Event-Condition-Action Rule Languages for the Semantic Web

Alexandra Poulovassilis, George Papamarkos, Peter T. Wood

London Knowledge Lab, Birkbeck, University of London, London WC1E 7HX
email: {ap,gpapa05,ptw}@dcs.bbk.ac.uk

Abstract. The Semantic Web is based on XML and RDF as standards for exchanging and storing information on the World Wide Web. Event-Condition-Action rules are a possible candidate technology for distributed web-based applications that require timely notification and propagation of events and information between different sites. This paper discusses the provision of ECA rules for XML and RDF data, and highlights some of the challenges that arise.

1 Introduction

XML and RDF are becoming dominant standards for storing and exchanging information on the World Wide Web, and are being increasingly used in distributed web-based applications in areas such as e-business, e-science, e-learning and e-government. Such applications may need to be *reactive*, i.e. to be able to detect the occurrence of specific events or changes within information repositories, and to respond by automatically executing the appropriate application logic. Event-condition-action (ECA) rules are one way of providing this kind of functionality. An ECA rule is of the form `on` *event* `if` *condition* `do` *actions*. The event part specifies when the rule is *triggered*. The condition part is a query which determines if the information system is in a particular state, in which case the rule *fires*. The action part states the actions to be performed if the rule fires. These actions may in turn cause further events to occur, which may in turn cause more ECA rules to fire[1].

References [24, 20] discuss ECA rules (triggers) in databases. More broadly, ECA rules are used in workflow management, network management, personalisation, publish/subscribe technology, and specifying and implementing business processes. In the distributed web-based applications that we envisage, rules are likely not to be hand-crafted but automatically generated by higher-level presentation and application services.

For some applications, content-based publish/subscribe [9] may be an alternative or complementary technology to ECA rules. Publish/subscribe systems

[1] Non-termination of rule execution is generally a possibility and thus much research has focussed on the development of static and dynamic analysis techniques for detecting possibly non-terminating ECA rule sets. There has also been considerable research into techniques for verifying the confluence of ECA rules.

such as [7, 23] support more sophisticated distributed event definition and detection than ECA rules. On the other hand, ECA rules allow the definition and execution of more complex actions than just simple notifications.

This paper discusses the provision of ECA rules for XML and RDF data, and highlights some of the issues that arise in the context of such data. This work has been motivated by our participation in the EU FP5 "SeLeNe: Self e-Learning Networks" project (see `http://www.dcs.bbk.ac.uk/selene/`). The aim of this project was to investigate techniques for managing evolving distributed repositories of educational metadata and for providing a variety of services over such repositories, including syndication, notification and personalisation services. Peers in a SeLeNe (self e-learning network) store metadata relating to learning objects (LOs) registered with the SeLeNe, and also metadata relating to users of the SeLeNe. SeLeNe's reactive functionality includes features such as propagating changes in the description of a LO to those of composite LOs dependent on it; propagating changes in a learner's history of accesses to LOs to the learner's personal profile; notifying users of the registration of new LOs of interest to them; and notifying users of changes in the description of LOs of interest to them. We investigated the provision of this kind of reactive functionality by means of ECA rules over SeLeNe's metadata, considering first XML and then RDF encodings of the metadata.

2 ECA Rules for XML

In [4, 5] we introduced a language for defining ECA rules on XML data, based on XPath and XQuery. This language uses a fragment of XPath for selecting and matching sub-documents of XML documents within the event and condition parts of ECA rules, while a fragment of XQuery is used within insertion actions where there is a need to be able to construct new XML sub-documents. We also developed techniques for analysing the triggering and activation relationships between such rules[2] which can be 'plugged into' existing generic frameworks for ECA rule analysis and optimisation.

The semistructured nature of XML data gives rise to a number of issues in the context of ECA rules:

- *Event semantics*: For relational data, the semantics of data manipulation events is straightforward, since insert, delete or update events occur when a relation is inserted into, deleted from, or updated. With XML, specifying where data has been inserted or deleted within an XML document is more complex, and path expressions that identify locations within the document are necessary.

[2] 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.

– *Action semantics*: Again for relational data, the effect of data manipulation actions is straightforward, since an insert, delete or update action can only affect tuples in a single relation. With XML, actions now manipulate entire subdocuments, and the insertion or deletion of subdocuments can trigger a set of different events.
– *Rule analysis*: The determination of triggering and activation relationships between ECA rules is more complex for XML data than for relational data. The associations between actions and events/conditions are more implicit, and more sophisticated semantic comparisons between sets of path expressions are required.

Details of the syntax and rule execution semantics of our XML ECA rule language can be found in [5]. Reference [3] describes a prototype implementation the language: A *Parser* component parses and checks the syntactic validity of new ECA rules. Valid rules are stored in a *Rule Base*. An *Execution Engine* encapsulates the rule processing functionality, comprising an *Event Dispatcher*, a *Condition Evaluator* and an *Action Scheduler*. All of these components interface with a *Wrapper* which sends/receives data to/from the underlying XML files. The Action Scheduler places the updates resulting from rules that have fired at the head of an *Execution Schedule*. If multiple rules have fired, then the updates that result from their actions are prefixed to the schedule in order of the rules' specified priorities[3].

A number of other ECA rule languages for XML have also been proposed, although none of this other work has focussed on analysing rule behaviour. Most notably, Active XQuery [6] is an ECA rule language for XML based on the SQL3 triggers standard [13]. This language is more complex than ours as it allows full XPath in the event parts of rules, and full XQuery in the condition and action parts. However, analysing the behaviour of ECA rules expressed in this more complex language has not been considered. The rule execution model is also different to ours: 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 Active XQuery such updates are broken up into a sequence of finer granularity requests each of which may invoke the ECA rule execution. In general, these semantics may produce different results for the same initial update.

ARML [8] provides an XML-based rule description for rule sharing among different heterogeneous ECA rule processing systems. In contrast to our language and Active XQuery, conditions and actions are defined abstractly as XML-RPC methods which are later matched with system-specific methods. Active XML [1] provides similar functionality to that provided by XML ECA rules by embedding calls to web services within XML documents via special tags, aiming to integrate distributed data and distributed computation in P2P architectures.

[3] This prefixing to the schedule is `Immediate` rule scheduling, and other rule scheduling alternatives would also be possible e.g. `Deferred` and `Detached`, where updates are appended to the transaction or are executed as a separate transaction, respectively.

In the commercial arena, triggers on XML data are now supported by all the major relational DBMS vendors and also by some native XML repository vendors. However, this is confined to document-level triggering and only events concerning the insertion, deletion or update of an XML document can be caught. In relational DBMS it is however possible to decompose XML documents into a set of relational tables, potentially allowing developers to exploit existing relational triggering functionality in order to define finer-grain triggers over XML data.

3 ECA Rules for RDF

XML ECA rule languages can be used for RDF data which has been serialised as XML. However, we have also developed an RDF ECA rule language, RDFTL, that will operate directly on a graph/triple representation [17, 18]. To our knowledge, this is the first ECA rule language developed specifically for RDF.

Languages for updating RDF descriptions have been considered in [15, 14]. The Modification Exhange Language (MEL) of [15] is based on an RDF representation of Datalog and is used for updating RDF in the distributed environment of Edutella [16] while RUL (RDF Update Language) [14] is based on the RQL [12] query language.

RDFTL operates over RDF graphs and it is assumed that these RDF graphs conform to one or more RDFS schemas, in the sense that (a) every resource in the RDF graph belongs to an RDFS class (in addition to belonging to the default `rdfs:Resource` class); (b) every property in the RDF graph is declared in the RDFS schema, along with domain and range constraints; (c) the subject and object of every property in the RDF graph are of the declared subject and object type of the property in the RDFS schema.

RDFTL uses a path-based query sublanguage, syntactically similar to XPath, for defining queries over an RDF graph. Each RDFTL rule has an optional preamble consisting of one or more namespace definition clauses and a set of *let-expressions* of the form `let` *variable* := *e* associating a variable with a query.

The event part of an RDFTL rule describes updates whose occurrence will cause the rule to trigger, and is an expression of one of the following three forms:

1. `(INSERT | DELETE)` *e* `[AS INSTANCE OF` *class*`]`
2. `(INSERT | DELETE)` *triple*
3. `UPDATE` *upd_triple*

Form 1 detects insertions or deletions of resources specified by the expression *e*. *e* is a query, which evaluates to a set of nodes. Optionally, *class* is the name of the RDFS schema class to which at least one of the nodes identified by *e* must belong in order for the rule to trigger. The rule is triggered if the set of nodes returned by *e* includes any new node (in the case of an insertion) or any deleted node (in the case of a deletion) that is an instance of *class*, if specified. 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 that have triggered the rule.

Form 2 detects insertions or deletions of arcs specified by *triple*, which has the form *(source_node, arc_name, target_node)* where *source_node* and *target_node* may be expressions of the form *e* or variables defined in the rule's preamble. The wildcard '`_`' is allowed in the place of any of a triple's components. The rule is triggered if an arc labelled *arc_name* from *source_node* to *target_node* is inserted/deleted. The variable `$delta` has as its set of instantiations the triples which have triggered the rule; the individual components of these triples are identified by `$delta.source`, `$delta.arc_name` or `$delta.target`.

Form 3 similarly detects updates to the target nodes of arcs, specified by *upd_triple* which has the form *(source, arc_name, old_target → new_target)*. The wildcard '`_`' is allowed in the place of any of these components. The rule is triggered if an arc labelled *arc* from *source* changes its target from *old_target* to *new_target*. The variable `$delta` has as its set of instantiations the triples which have triggered the rule and the components of these triples can be obtained by `$delta.source`, `$delta.arc_name`, `$delta.old_target` or `$delta.new_target`.

The condition part of rule is a boolean-valued expression which may consist of conjunctions, disjunctions and negations of queries.

The actions part of a rule is a sequence of one or more actions. Actions can `INSERT` or `DELETE` a resource — specified by its URI — and `INSERT`, `DELETE` or `UPDATE` an arc. The actions language has the following form for each one of these cases, where triples in the actions part have a similar form as in the event part:

1. `INSERT` *e* `AS INSTANCE OF` *class*
   `DELETE` *e* `[AS INSTANCE OF` *class*`]`
   for expressing insertion or deletion of a resource, where the `AS INSTANCE OF` keyword classifies the resource to be deleted or inserted.
2. `(INSERT | DELETE)` *triple* `(',' triple)*`
   for expressing insertion or deletion of the arcs(s) specified.
3. `UPDATE` *upd_triple* `(',' upd_triple)*`
   for updating arc(s) by changing their target node.

The condition and action parts of a rule may contain occurrences of the `$delta` variable in place of a named resource in a query, or a component of a triple. If neither the condition nor the action part contain occurrences of `$delta`, then the rule is a *set-oriented rule*, otherwise it is an *instance-oriented rule*. A set-oriented rule *fires* if it is triggered and its condition evaluates to true. A copy of the rule's action part is executed as a new transaction (i.e. Detached rule coupling). An instance-oriented rule fires if it is triggered and its condition evaluates to true for some instantiation of `$delta`. A copy of the rule's action part is executed as a new transaction for each value of `$delta` for which the rule's condition evaluates to true, in each case substituting all occurrences of `$delta` within the action part by one specific instantiation for `$delta`.

We refer the reader to [18] for full details of the syntax and execution semantics of RDFTL, and that paper also discusses conservative tests for determining the termination and confluence of sets of RDFTL rules.

3.1 RDFTL Rules in P2P Environments

We have developed a system for processing RDFTL rules in P2P environments. The rule processing functionality is provided by a set of services that constitute the *RDFTL ECA Engine*. This acts as a wrapper over a distributed set of RDF/S repositories, exploiting their query, storage and update functionality. In the current version of our system we are using ICS-FORTH RSSDB [2] as the RDF repository. For the future we plan also to support Jena2 [11].

Our system architecture is similar to the superpeer-based architecture of Edutella [16]. Each peer in the network is supervised by a superpeer (each superpeer supervises itself). The set of peers supervised by a superpeer is termed its *peergroup*. At each superpeer there is an ECA Engine installed. Each peer or superpeer hosts a fragment of an overall global RDFS schema. As in Edutella, the metadata distribution in RDFTL allows hybrid fragmentation, with possible replication between peers. The fragment of the global RDFS schema stored at a peer may change as a result of changes in the peer's RDF/S descriptions. Peers notify their supervising superpeer of any updates to their local RDF/S repository. Peers may dynamically join or leave the network at any time.

Each superpeer's RDFS schema is a superset of its peergroup's individual RDFS schemas. Each superpeer defines access privileges over the classes and properties in its RDFS schema describing the corresponding access level to the instances of each class and property. More fine-grained access privileges are also allowed on specific RDF resources and triples. These facilities allow a superpeer to specify which information can be shared with other superpeers outside its peergroup.

An ECA rule generated at one site of the network might be replicated, triggered, evaluated, and executed at different sites. Within the event, condition and action parts of ECA rules there might be references to specific RDF resources.

Whenever a new ECA rule r is generated at a peer P, it is sent to P's superpeer for syntax validation, translation into the local repository's query and update language, and storage. From there, r will also be forwarded to all other superpeers, and a replica of it will be stored at those superpeers where an event may occur that may trigger r's event part, i.e. those superpeers that are *e-relevant* to r (see below). A rule r has a globally unique identifier of the form $SP_i.j$, where SP_i is the originating superpeer identifier and j a locally unique identifier for the rule in SP_i's rule base.

At run-time rules are triggered by events occurring within a single peer's local RDF repository, i.e. there is no distributed event detection. Also, each particular copy of a rule's action part executes within a single peer's RDF repository, i.e. there is no distributed update execution. If there is a need to distribute a sequence of updates across a number of peers in reaction to some event, then rather than specifying one rule of the form `on` e `if` c `do` $a_1; \ldots; a_n$ instead, n rules $r_1, \ldots, r_n$ can be specified, where each r_i is `on` e `if` c `do` a_i and r_1 has a higher precedence than r_2, which has a higher precedence than r_3 etc.

There are three types of *relevance* of a rule r to an RDF schema S:

- r is *e-relevant* to S if each of the queries that either appear in the event part of r or are used by the event part through variable references, can be evaluated on S, i.e., each step in each path expression exists in S.
- r is *c-relevant* to S if some step in one of the queries referenced by the condition part of r can be evaluated on S (unlike events and actions, conditions may be evaluated at multiple sites).
- r is *a-relevant* to S if all actions in the action part of r are *a-relevant* to S. An individual action is a-relevant to S if it satisfies one of the following:
  - If it is a deletion or insertion of resources that uses `AS INSTANCE OF` *`class`*, then *`class`* must be in S.
  - If it is a deletion of resources that does not use `AS INSTANCE OF` *`class`*, then the most specific class of resources that the path expression in the deletion would return must be in S.
  - If it is an action over triples that uses a property p, then p must be in S. If it is a deletion of triples that uses the wildcard '_' instead of a property (the only action allowed to do this), then the classes of resources returned by the path expressions involved in the deletion must exist in S.

A peer or superpeer is e-relevant, c-relevant or a-relevant to a rule r if r is e-, c- or a-relevant, respectively, to the peer or superpeer's RDFS schema.

At each superpeer, each rule is annotated with the IDs of local peers that are e-relevant, c-relevant and a-relevant to it. These annotations are kept synchronised with changes in peers' and superpeers' schemas.

3.2 P2P Rule Execution

The RDF graph is fragmented, and possibly replicated, amongst the peers, and each superpeer manages its own local rule execution schedule. Each execution schedule at a superpeer is a sequence of updates which are to be executed on the fragment of the global RDF graph which is stored at the superpeer or its local peergroup. Each superpeer coordinates the execution of transactions that are initiated by that superpeer, or by any peer in its local peergroup.

Whenever an update u is executed at a peer P, P notifies its supervising superpeer SP. SP determines whether u may trigger any ECA rule whose event part is annotated with P's ID. If a rule r may have been triggered, then SP will send r's event query to P 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. If a condition evaluates to true, SP will send each instance of r's action part (one instance if r is a set-oriented rule, and one or more instances if r is an instance-oriented rule) to the local peers that are a-relevant to it. All instances of r's actions part will also be sent to all other superpeers of the network. All superpeers that are a-relevant to r will consult their schemas and access privileges in order to determine whether the updates they have received can be scheduled and executed on their local peergroup.

In summary therefore, local execution of the update at the head of a local schedule may cause events to occur. These events may cause rules to fire, modifying the local schedule or remote schedules with new updates to be executed. We refer the reader to [18] for full details of the P2P implementation of RDFTL.

Our current RDFTL implementation does not yet support any concurrency control or recovery mechanisms. In principle, any distributed concurrency control protocol could be adapted to a P2P environment. For example, the AMOR system adopts optimistic concurrency control [10]. The serialisation graph is distributed amongst those peers responsible for transaction coordination (analogous to our superpeers). The AMOR system assumes that conflicts are only possible between those transactions that are accessing a particular 'region' of resources (analogous to our peers) and thus subgraphs of the global serialisation graph are stored and replicated amongst those coordinators which service a particular region. The regions are not static and these subgraphs are dynamically merged and replicated as transactions execute and regions evolve.

In the classical approach to distributed transactions, global transactions hold on to the resources necessary to achieve their ACID properties until such time as the whole transaction commits or aborts. In a P2P environment this may not be feasible: the resources available at peers may be limited, peers may not wish to cooperate in the execution of global transactions, and peers may disconnect at any time from the network, including during the execution of a global transaction in which they are participating. The cascaded triggering and execution of ECA rules will cause longer-running transactions which may further exacerbate these problems. It is therefore necessary to relax the Atomicity and Isolation properties of transactions.

In particular, subtransactions executing at different peers may be allowed to commit or abort independently of their parent transaction committing or aborting, and parent transactions may be able to commit even if some of their subtransactions have failed. Subtransactions that have committed ahead of their parent transaction committing can be reversed, if necessary, by executing compensating subtransactions. These can be generated as transactions execute and they reverse the effects of a transaction by compensating each of the transaction's updates in reverse order of their execution. Generating compensating updates is straight-forward for RDFTL updates: the insertion of a triple is reversed by deletion of the triple, the deletion of a triple by an insertion, and an update by the restoration of the original value. If transactions have read from committed (sub)transactions which are subsequently reversed, then a cascade of compensations will result.

3.3 Performance

We have developed an analytical performance model for our P2P RDFTL rule processing system, which is described in [19]. We use as the main performance criterion the update response time i.e. the mean time required to complete all rule processing resulting from a top-level update submitted to one of the peers in the network. In [19] we examine how the update response time varies with

the network topology, number of peers, number of rules, and degree of data replication between peers. We also describe a simulation of the system, and present the results of similar performance and scalability experiments with the simulator.

The two sets of experimental results show good agreement. Both sets of experiments show that the system performance is significantly reliant on the network topology between superpeers. In particular, if a Hypercup topology [22] is used for interconnecting the superpeers, then rule processing shows good scalability, pointing to its practical usefulness as a technology for real applications. For the future we would like to conduct large-scale experiments with the actual RDFTL system itself, possibly using the PlanetLab [21] infrastructure. As well as giving insight into the actual system behaviour in a real P2P environment, this will allow measurements on actual system workloads and rule sets, which can then be fed into the analytical performance model and the simulator to allow more accurate predictions from these.

4 Concluding Remarks

In this paper we have discussed the provision of ECA rules for XML and RDF data, and have highlighted some of the new issues that arise in the context of such data. We have described a language for ECA rules over XML data, a language for ECA rules over RDF data, implementations of these languages, and the results of a study into the performance and scalability of our RDF ECA rule processing system in P2P environments.

Although conducted in the context of ECA rules operating on RDF, we expect that similar behaviour would occur for P2P ECA rules operating on other types of data e.g. XML and relational, and this is an area of planned future work. Also planned for the future is a distributed version of our XML ECA rule system, and a deeper study into expressiveness of our languages, in terms of their update and constraint enforcement capabilities.

References

1. S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: peer-to-peer data and web services integration. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 1087–1090, 2002.
2. 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*, 2001.
3. J. Bailey, G. Papamarkos, A. Poulovassilis, and P. T. Wood. An Event-Condition-Action Rule Language for XML. In M. Levene and A. Poulovassilis, editors, *Web Dynamics*. Springer, 2004.
4. J. Bailey, A. Poulovassilis, and P.T. Wood. An Event-Condition-Action Language for XML. In *Proc. 11th Int. Conf. on the World Wide Web*, pages 486–495, 2002.
5. J. Bailey, A. Poulovassilis, and P.T. Wood. Analysis and optimisation for event-condition-action rules on XML. *Computer Networks*, 39:239–259, 2002.

6. A. Bonifati, D. Braga, A. Campi, and S. Ceri. Active XQuery. In *Proc. 18th Int. Conf. on Data Engineering*, pages 403–418, 2002.
7. P. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Publish/subscribe for RDF-based P2P networks. In *Proc. ESWS 2004, Heraklion, Crete*, pages 182–197, 2004.
8. E. Cho, I. Park, S. J. Hyun, and M. Kim. ARML: an active rule mark-up language for heterogeneous active information systems. In *Proc. RuleML 2002*, Sardinia, June 2002.
9. P. T. Eugster, P. A. Felber, R. Guerraoui, and A. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.
10. K. Haller, H. Schuldt, and H.J. Schek. Transctional peer-to-peer information processing: The AMOR approach. In *4th Int. Conf. on Mobile Data Management*, pages 356–362. Springer, 2003.
11. Jena: A Semantic Web Framework for Java. http://jena.sourceforge.net/.
12. G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. RQL: A Functional Query Language for RDF. In *Functional Approaches to Computing Data*, pages 592–603. Springer, 2003.
13. K. Kulkarni, N. Mattos, and R. Cochrane. Active database features in SQL3. In N. Paton, editor, *Active Rules in Database Systems*, pages 197–219. Springer, 1999.
14. M. Magiridou, S. Sahtouris, V. Christophides, and M. Koubarakis. RUL: A declarative update language for RDF. In *Proc. Fourth Int. Semantic Web Conference*, pages 506–521, 2005.
15. W. Nejdl, W. Siberski, B. Simon, and J. Tane. Towards a modification exchange language for distributed RDF repositories. In *Proc. First Int. Semantic Web Conference*, pages 236–249, 2002.
16. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *Proc. 11th Int. Conf. on the World Wide Web*, pages 604–615, 2002.
17. G. Papamarkos, A. Poulovassilis, and P. T. Wood. RDFTL: An Event-Condition-Action Language for RDF. In *Proc. 3rd Int. Workshop on Web Dynamics (in conjunction with WWW2004)*, 2004.
18. G. Papamarkos, A. Poulovassilis, and P. T. Wood. Event-Condition-Action Rules on RDF metadata in P2P environments, Technical Report BBKCS-05-05, to appear in Computer Networks, 2006.
19. G. Papamarkos, A. Poulovassilis, and P. T. Wood. Performance Modelling and Evaluation of Event-Condition-Action Rules RDF in P2P networks, Technical Report BBKCS-06-02, 2006.
20. N. Paton. *Active Rules in Database Systems*. Springer, 1999.
21. PlanetLab. `http://www/.planet-lab.org`.
22. M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP – hypercubes, ontologies and efficient search on p2p networks. In *First Int. Workshop on Agents and P2P Computing*, pages 112–124, 2002.
23. W. W. Terpstra, S. Behnel, L. Fiege, A. Zeidler, and A. P. Buchmann. A peer-to-peer approach to content-based publish/subscribe. In *DEBS '03: Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–8, New York, NY, USA, 2003. ACM Press.
24. J. Widom and S. Ceri. *Active Database Systems*. Morgan-Kaufmann, San Mateo, California, 1995.