exceeded. More details on the foundations of ECA rules in active databases, and descriptions of a range of implemented active database prototypes can be found in [25, 20].

We begin this paper with a brief review of related work on ECA rules for XML. We then describe our language for specifying ECA rules on XML repositories, and present a prototype implementation of it. This language can be used for RDF data which has been serialised as XML but we are also exploring ECA rule languages for RDF that will operate on a graph/triple representation. We present an archetypal such language and also an architecture for distributed deployment of such RDF ECA rules. Along the way, we discuss directions of further work for both languages.

The work reported here is part of the ongoing SeLeNe project, which is investigating techniques for managing RDF repositories of educational metadata, and providing syndication, personalisation and notification services over this metadata (see `http://www.dcs.bbk.ac.uk/selene`).

2 ECA Rules for XML

In recent work [7, 6], we specified a language for defining ECA rules on XML data, based on the XPath and XQuery standards. We also developed techniques for analysing the triggering and activation relationships between such rules[1] and showed how these techniques can be used to detect possibly non-terminating sets of ECA rules. A number of other ECA rule languages for XML have also been proposed, although none of this other work has focussed on analysing the behaviour of the ECA rules.

Reference [9] discusses extending XML repositories with ECA rules in order to support e-services. Active extensions to the XSLT [27] and Lorel [2] languages are proposed which handle insertion, deletion, and update events on XML documents. Reference [10] discusses a more specific application of the approach to push technology where rule actions are methods that cannot update the repository, and hence cannot trigger other rules.

Reference [8] also defines an ECA rule language for XML. The rule syntax is similar to ours, but the rule execution model is different. In our case we treat insertions or deletions of XML fragments as "atomic" updates and ECA rule execution is invoked only after the completion of such an update, whereas in [8] such updates are broken up into a sequence of finer granularity requests each

[1] A rule r_i *may trigger* a rule r_j if execution of the action of r_i may generate an event which triggers r_j. A rule r_i *may activate* another rule r_j if r_j's condition may be changed from False to True after the execution of r_i's action. A rule r_i *may activate* itself if its condition may be True after the execution of its action.

of which may invoke the ECA rule execution. In general, these semantics may produce different results for the same initial update.

ARML [13] provides an XML-based rule description for rule sharing among different heterogeneous ECA rule processing systems. In contrast to our language, conditions and actions are defined abstractly as XML-RPC methods which are later matched with system-specific methods.

GRML [24] is a multi-purpose rule markup language for defining integrity, derivation and ECA rules. GRML uses an abstract syntax based on RuleML, leaving the mapping to a real language for each underlying system implementation. GRML aims to provide semantics for defining access over distributed, heterogeneous data sources for rule evaluation and allows the user to declare most of the semantics necessary for processing a rule, and to evaluate events and conditions coming from heterogeneous data sources.

Finally, [23] proposes extensions to the XQuery language [28] to incorporate update operations. These are more expressive than the actions supported by our ECA rule language since they also include renaming and replacement operations, and specification of updates at multiple levels of documents. Triggers are discussed in [23] as an implementation mechanism for deletion operations on the underlying relational store of the XML. However, provision of ECA rules at the "logical" XML level is not considered.

2.1 Our XML ECA Rule Language

An XML repository consists of a set of XML documents. In our XML ECA rule language, we use XPath [26] and XQuery [28] to specify the event, condition and actions parts of rules. XPath is used for selecting and matching fragments of XML documents within the event and condition parts while XQuery is used within insertion actions, where there is a need to be able to construct new XML fragments.

The event part of an XML ECA rule is an expression of one of the following two forms:

`INSERT` e

`DELETE` e

where e is an XPath expression which evaluates to a set of nodes. The rule is *triggered* if this set of nodes includes any node in a new XML fragment, in the case of an insertion, or in a deleted fragment, in the case of a deletion.

The system-defined variable `$delta` is available for use within the condition and actions parts of the rule, and its set of instantiations is the set of new or deleted nodes returned by e.

The condition part of a rule is either the constant `TRUE`, or one or more XPath expressions connected by the boolean connectives `and`, `or`, `not`. The rule *fires* if it is triggered and its condition evaluates to true.

The actions part of a rule is a sequence of one or more actions:

$$action_1;\ \ldots;\ action_n$$

where each $action_i$ is an expression of one of the following three forms:

`INSERT` r `BELOW` e `BEFORE` q

INSERT r BELOW e AFTER q
DELETE e

Here, r is an XQuery expression, e is an XPath expression and q is either the constant `TRUE` or an XPath qualifier.

In an `INSERT` action, the expression e specifies the set of nodes, N, immediately below which new XML fragment(s) will be inserted. These fragments are specified by the expression r. If e or r references the `$delta` variable, then one XML fragment is constructed for each instantiation of `$delta` for which the rule's condition evaluates to True. If neither e nor r references `$delta`, then a single fragment is constructed. The expression q is an XPath qualifier which is evaluated on each child of each node $n \in N$. For insertions of the form `AFTER` q, the new fragment(s) are inserted after the last sibling for which q is True, while for insertions of the form `BEFORE` q, the new fragment(s) are inserted before the first sibling for which q is True. The order in which new fragments are inserted is non-deterministic.

In a `DELETE` action, the expression e specifies the set of nodes which will be deleted (together with their descendant nodes). Again, e may reference the `$delta` variable.

Example 1. Consider an XML repository containing metadata about learning objects (LOs) available on the web, as well as personal metadata about users of these LOs. The XML document `los.xml` contains information about the LOs, and we show below some of the information held for a particular book, "Data On the Web". Under `annotations`, a new `review` is appended every time a user submits a review of the book.

```
<LOs>
  ..
  <LO type="book" title="Data On the Web">
    <subject>Computer Science</subject>
    <creator>S. Abiteboul</creator>
    <creator>P. Buneman</creator>
    <creator>D. Suciu</creator>
    <description>From Relations to Semistructured data and XML
    </description>
    <publisher>Morgan Kaufmann</publisher>
    <isbn>1-55860-621-Y</isbn>
    <annotations>
      <review>
        <reviewer>Teacher Education Review Panel</reviewer>
        <date>2002-10-20</date>
        <rating>9</rating>
        <description>
          This book gives a comprehensive, state-of-the art
          discussion of data models, query languages and ...
        </description>
      </review>
```

```
      <review>
        <reviewer>John Smith</reviewer>
        <date>2002-12-20</date>
        <rating>10</rating>
        <description>
          I found this a great book to learn about querying
          semi-structured data, which I didn't know much about.
        </description>
      </review>
    </annotations>
  </LO>
  ...
</LOs>
```

The XML document `users.xml` contains information about users, and we show below some of the information held for a particular user "Johnny Mnemonic". Users can subscribe to be notified of the latest review submitted for books in subjects that they are interested in, and this information is used to automatically update their personal metadata:

```
<users>
  ...
  <user id="217">
    <name>Johnny Mnemonic</name>
    <profession>student</profession>
    <subjects>
      <subject>Computer Science</subject>
      <subject>Mathematics</subject>
      <subject>Economics</subject>
    </subjects>
    <LOs>
      <LO type="book" title="Data On the Web">
        <isbn>1-55860-621-Y</isbn>
        <latest-review>
          <reviewer>John Smith</reviewer>
          <date>2002-12-20</date>
          <rating>10</rating>
          <description>
            I found this a great book to learn about querying
            semi-structured data, which I didn't know much about.
          </description>
        </latest-review>
      </LO>
      ...
    </LOs>
  </user> ...
</users>
```

Johnny Mnemonic is interested in "Computer Science" and the following rule replaces the current latest review (if there is one) of any Computer Science book in his personal metadata by a new review of that book:

```
ON INSERT document('los.xml')/LOs/LO/annotations/review
IF $delta/../../subject[.='Computer Science']
DO DELETE document('users.xml')/users/user[@id="217"]/LOs/LO
          [isbn=$delta/../../isbn]/latest-review;
   INSERT <latest-review>{$delta/*}</latest-review>
   BELOW  document('users.xml')/users/user[@id="217"]/LOs/
          LO[isbn=$delta/../../isbn]
   AFTER  isbn
```

Here, the system-defined `$delta` variable is bound to a newly inserted `review` node detected by the event part of the rule. The rule's condition checks that the subject of the book in question is Computer Science. The rule's action then deletes the existing latest review for this book within Johnny Mnemonic's metadata (if there is one) and inserts the new review.

Suppose now that the following update occurs, appending a new review for the "Data On the Web" book:

```
INSERT <review>
         <reviewer>Neo Anderson</reviewer>
         <date>2003-04-29</date>
         <rating>9</rating>
         <description>
           Very clearly written and very well-organised.
           Describes in detail all the ...
         </description>
       </review>
BELOW document('los.xml')/LOs/
      LO[isbn="1-55860-621-Y"]/annotations
AFTER TRUE
```

This update triggers the rule above, causing the replacement within Johnny Mnemonic's personal metadata of the previous review submitted by John Smith by the new review submitted by Neo Anderson.

As another example rule, the following rule removes the current latest review (if there is one) of a Computer Science book in Johnny Mnemonic's personal metadata if this review is removed from the list of reviews for this book (this rule assumes that each reviewer reviews a book only once):

```
ON DELETE document('los.xml')/LOs/LO/annotations/review
IF $delta/../../subject[.='Computer Science']
DO DELETE document('users.xml')/users/user[@id="217"]/LOs/
          LO[isbn=$delta/../../isbn]/
          latest-review[reviewer=$delta/reviewer]
```

We refer the reader to [7, 6] for a more detailed discussion of the syntax and semantics of our XML ECA rule language. Here, we next describe a prototype implementation.

2.2 A Prototype Implementation

Due to the current immaturity of existing XML repository products in supporting a sufficiently expressive update language, for this first prototype implementation we have used flat files and have exploited the functionality provided by the W3C DOM standard [29] for interacting with them. The architecture of our system is illustrated in Figure 1.

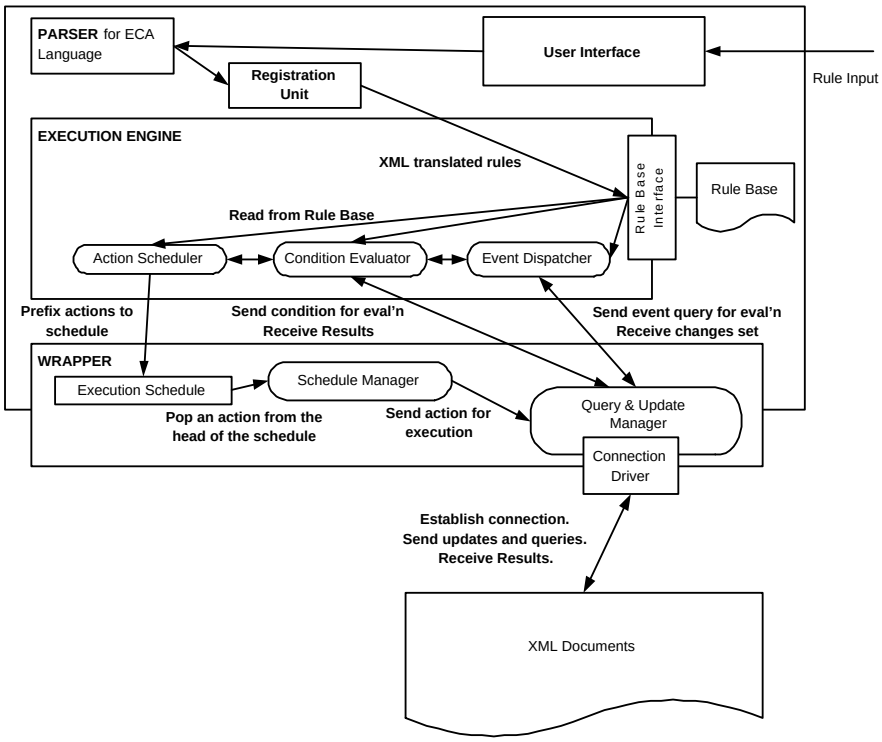

Fig. 1. ECA Engine Architecture

The *Parser* parses and checks the syntactic validity of a new rule. For construction of the parser, we have used the JavaCC lexer-parser generator. Valid rules are translated into an XML form and are added by the *Registration Unit* to the *Rule Base* (which is an XML file). Details about each rule are stored here, including its name, priority, and event, condition and action parts.

The *Execution Engine* encapsulates the rule processing functionality. In particular, the *Event Dispatcher*, *Condition Evaluator* and *Action Scheduler* implement these aspects of the rule processing, as we describe in more detail below. All of these components interface with the Wrapper in order to send and receive data to and from the underlying XML files.

The *Execution Schedule* contains a sequence of *updates* — these have the same syntax as rule actions except that they do not contain any $delta expressions within them. By "$delta expression" we mean an XPath expression (either stand-alone or possibly nested within another XPath expression) that starts with `$delta`. These portions of a rule's action part are replaced by the result of evaluating the expressions on the current document — see below.

The *Wrapper* interfaces with the XML files on disk. All update and query requests from the upper levels of the system pass through this component, which coordinates them. It undertakes to open files, submit queries and updates, and receive back results from them. The Wrapper performs these services by using the functionality of the Apache Xalan API. All queries are performed directly by using XPath. For deletions, we identify the set of nodes that will be deleted by using the XPath expression within the `DELETE` part of the request, and we then remove all the subdocuments rooted at the nodes identified. For insertions, we identify the set of nodes that will be affected by using the XPath expression within the `BELOW` part of the request, and we then add the fragment specified within the `INSERT` part as a new child of each of the nodes identified, placing it relative to the existing children according to the `AFTER` or `BEFORE` qualifier.

Rule execution begins with a request from the *Schedule Manager* to the *Query & Update Manager* to execute the update currently at the head of the schedule. In case of an insertion, the Query & Update Manager executes the update and annotates the newly inserted nodes, while in the case of a deletion it annotates the nodes to be deleted without executing the deletion yet[2].

Following the execution of the update, control then passes to the *Event Dispatcher*. This requests the Query & Update Manager to evaluate the XPath query of the event part of each rule that may be triggered by the update that was just executed. For each rule whose query result set contains annotated nodes (either newly inserted or about-to-be deleted), the Event Dispatcher creates a `changes` set containing these annotated nodes, and the rule is triggered.

The *Condition Evaluator* then requests the Query & Update Manager to evaluate the condition part of each triggered rule on the affected document, using as the evaluation context either the root node if there are no occurrences of `$delta` within a query, or each instance of the `changes` set otherwise. The rule's `delta` set is thus created, consisting of those members of its `changes` set for which the condition evaluates to true. If the `delta` set is non-empty, the rule fires and control is passed to the Action Scheduler to further process the rule. Otherwise, processing of this rule ends.

[2] The annotation of nodes is performed using non-DOM methods provided by Apache Xerces API that allow us to attach data to XML nodes without affecting the physical representation of the file.

The *Action Scheduler* reformulates a given rule's action(s) in order to eliminate any instances of `$delta` expressions within them. The reformulation algorithm performs the following steps for each node within the rule's `delta` set:

- Replaces the `$delta` variable in each of the $delta expressions by the current node of the `delta` set.
- Evaluates each of the modified $delta expressions with respect to the updated document.
- Replaces each $delta expression within the rule's action(s) by the corresponding result of the previous step.

The outcome of this reformulation is that one instance of the rule's action(s) is created for each node in the rule's `delta` set. These updates are now prefixed, in an arbitrary order, to the front of the schedule — this is known as Immediate scheduling, although other alternatives are also possible (see [20]). If multiple rules have fired as a result of the last update executed, then the updates that result from their actions are prefixed the schedule in order of the rules' specified priorities. Control then passes once more to the Schedule Manager and the cycle repeats. If the last update executed by the Query & Update Manager was a `DELETE`, then before control passes back to the Schedule Manager, the actual deletion of the annotated nodes is first performed.

2.3 Future Work

There is as yet no accepted standard update language for XML. If ECA rules are to be supported on XML repositories, then whatever standard eventually emerges, there is also the parallel issue of designing the event language to match up with this update language. Here we have seen how this was done in the context of our particular update language for XML. Elsewhere [6] it is shown how triggering and activation relationships can be detected for our particular XML ECA rules. In general, the ability to analyse and optimise ECA rules needs to be balanced against their complexity and expressiveness, and this issue also needs to be borne in mind in future developments in ECA rule languages for XML, and indeed for RDF.

It would be straightforward to extend our language to also support `REPLACE` events and actions, where the former would have the syntax

`REPLACE` e

and the latter the syntax

`REPLACE` e `BY` r

meaning that the set of nodes identified by e (and their subdocuments, if any) should be replaced by r. For example, the pair of actions in the first rule in Example 1 could be replaced by the single action

```
REPLACE document('users.xml')/users/user[@id="217"]/LOs/LO
        [isbn=$delta/../../isbn]/latest-review
BY      <latest-review>{$delta/*}</latest-review>
```

In general, our `INSERT` actions may result in non-determinism in the order in which a set of new fragments are inserted under a common parent, since the `BEFORE` and `AFTER` constructs only specify the ordering of new fragments with respect to the existing document content. It is an area of further work to extend our XML ECA language to capture ordering relationships between new fragments being inserted into a document.

At present we assume Immediate scheduling of rules that have fired, though it would be straightforward to also allow rules with other scheduling modes. However, the practical applicability and performance implications of these extensions is an area that requires further investigation.

Another important area is combining ECA rules with transactions and consistency maintenance in XML repositories.

3 ECA rules for RDF

The above language can be used for RDF which has been serialised as XML. However, we are also exploring ECA rule languages for RDF that will operate directly on a graph/triple representation. In our archetypal RDF ECA rule language, the event part of a rule is an expression of one of the following two forms:

`INSERT` e

`DELETE` e

where e is a path expression which again evaluates to a set of nodes.

The rule is triggered if this set includes any new node, in the case of an insertion, or any deleted node, in the case of a deletion. The system-defined variable `$delta` is again available for use within the condition and actions parts of the rule, and its set of instantiations is the set of new or deleted nodes returned by e.

The condition part of a rule is a query which may reference the `$delta` variable. Analogously to our XML ECA rule language, condition queries consist of conjunctions, disjunctions and negations of path expressions.

The actions part of a rule is a sequence of one or more actions, where each action is of one of the following two forms:

$[let-expressions$ `IN`$]$ `INSERT` $triples$

$[let-expressions$ `IN`$]$ `DELETE` $triples$

Here, $let-expressions$ is an optional set of local variable definitions of the form `let` $variable$ `=` e, where e is a path expression, and $triples$ is a set of triples of the form $(subject, predicate, object)$.

Example 2. Consider the two RDF graphs illustrated in Figures 2 and 3. Based on the application described in Example 1, the first shows the metadata relating to the "Data on the Web" book, while the second shows the personal metadata relating to user 128.

Suppose that user 128 wants to keep his set of reviews of Computer Science books up-to-date. If a new review of a Computer Science book is inserted, then the following ECA rule adds a new arc linking the new review into user 128's personal metadata:

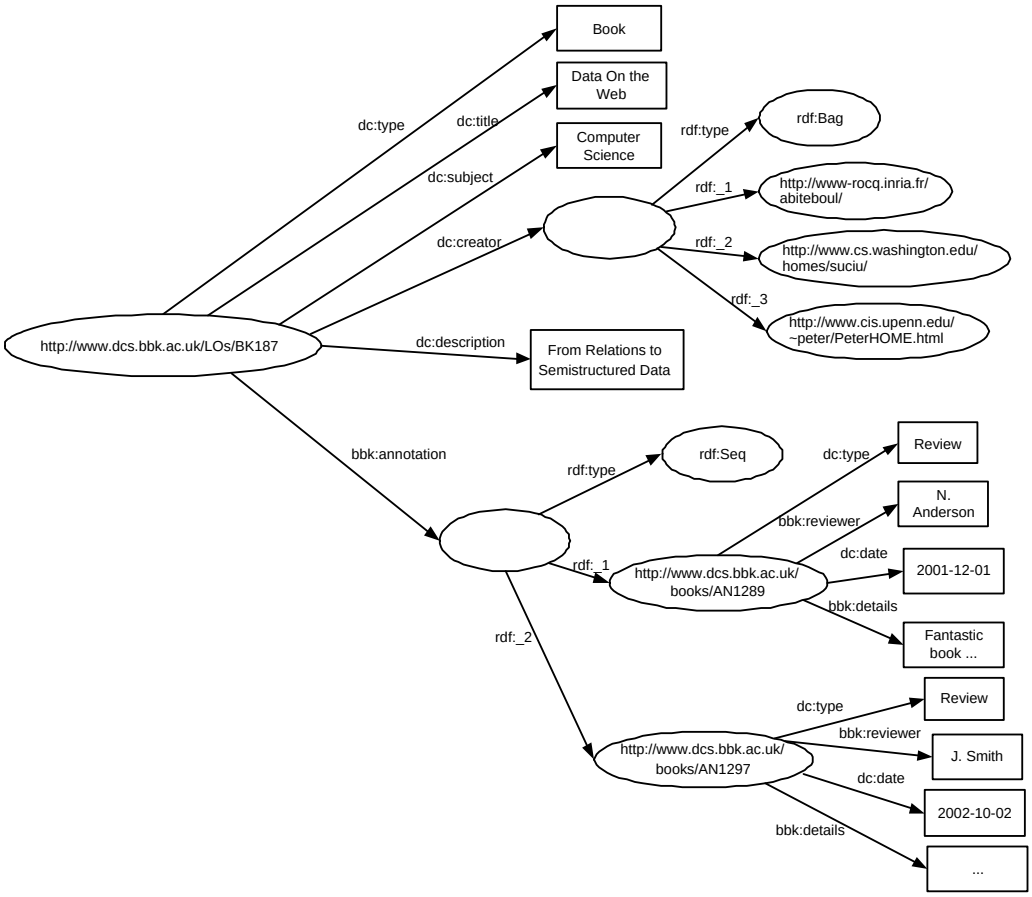

Fig. 2. Learning Object Metadata

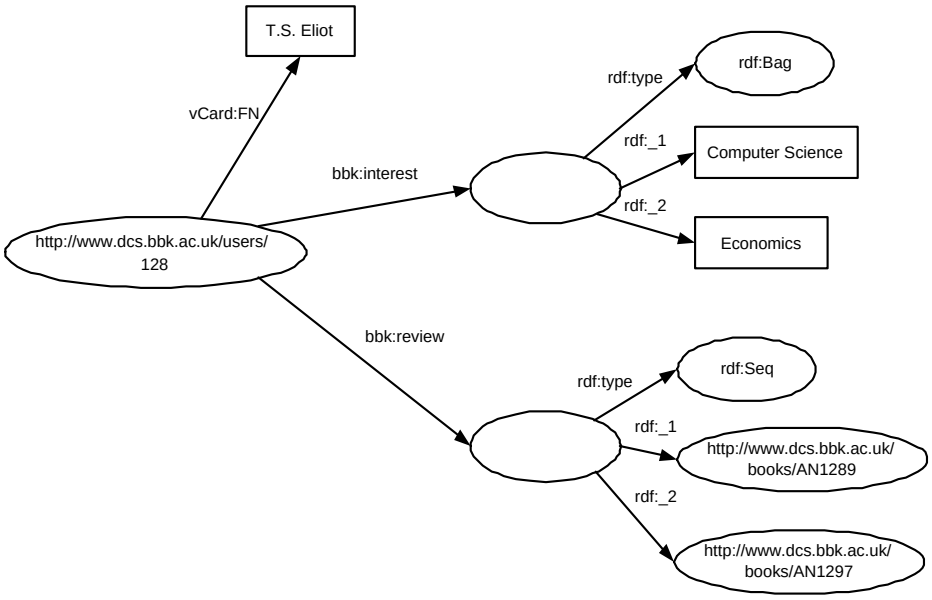

Fig. 3. User Metadata

```
ON INSERT resource()[child(dc:type)='Book']/child(bbk:annotation)/
          element()[child(dc:type)='Review']
IF $delta/parent()/parent()[child(dc:subject) = 'Computer Science']
DO LET $reviews = resource(``http://www.dcs.bbk.ac.uk/users/128'')
                  /child(bbk:review) IN
   INSERT ($reviews,seq++,$delta)
```

Here, the event part of the rule checks whether a new review has been added for a book (expressed in the syntax of RDFPath [17]). The condition part checks if the new review is for a Computer Science book. If so, the action part inserts the new arc between user 128's reviews collection and the new review (we use the syntax `seq++` to indicate an increment in the collection's element count).

As another example, if user 128 removes one of his interests, then the following rule removes from his personal metadata all arcs to reviews of learning objects on that subject:

```
ON DELETE resource(``http::://www.dcs.bbk.ac.uk/users/128'')
          /child(bbk:interest)/element()
IF TRUE
DO LET $reviews = resource(``http://www.dcs.bbk.ac.uk/users/128'')
                  /child(bbk:review);
       $review  = $reviews/element()[parent()/parent(bbk:annotation)
                  [child(dc:subject) = $delta]] IN
   DELETE ($reviews,seq?,$review)
```

Here, the event part checks if an interest of user 128 has been deleted. The condition part always holds. The `LET` part of the rule's action defines `$reviews` to be user 128's reviews collection and defines `$review` to be those reviews which relate to learning objects whose subject is the same as the deleted interest. Finally, the `DELETE` part will generate one triple to be deleted for each pair of distinct values of `$reviews` and `$review` (we use the syntax `seq?` to match any order of the review being deleted within the collection).

3.1 Future Work

There is as yet no standard query/update language for RDF and hence our RDF ECA language is even more prototypical than our XML ECA language. Some of the observations we made in Section 2.3 regarding the XML ECA language also apply here, namely the need to match up the event sub-language with the update sub-language, the need to balance expressiveness of ECA rules against the ability to analyze and optimize them, the possibility of a variety of scheduling modes beyond Immediate rule scheduling, and combining ECA rules with transactions and consistency maintenance in RDF repositories.

For the immediate future, we plan to:

- define formally the event, condition and action sub-languages of our RDF ECA rules;
- define the API requirements for the support of such rules over SeLeNe's RDF repository (likely to be FORTH's RDFSuite [5]);
- implement and experiment with the language, using as a testbed SeLeNe's educational metadata.

4 ECA Rules in a Distributed Environment

Beyond the centralized version of our system, we plan to develop a distributed version supporting ECA rules on distributed RDF repositories, as part of the ongoing SeLeNe project (`http://www.dcs.bbk.ac.uk/selene`). This project is investigating the technical requirements, and possible technical solutions, for 'self e-learning networks', where a self e-learning network is a distributed repository of metadata relating to learning objects (LOs) accessed by users wishing to publish or use such LOs. A self e-learning network (SeLeNe) will have a peer-to-peer topology, with facilities for peers to join or leave the network. Each peer will manage a fragment of the overall distributed metadata. This metadata will be expressed in RDF, and will contain information about learning objects and about the users of the SeLeNe (see [16]). Support of such networks will require:

- techniques for reconciliation and integration of heterogeneous metadata;
- definition of personalised views over this distributed metadata resource;
- detection, notification and propagation of changes to the metadata descriptions.

These requirements have a good fit with the functionality that could potentially be provided by ECA rules, and the architecture that we envisage is illustrated in Figure 4. Each 'peer' shown in that diagram is actually a 'super-peer' (SP) which may be coordinating a group of further peers (not shown in the figure).

At each SP there is installed one local ECA Engine, which has the same features and components as the centralized architecture discussed in Section 2.2 above and illustrated in Figure 1. One possibility is that each local ECA Engine will operate as a Web Service and that the communication between them can be via XML messages (e.g. SOAP).

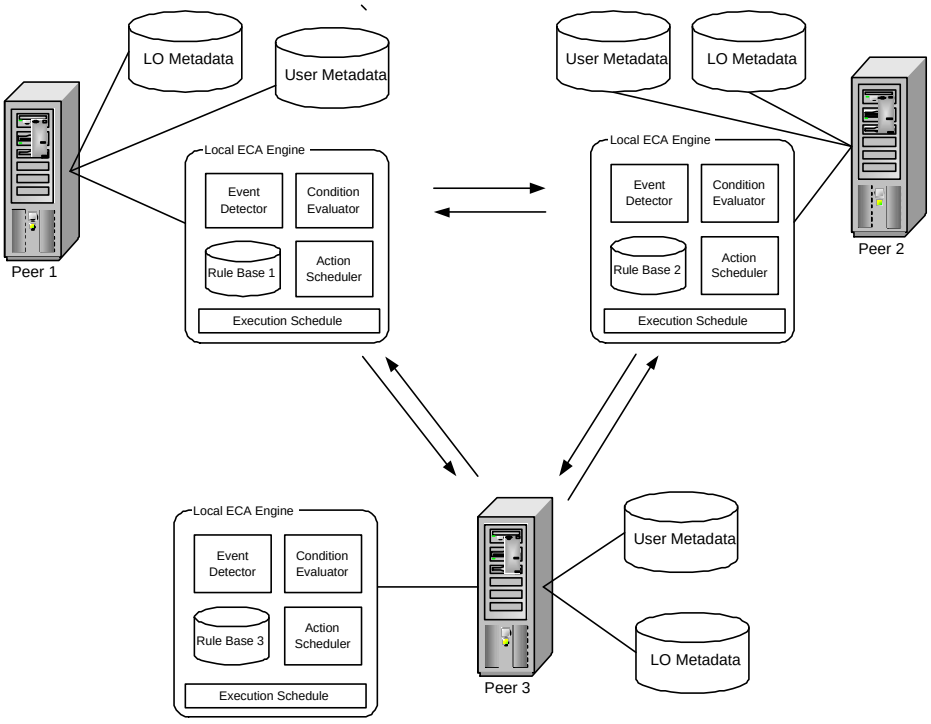

Fig. 4. Distributed System Architecture

Whenever a new ECA rule r is registered at a peer P, it will be sent to P's SP for storage. As we will see below, from there r will also be sent to all other SPs, and a replica of it will be stored at those SPs that are *relevant* to r i.e. where an event may occur that may trigger r's event part, or which may participate in evaluating r's condition part, or where r's actions may have to be scheduled for execution. At present, we assume that individual events and actions will occur at a single peer (which is likely to be the case in SeLeNe) although condition evaluation may be distributed.

Indexing at Peers and Super-Peers In order to determine whether an SP is relevant for a rule, an index can be kept at each peer and super-peer. There are a number of possibilities for doing this and we indicate here one solution:

As the RDF descriptions stored at each peer change over time, so each peer maintains an annotated copy of its local RDF Schema, which shows for each node in the schema whether or not there is RDF data of this type at this peer (a '0' or '1' bit).

This information is also propagated to the peer's coordinating SP. This SP maintains a combined RDF Schema which is annotated so that each node shows the set of peers in its own peer group that manage data of this type (a set of peer IDs), and also the remote SPs whose peer group manages such data (a set of SP IDs).

The latter information is gathered and maintained as follows: if a node in the RDF Schema of an SP changes from not having any data in this peer group to having data, or vice versa, this change is notified to all other SPs so that these can update the relevant annotation in their RDF Schemas. Note that in general the SPs may hold heterogeneous RDF Schemas, so there will need to be an RDF Schema translation service between SPs (as is indeed envisaged for SeLeNe).

Finally, as well as this annotated RDF Schema, each SP also keeps for each node annotated with a '1' in its RDF Schema a list of the RDF resources of this type that each peer in its peer group references — we call these lists of RDF resources *resource indexes*.

Comparison with related approaches: Querying and indexing data in a distributed RDF-based P2P network is more complex than for distributed structured databases. In the latter, the database servers and the database schema at each of them is known and fixed whereas in the former peers may dynamically join or leave the network and may manage data conforming to varying schema fragments. Schema-based routing indexes have been proposed to address this problem in Edutella [18]. Edutella uses two kinds of routing indexes: Super-Peer/Peer (SP/P RI) and Super-Peer/Super-Peer (SP/SP RI).

An SP/P RI stores information about metadata usage in each peer in its peer group. This includes information such as the schemas (e.g., `dc` or `lom`) or properties (e.g., `dc:subject`) used, as well as possibly conventional indexes on property values. When a peer registers with a SP, it provides the SP with its metadata usage, a process called advertisement. The peer undertakes to keep this advertisement up-to-date by informing its SP each time that a change affecting the advertised metadata takes place. At each super-peer, query fragments are matched against the SP/P RIs in order to determine peers that are relevant to this query (although this gives no guarantee that the returned result set from a peer is not empty). A similar approach is used in SP/SP RIs, but at a higher level of granularity and possibly only representing approximations of the information regarding their peers. A further difference to the SP/P RI is that an SP/SP RI contains information only about its neighbouring SPs in the SP topology. Update of SP/SP RIs is again based on broadcast messages sent between SPs.

For our purposes, we want to maintain more precise information about where various forms of metadata reside in the network and, as far as possible, do not want unnecessary routing of queries and updates to peers and super-peers that are not relevant. Hence, we have adopted the approach of using annotations on a full RDF schema and also resource indexes. The scalability of our proposal, however, still needs to be investigated.

Registering an ECA rule When a new rule is generated at a peer, it is sent to the peer's own SP for storage in its local rule base. The SP annotates the event, condition and action parts of the rule with the local peers that are relevant to each part (a list of peer IDs).

This can be determined by matching each part of the rule against the SP's annotated RDF Schema and/or its resource indexes — the former is useful if no resource is specified in this part of the rule and the latter is useful if a resource is specified. As the annotated RDF Schema and resource indexes at the SP evolve, so the annotations on the ECA rules can also be evolved to maintain consistency.

The rule is also sent to all other SPs that may be relevant to it — this is determined from the SP ID annotations on the originating SP's RDF Schema. These SPs repeat the above process of matching each part of the rule against their own annotated RDF Schema, and storing the resulting annotated rule in their own rule base it if it is indeed relevant to any of their peer group. Note that, due to schema heterogeneity, the rule may first have to be translated so that its parts are expressed with respect to the local RDF Schema.

The final result is a replica of the rule at each SP which is relevant to the rule, annotated with local information about which peers may be affected by each part of the rule.

As the information at SPs changes with time, it may be that an ECA rule is no longer relevant to that SP, in which case the rule can be deleted from the SP's local rule base. Conversely, an ECA rule stored somewhere else may become relevant to an SP. This can be handled as follows:

Since all SPs know what kinds of data is stored at all other SPs, if any SP, SP1, is notified of a change in status of another SP, SP2, from not having data associated with a particular RDF Schema node to having such data, then SP1 will send SP2 a copy of any ECA rules that originated from SP1 and that may now have become relevant to SP2.

Rule triggering and execution At run-time, whenever an event E occurs at a peer P, it will notify its SP. This will determine whether E may trigger any ECA rule annotated with P's ID. If a rule r might have been triggered, the SP will send P r's event query to evaluate.

If r has indeed been triggered, its condition will need to be evaluated, after generating an instantiation of it for each value of the `$delta` variable if this is present in the condition. The annotations on r can be used to determine to which local peers and other SPs sub-queries of the condition should be dispatched for evaluation. If the `$delta` variable is present in the condition, it will have

been instantiated and so we also consult the SPs' resource indexes for more precise information about which local peers are relevant to sub-queries of the instantiated condition.

If a condition evaluates to true, each corresponding rule action will be sent to, and scheduled, by the SPs that will execute it. Again this can be determined by the annotations on the rule action and consulting the SPs' resource indexes.

4.1 Future Work

There are several open issues remaining in realising the P2P ECA architecture we describe above:

- developing algorithms for matching rule event, condition and action parts with the schema-based indexes;
- defining the syntax of messages that will be passed between peers for distributed processing of ECA rules;
- defining the coordination with SeLeNe's distributed query processor for the evaluation of rule conditions;
- defining the coordination with SeLeNe's mediation functionality, for translating data and rules between heterogeneous schemas;
- more generally, mapping this distributed ECA functionality onto SeLeNe's service-based architecture;
- exploring distributed transactional aspects of the ECA rules (even though we assume that individual events and actions will occur at a single peer, the execution of an ECA rule may trigger another ECA rule and this whole cascade of rule firings may need to have the semantics of a single transaction).

5 Conclusions

In this paper we have discussed the provision of ECA rules for XML and RDF repositories, and have highlighted some of the new issues that arise in the context of such data. We have described a language for ECA rules on XML, and some preliminary ideas regarding a language for ECA rules on a graph/triple representation of RDF. We have described a prototype centralised implementation of the XML ECA rule language, and the architecture of a distributed implementation of the latter. For future work there are several directions to explore, as highlighted in Sections 2.3, 3.1 and 4.1 above.

An important issue is to evaluate the applicability and scalability of our languages, their execution models, and implementation. For this, we plan to deploy them for providing reactive functionality on distributed RDF repositories of educational metadata, as part of the ongoing SeLeNe project. This will also provide an opportunity to assess the impact of moving from a centralised to a distributed environment, with the additional challenges of network delay, network reliability, synchronisation of rule execution, maintaining consistency of the distributed metadata resource, tolerance of delays and failures etc.

References

1. S. Abiteboul, S. Cluet, G. Ferran, and M.-C. Rousset. The Xyleme project. *Computer Networks*, 39:225–238, 2002.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The Lorel query language for semistructured data. *VLDB Journal*, 1(1):68–88, 1997.
3. S. Abiteboul, V. Vianu, B. S. Fordham, and Y. Yesha. Relational transducers for electronic commerce. *JCSS*, 61(2):236–269, 2000.
4. A. Adi, D. Botzer, O. Etzion, and T. Yatzkar-Haham. Push technology personalization through event correlation. In *Proc 26th Int. Conf. on Very Large Databases*, pages 643–645, 2000.
5. S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proc. 2nd. Int. Workshop on the Semantic Web (SemWeb 2001)*, 2001.
6. J. Bailey, A. Poulovassilis, and P.T. Wood. An Event-Condition-Action Language for XML. In *Proc. WWW'2002*, Hawaii, 2002.
7. J. Bailey, A. Poulovassilis, and P.T. Wood. Analysis and optimisation for event-condition-action rules on XML. *Computer Networks*, 39:239–259, 2002.
8. A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active XQuery. In *Proc. of the IEEE Conference on Data Engineering (ICDE)*, 2002.
9. A. Bonifati, S. Ceri, and S. Paraboschi. Active rules for XML: A new paradigm for e-services. *VLDB Journal*, 10(1):39–47, 2001.
10. A. Bonifati, S. Ceri, and S. Paraboschi. Pushing reactive services to XML repositories using active rules. In *WWW'01*, 2001.
11. S. Ceri and P. Fraternali. *Designing Database Applications with Objects and Rules: The IDEA Methodology*. Addison-Wesley, 1997.
12. S. Ceri, P. Fraternali, and S. Paraboschi. Data-driven one-to-one web site generation for data-intensive applications. In *Proc. 25th Int. Conf. on Very Large Databases*, pages 615–626, 1999.
13. E. Cho, I. Park, S. J. Hyum, and M. Kim. ARML: an active rule mark-up language for heterogeneous active information systems. In *Proc. RuleML 2002*, Sardinia, June 2002.
14. S. Cluet, P. Veltri, and D. Vodislav. Views in a large scale XML repository. In *Proc. 27th Int. Conf. on Very Large Databases*, pages 271–280, 2001.
15. H. Ishikawa and M. Ohta. An active web-based distributed database system for e-commerce. In *Proc. Web Dynamics Workshop, London*, 2001. `http://www.dcs.bbk.ac.uk/webDyn/`.
16. K. Keenoy *et al.* Self e-Learning Networks — Functionality, User Requirements and Exploitation Scenarios. See `http://www.dcs.bbk.ac.uk/selene/reports/Del22.pdf`, August 2003. SeLeNe Project Deliverable 2.2.
17. S. Kokkelink. Transforming RDF with RDFPath. See `zoe.mathematik.uni-osnabrueck.de/QAT/Transform/RDFTransform.pdf`, March 2001.
18. W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, and A. Loser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *Proc. WWW2003*, pages 536–543, 2003.
19. W. Nejdl *et al.* EDUTELLA: A P2P Networking Infrastructure Based on RDF. In *Proc. WWW'2002*, 2002.
20. N. Paton, editor. *Active Rules in Database Systems*. Springer-Verlag, 1999.

21. J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In *Proc 7th Int. Conf. on Cooperative Information Systems (CoopIS'2000)*, pages 162–173, 2000.
22. B. Simon *et al.* Smart space for learning: A mediation infrastructure for learning services. In *Proc. WWW'2003*, 2003.
23. I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 413–424, 2001.
24. G. Wagner. How to Design a General Rule Markup Language? In *Invited talk at the Workshop XML Technologien für das Semantic Web (XSW 2002)*, Berlin, June 2002.
25. J. Widom and S. Ceri. *Active Database Systems*. Morgan-Kaufmann, San Mateo, California, 1995.
26. World Wide Web Consortium. XML Path Language (XPath), Version 1.0. See `http://www.w3.org/TR/xpath`, November 1999. W3C Recommendation.
27. World Wide Web Consortium. XSL Transformations (XSLT), Version 1.0. See `http://www.w3.org/TR/xslt`, November 1999. W3C Recommendation.
28. World Wide Web Consortium. XQuery 1.0: An XML Query Language. See `http://www.w3.org/TR/xquery`, November 2002. W3C Working Draft.
29. World Wide Web Consortium. Document Object Model (DOM) Level 3 Core Specification. See `http://www.w3.org/TR/DOM-Level-3-Core/`, February 2003. W3C Working Draft.