Efficient and Scalable Filtering of Graph-based Metadata

Haifeng Liu

Department of Computer Science
University of Toronto

Milenko Petrovic

Department of Computer Engineering
University of Toronto

Hans-Arno Jacobsen

Department of Computer Engineering
Department of Computer Science
University of Toronto

Abstract

RDF Site Summaries constitute an application of RDF on the Web that has considerably grown in popularity. However, the way RSS systems operate today limits their scalability. Current RSS feed arregators follow a pull-based architecture model, which is not going to scale with the increasing number of RSS feeds becoming available on the Web. In this paper we introduce G-ToPSS, a scalable publish/subscribe system for selective information dissemination. G-ToPSS only sends newly updated information to the interested user and follows a push-based architecture model. G-ToPSS is particularly well suited for applications that deal with large-volume content distribution from diverse sources. G-ToPSS allows use of an ontology as a way to provide additional information about the data disseminated. We have implemented and experimentally evaluated G-ToPSS and we provide results demonstrating its scalability compared to alternative approaches. In addition, we describe an application of G-ToPSS and RSS to a Web-based content management system that provides an expressive, efficient, and convenient update notification dissemination system.

Key words: publish/subscribe, content-based routing, RDF, information dissemination, graph matching

Accepted for publication in Journal of Web Semantics,2006 29 September 2005

1 Introduction

The amount of information on the Internet is continuously increasing. It is becoming increasingly easier for non-computer oriented users to publish information on the Internet because of myriads of user-friendly tools that now exist. For example, it is very easy for a user to keep an "online" diary (e.g., blogs) using a variety of tools. Collaboration tools such as a wiki, allow users to quickly publish information from within a web browser, without requiring access or knowledge of any additional applications. Finally, applications for web page authoring are becoming ever so easier to use. As a result of the advances in web page authoring tools, the number of information publishers has grown considerably.

RDF Site Summary (RSS) is a metadata language developed by the W3C for describing content changes. [1] RSS is so versatile that any kind of content changes can be described (e.g., web site modifications, wiki updates, and source code versioning histories). A RSS feed is a stream of RSS metadata that tracks changes for a particular content over time.

Typically, users apply a tool, which can read RSS feeds, to periodically check a number of RSS feeds by pulling RSS files from a web site. When RSS feeds indicate that the content has been updated, the user is informed. The user is expected to explicitly specify which RSS feeds to monitor.

A RSS feed aggregator is a service that monitors large numbers of feeds. It allows users to subscribe to the content that they are interested in without explicitly specifying which RSS feeds the content is coming from. This is particularly convenient for the user, since the number of RSS feeds that can carry information of interest to the user can be very large. In addition, a user does not have the resources to monitor large number of feeds and hence the user can easily miss information of interest.

RSS feed aggregators use pull-based architectures, where the aggregator pulls RSS feeds from a web site that hosts the feed. As the number of feeds on the web proliferates (e.g., due to ease of publishing information on the web), this architecture is not going to scale. It not only consumes unnecessary resources, but also becomes difficult to ensure timely delivery of updates.

Figure 1 illustrates the scalability problem. Multiple RSS aggregators (i.e., personal (desktop) aggregators, online news aggregators, and server side aggrega-

Email addresses: hfliu@cs.toronto.edu (Haifeng Liu), petrovi@eecg.toronto.edu (Milenko Petrovic), jacobsen@eecg.toronto.edu (Hans-Arno Jacobsen).

[1] http://web.resource.org/rss/1.0/spec

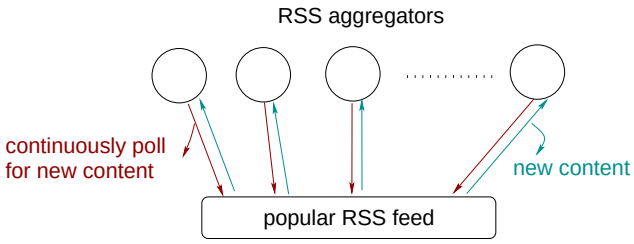


Fig. 1. Current RSS dissemination architecture

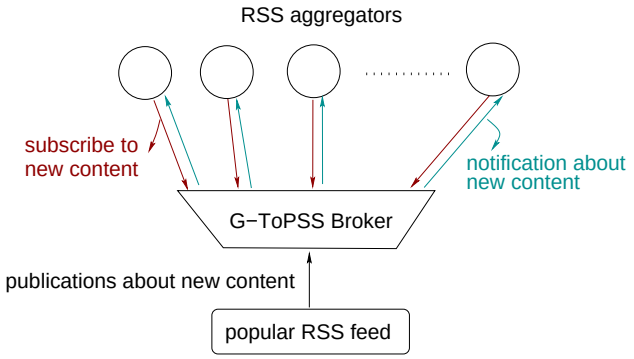


Fig. 2. G-ToPSS RSS dissemination architecture

tors) *poll* numerous RSS feed sites, *each*. Anecdotal evidence suggests that the way RSS dissemination is currently done can severely affect the performance of web sites hosting popular RSS feeds. [2]

In this paper, we describe G-ToPSS [3], a graph-based publish/subscribe architecture for dissemination of RDF data. This paper extends our previous work [22] with a presentation of a detailed application case study initially described in [21]. The G-ToPSS system provides fast filtering of RDF metadata such as RSS publications, as well as timely delivery of publications to interested subscribers in a scalable manner. Figure 3 shows the architecture of G-ToPSS. The new information system architecture significantly reduces the number of unnecessary polls of RSS feed sites. New content are only sent back to the interested aggregator, not all (see Figure 2).

RSS is just one application that can benefit from this architecture. Another application that is increasingly becoming important is content management in the enterprise. PDF is the de facto standard for representing documents in electronic form while preserving their original formatting. RDF metadata can

[2] InfoWorld RSS growing pains, July 16, 2004, RSS Traffic Burdens Publisher's Servers, July 19, 2004

[3] G-ToPSS is a part of the Toronto Publish/Subscribe System (ToPSS) research effort, which comprises a large number of publish/subscribe research projects, such as M-ToPSS (mobility-aware) [3,26,23,19], S-ToPSS (semantic matching) [20], A-ToPSS (approximate matching) [18], L-ToPSS (location-based matching) [26], PADRES (federated p/s) [13,16,17] and others.

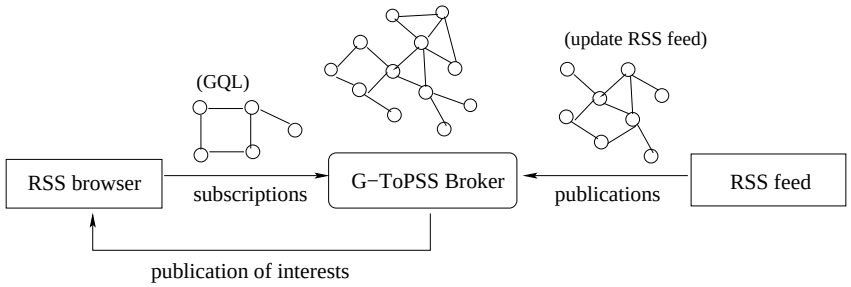


Fig. 3. RDF Site Summary Dissemination System based on G-ToPSS

be embedded in PDF documents, which aids in document management. G-ToPSS provides an architecture that could be applied to content-based routing to disseminate relevant documents throughout a wide area enterprise network.

In addition, [11] describes a number of uses cases for RDF data access, many of which can directly benefit from the described architecture. Some examples include "finding unknown media objects", "avoiding traffic jams" and "exploring the neighborhood."

G-ToPSS employs the publish/subscribe, data-centric communication model. There are three main entities in this model: publishers, subscribers and brokers. Publishers send all data to a broker (or a network of brokers). Subscribers register their interest with the broker in receiving relevant data. The role of a broker is to mediate communication between the publishers and the subscribers by matching the published data with the interests of the subscribers. This way the subscribers do not need to know who is publishing the data, as long as the data meets their specific interest, and the publishers do not need to know who are the ultimate receivers of their publications. This provides decoupling of senders and receivers of data both in space and time, which makes the publish/subscribe paradigm particularly well suited for structuring of large and dynamic distributed systems such as RSS feed dissemination, for example.

The contributions of this paper are three-fold. First, we present an original publish/subscribe system model, referred to as G-ToPSS, for large-volume graph-based content filtering. The G-ToPSS system supports the use of an ontology to specify taxonomy information about the data disseminated. Second, we develop a novel algorithm for filtering of graph-structured data and experimentally demonstrate the scalability of the approach. Finally, we present an application of G-ToPSS for the dissemination of content changes to users of a Web-based content management system.

The paper is organized as following. In Section 2 we briefly summarize related work. The G-ToPSS publish/subscribe model supporting graph matching is developed in Section 3. Section 4 describes the graph matching algorithms and data structures. Section 5 presents the experimental evaluation. In Section 6

we describe an application of G-ToPSS to a content management system. Section 7 concludes the paper and discusses possible directions for future work.

2 Related Work

The use of the publish/subscribe communication model for selective information dissemination has been studied extensively. Existing publish/subscribe systems [12,1,9,5] use attribute-value pairs to represent publications, while conjunctions of predicates with standard relational operators are used to represent subscriptions. Systems such as those described in [2,10] process XML publications and XPath subscriptions. XPath expressions represent path patterns over a document tree. XTrie [6] propose an index structure that supports filtering of XML documents based on many XPath expressions. The approach is extensible supporting patters including constraint predicates. Gupta *et al.* [14] show how to process XML stream over XPath queries including predicates. These approaches do not support the filtering of graph-structured data, which is the main motivation of our work.

Previously, we have built a prototype publish/subscribe system S-ToPSS [20] that extends the traditional attribute-value-pair-based systems with capabilities to process syntactically different, but semantically-equivalent information, thus achieving another level of decoupling, which we termed *representational decoupling*. S-ToPSS uses an ontology to be able to deal with syntactically disparate subscriptions and publications. The ontology which can include synonyms, a taxonomy and transformation rules was specified using S-ToPSS specific methods. On the other hand, G-ToPSS publication and subscription data models are based on directed graphs in general and RDF in particular. Use of RDF makes it possible for G-ToPSS to use ontologies built on top of RDF using languages such as RDFS and OWL. To illustrate this, in this paper, we extend the G-ToPSS subscription language with type constraints for subjects and objects, where the type information is represented in a RDFS taxonomy.

OPS [25] is another ontology-based publish/subscribe system whose publication and subscription model is also based on RDF. OPS uses a very general subgraph isomorphism algorithm for matching over overlapping graphs. However, this approach, as we show in this paper, unnecessarily increases the matching complexity because it assumes that any node of the publication graph can map to any node of the subscription graph. In this paper, we compare the performance of G-ToPSS to OPS and show that G-ToPSS always outperforms OPS.

A RDF document can be represented as directed labelled graph. Every node

in the graph has a unique name, and no two edges between any two nodes can have the same label. Given this assumption, in this paper, we show how to store such graphs in a way that exploits commonalities between them and how to use this data structure to efficiently filter publications.

Racer [15] is a publish/subscribe system based on a description logics inference engine. Since OWL is based on description logics, Racer can be used for RDF/OWL filtering. Racer does not scale as well as G-ToPSS. Its matching times are in the order of 10s of seconds even for very simple subscriptions [15], however, it offers more powerful inference capabilities not available in G-ToPSS. Chirita *at el.* and Cai *et al.* [7,4] design a publish/subscribe system supporting metadata and propose a query language based on RDF. Both approaches are based on peer-to-peer network abstractions and express queries in a triple pattern, rather than a graph-based language as central to G-ToPSS. No support for including ontology information in the filtering process is provided in either approach. Furthermore, G-ToPSS demonstrates greater scalability with a demonstrated throughput of millions of queries per second as compared to the throughput of 250 queries per second reported by RDFPeers [4].

CREAM [8] is an event-based middleware platform for distributed heterogeneous event-based applications. Its event dissemination service is based on the publish/subscribe model. Similar to other publish/subscribe systems, the subscription and publication model in CREAM, is based on attribute-value pairs. Like S-ToPSS, attributes and values can be associated with semantic information from an ontology. Unlike G-ToPSS, which is based on RDF, ontology and data are represented in a CREAM-specific data model. In addition, we are not aware of any quantitative evaluations of CREAM's scalability such as the one for G-ToPSS presented in this paper.

3 G-ToPSS Model

In this section, we describe the four components of the G-ToPSS data model: publications, subscriptions, matching semantics and ontology support. Publications are RDF documents. Subscriptions are queries for filtering of RDF documents following certain patterns. Our subscription language model is similar to RQL (RDF Query Language), but the difference is that RQL is a *typed* language featuring variables on labels for nodes (classes) and edges (properties). However, our G-ToPSS model only supports variables on node labels and opts to include ontology information in a separate taxonomy. We refer to our subscription language as *GQL*.

3.1 Publication Data Model

A G-ToPSS publication is a RDF document, which is represented as a *directed labelled graph*. By the specification of RDF semantics by Pat Hayes, an RDF graph is a set of triples *(subject, property, object)*. Each triple is represented by a node-edge-node link (as shown in Figure 4). *subject* and *property* are URI references, while *object* is either an URI reference or a literal. A publication is a directed graph where the vertices represent *subject*s and *object*s and edges between them represent *properties*.

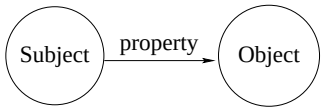


Fig. 4. RDF triple graph

Figure 5(c) illustrates a publication about one of Prof. Jacobsen's papers published in the 2001 SIGMOD conference.

3.2 Subscription Language Model

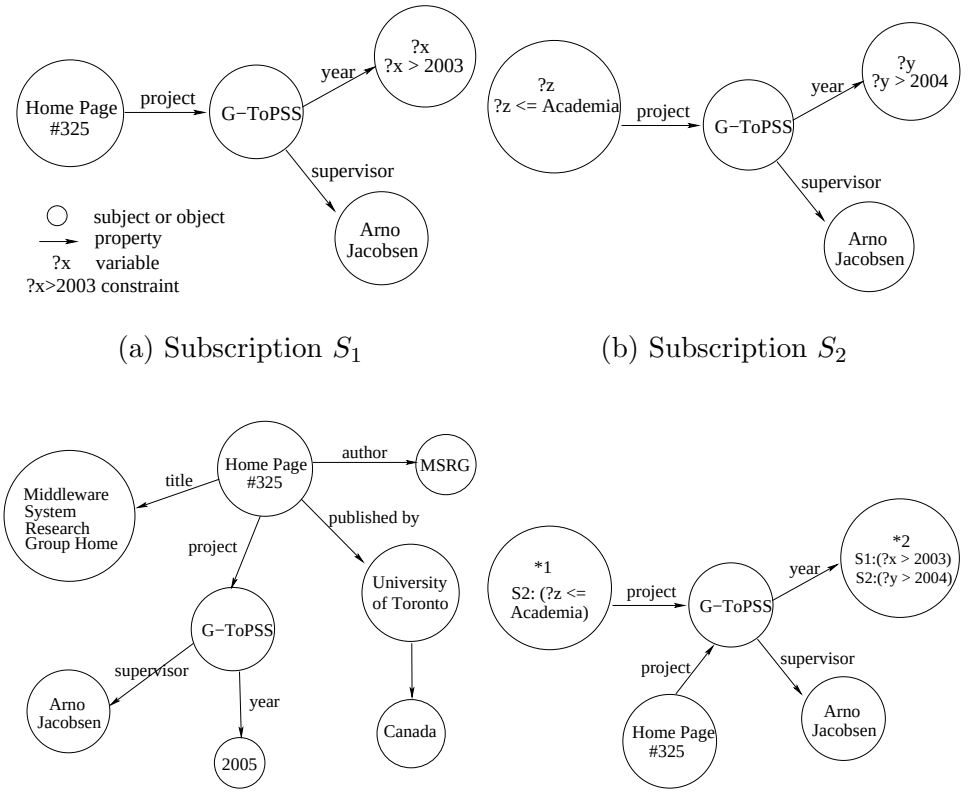
A G-ToPSS subscription is a *directed graph pattern* specifying the structure of the publication graph with optional constraints on vertices. A subscription is represented by a set of 5-tuples (*subject, property, object, constraintSet (subject), constraintSet (object)*). Constraint sets can be empty.

Similar to the publication data model, each 5-tuple can be represented as a link starting from the *subject* node and ending at the *object* node with the *property* as its label. From the publication data model, we know that each node is labelled with a specific value. However, in a subscription, we also allow *subject* and *object* to be either a constrained or unconstrained variable. An unconstrained variable matches any specific value of the publication; while the constraint variable matches only values satisfying the constraint. A constraint is represented as a predicate of the form $(?x, op, v)$ where $?x$ is the variable, op is an operator and v is a value.

There are two types of operators: Boolean, for literal value filtering and *is-a*, for RDFS taxonomy filtering. Boolean constrains are one of $=$, $\leq$ and $\geq$ with traditional relational operator semantics. *is-a* operators are also one of $=$, $\leq$ and $\geq$ but with alternative semantics. $\leq$ is "descendantOf" which means that variable $?x$ is an instance of a descendant of class v. $\geq$ is "ancesterOf" which means that $?x$ is an instance of an ancestor of class v. $=$ means that $?x$ is the direct instance of class v (i.e., a child of v).

For example, Figure 5(a) illustrates a subscription that specifies interest in a web page which is about the G-ToPSS project supervised by Arno and published after the year 2003. This type of constraint is for literal value filtering.

The subscription in Figure 5(b) is looking for a web page about a new project after 2004. There are two variables; the one constraining the year is a literal value filter; the other is a semantic constraint which uses the class taxonomy. Only an instance about HomePage which is a descendant of the "Academia" class is going to match (refer to Figure 6).



(a) Subscription S_1 (b) Subscription S_2

(c) Publication (d) G_M contains S_1 and S_2

Fig. 5. Example subscriptions, publication and G_M

3.3 Matching Semantics

We denote G_P as the publication graph and G_S as the subscription graph pattern. The matching problem is then defined as verifying whether G_S is embedded in G_P (or isomorphic to one or more subgraphs of G_P). Graph pattern G_S is *embedded* in G_P if every node in G_S maps to a node in G_P such that all constraints of G_S are satisfied.

Formally speaking, for each 5-tuple (*subject, property, object, constraintSet (subject), constraintSet (object)*) in subscription graph G_S, there is at least one triple (*subject, property, object*) in publication G_P such that the *subject* and *object* nodes are matched and linked by the same *property* edge. The nodes that match are either the same (i.e., their labels are lexicographically equal) or the node in G_S is a variable for which the value of the node in G_P satisfies all constraints associated with the variable.

For example, the subscription in Figure 5(a) is matched by the publication in Figure 5(c) since the publication contains the same links (*Home Page* ♯325*, project, G-ToPSS*), (*G-ToPSS, supervisor, Arno Jacobsen*), and (*2005 > 2003*), thus (*G-ToPSS, year,?x(?x > 2003)*) is satisfied.

3.4 Ontology Support

An RDFS class taxonomy with *is-a* relationship is the semantic information about a *subject* or an *object* that is available in the G-ToPSS ontology. An RDF schema supports constrain *is-a* relationship on *properties* (i.e., represented by the edge between *subject* and *object*). However, to simplify the system design, we only support the taxonomy information about *subject* and *object* nodes in our G-ToPSS model. As explained in the following section, structure matching and constraint matching are separate stages in the matching algorithm. It is straight forward to extend the current model to support other RDF schema semantics (e.g., *subPropertyOf*, *Datatype*, etc.).

G-ToPSS allows the designer to use multiple inheritance in the taxonomy, with the restriction that the taxonomy must be acyclic. The taxonomy lists all instances of a class. Alternatively, this information can be specified in the RDF graph using a *type* property, but for simplicity we have opted to include this information in the taxonomy. Note that an instance can also have multiple parents.

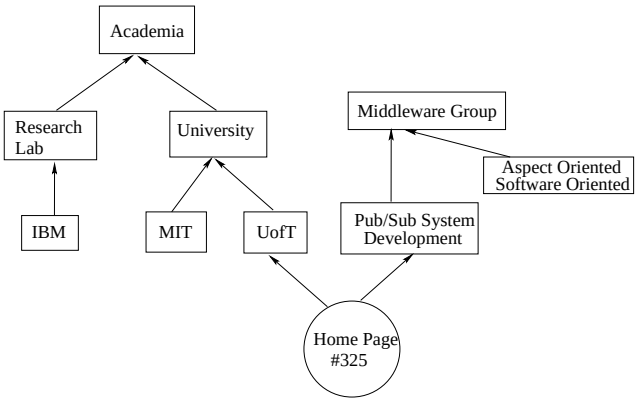


Fig. 6. Example taxonomy

In Figure 6, we show an example of a class taxonomy about an academic webpages system. Class "Academia" includes two subclasses: "Research Lab" and "University". Class "Middleware Group" includes "Pub/Sub System Development" and "Aspect Oriented Software Development" two subclasses. The document instance "Home page ♯325" belongs to both "UofT" and "Pub/Sub System Development".

As a side note, existing publish/subscribe systems are classified as either content-based or hierarchical (topic) based. Thus, a class taxonomy is a way to seamlessly integrate both models. When filtering, a subscription is matched if and only if both the content and the hierarchical constraints are satisfied.

4 Algorithm and Data Structure

To exploit overlap between subscriptions we integrate all subscriptions into a single graph. We denote the graph containing all subscriptions as G_M. Given all subscriptions, G_M, a publication, G_P, the publish/subscribe graph matching problem is to identify all the subgraphs G_{S_i} (representing a subscription S_i) in G_M which are matched by G_P. In other words, the goal is to determine all graph patterns, G_{S_i} that are subscriptions, in G_M that match some subgraph of G_P.

This matching problem is different from subgraph isomorphism [24]. The subgraph isomorphism problem is defined as follows: given graphs G_1 and G_2, identify all subgraphs of G_2 which are isomorphic to G_1. This differs from the problem we are trying to solve, which is to identify all subgraphs of G_2 that are isomorphic to *some* subgraph of G_1.

4.1 Data Structure

Since there can be multiple edges between the same pair of nodes, we use two-level hash tables to represent G_M. At the first level, we use a hash table to store all the pairs of vertices taking the names of the two nodes as the hash key. Each entry of the first hash table is a pointer to another (second-level) hash table that contains a list of all the edges between these two nodes. The edge label (i.e., "property" in the 5-tuple) is used as the hash key. Each edge points to a list of subscriptions that contain this edge.

Figure 7 shows the data structure of G_M. There are two edges between node A and B and both $s1$ and $s2$ contain the edge a between A and B.

Any subscription can contain multiple variables that can be matched by any

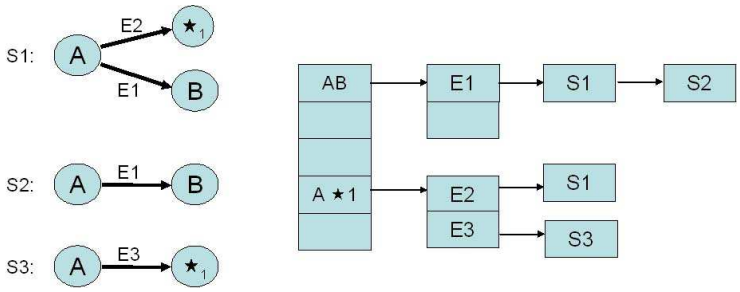


Fig. 7. Data Structure

vertex in the publication graph. For example, Figures 5(a) and 5(b) show two subscription graphs containing variables and the merged subscription graph, G_M, in Figure 5(d).

The data structure from Figure 7 allows us to store uniquely labelled nodes only once. In other words, nodes belonging to different subscriptions, but with the same label map to the same node in G_M. This is possible because each node in a graph is uniquely identified by its label. However, this is not the case with nodes with variable labels. Variable labels do not uniquely identify nodes, but instead they represent a (possibly constrained) pattern on node labels from a publication.

We introduce a special sequence of labels, $\star_i | i \geq 1$, to represent variables. The value of index i is bounded by the number of variables in the subscription with the most variables among all subscriptions in G_M.

For example, in Figure 5(d), we use one node labelled as $\star_1$ to represent both $?x$ and $?z$; $?x$ and $?y$ are represented by two nodes $\star_1$ and $\star_2$ since they appear in the same subscription. Mapping between original variable labels from the subscription (e.g., ?x) to the corresponding *star* name is preserved.

Mapping of variables from subscriptions to star labels is arbitrary for the sake of simplicity, even though some mappings are better than others since they can results in a sparser G_M. In the future, we are going to investigate how much can be gained, in terms of matching performance, by having a more sophisticated mapping.

4.2 Matching Algorithm

We use a graph G_M to contain all subscriptions. First, we discuss how G_M is created when inserting subscriptions. Suppose G_S is a subscription graph. $|G_S.\star|$ is the number of variables in the subscription graph, variable vertices in G_S are labelled as $\star_i$ where $0 < i < |G_S.\star|$. $G_M.\star$ is the number of stars in G_M. Note that all vertices in G_S and G_M are unique. $G_M.T1$ is the first-level hash

table, and $T2$ is the second-level hash table. $E.subs$ is a set of subscriptions containing edge E, $G_M.subs$ is the set of all subscriptions in G_M. E (and $E2$) is a directed edge from $E.v$ to $E.w$, $E.smEdge$ is an edge in G_M that overlaps with E. $newTable(A, B)$ creates a table with 2 columns A and B that will be used to decided on the bindings for variables.

Algorithm $Insert(G_S)$
1. **if** $G_S.\star > G_M.\star$
2. $\quad G_M.\star = G_S.\star$
3. **for** each edge $E \in G_S.edges$
4. $\quad T2 = G_M.T1.getTable(E.v, E.w)$
5. $\quad$ **if** ($T2$ is null)
6. $\quad\quad T2 = G_M.T1.insert(E.v, E.w)$
7. $\quad E2 = T2.getEdge(E)$
8. $\quad$ **if** ($E2$ is null)
9. $\quad\quad E2 = T2.insertEdge(E)$
10. $\quad E2.bindingTable = newTable(E.v, E.w)$
11. $\quad E2.subs = E2.subs + G_S$
12. $\quad G_M.subs = G_M.subs + G_S$
13. $\quad E.smEdge = E2$

Algorithm *Insert* is the procedure for subscription insertion. For each edge in G_S, we check if there is a corresponding edge in the first-level hash table. If there is no such edge, we update the hash tables by inserting $E.vE.w$ into the first-level hash table and inserting edge E into the corresponding second-level hash table. Finally, the subscription id is inserted into the list associated with edge E and added to $G_M.subs$.

Next, we explain how to perform matching using the subscription graph G_M when a publication arrives. G_P is the publication graph (the number of edges in G_E is m). G'_P is a completed graph containing vertices $E.v$, $E.w$, $\star_i$ such that $0 < i < |G_M.\star| + 1$. All nodes in G_P are unique. *SubSet* contains all subscriptions that have at least one edge in G_M that are referenced by G_P. *Result* is a set of (S,R) where S is a subscription and R is a satisfying binding for variables. Natural join ($\bowtie$) is an equality join on all common columns.

Algorithm $match(G_P)$
1. **for** each $E \in G_P.edges$
2. $\quad$ create a fully connected graph G'_P
3. $\quad$ **for** each edge $E2 \in G'_P$
4. $\quad\quad T2 = G_M.T1.getTable(E2.v, E2.w)$
5. $\quad\quad$ **if** ($T2$ not null)
6. $\quad\quad\quad E3 = T2.getEdge(E)$

```
7.                  if (E3 not null)
8.                      for all S ∈ E3.subs
9.                          S.edgeCount++
10.                         E3.bindingTable+ = (E.v, E.w)
11.                         SubSet = SubSet + E3.subs
12. result = 0
13. for all subscriptions S ∈ SubSet
14.     if (S.edgeCount ≥ |S.edges|)
15.         S.edgeCount = 0
16.         b = E.smEdge.bindingTable|E ∈ S
17.         for every edge E2 ∈ S.edges − E
18.             b = b ⋈ E2.smEdge.bindingTable
19.         for every row R ∈ b
20.             if CheckConstraint(R,C_S, T)
21.                 result = result + (S, R)
```

Algorithm *match* is the procedure for matching publications against subscriptions. There are two stages in the matching process. First, for each edge in the publication, we check all the corresponding subscription edges in G_M. Then we find the satisfying bindings for variables and evaluate the constraints.

In the first stage, for the publication edge v_1v_2, it can be matched by edges v_1v_2, $v_1\star_i$, $\star_iv_2$ and $\star_i\star_j$ in G_M. There are three actions to perform on these potentially matching edges. (1) Add v_1v_2 into the binding tables of all matching edges so that they can be used in the second stage. (2) Increase the counters of subscriptions associated with these edges. (3) Put the subscriptions into *Subset* as matching candidates. This completes the first stage of matching.

In the second stage, we find the matched subscriptions by checking the candidates in *Subset* one-by-one. For each subscription s_i in *Subset*, we join all the binding tables of edges belonging to s_i. If the result table is not empty, then the entries in the result table contain all valid binding values for all variables in the subscription.

Figure 8 provides an example for a binding table join. For example, the subscription contains two edges $A\star_1$ and $\star_1B$. There are three entries in the binding table of $A\star_1$ which means $A\star_1$ is matched by three edges AB, AC and AE in the publication. $\star_1B$ is matched by 5 edges in the publication. Joining of these two tables produces ACB and AEB and hence $\star_1$ can be bounded with value C and E.

After identifying all valid bindings of variables, we can use the binding value w to evaluate the constraint. For the constraint $(?x, op, v)$, we need to check whether $(w\ op\ v)$ is true. For the value filtering constraint, $(w\ op\ v)$ is evaluated using standard relational operator comparison.

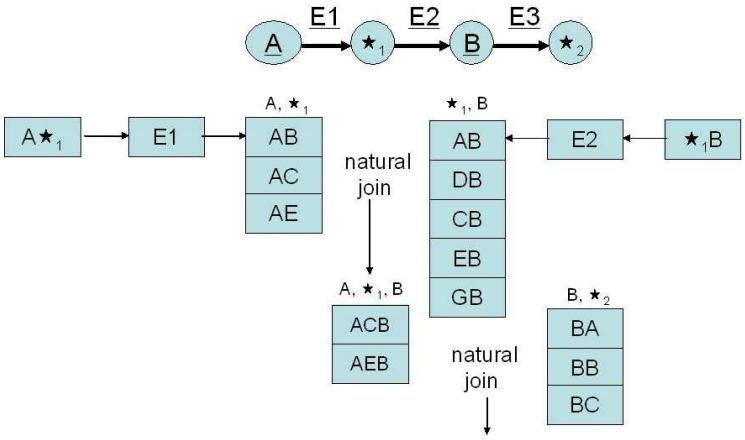


Fig. 8. Binding table join

For the class taxonomy filtering constraint $(w\ op\ v)$, we need to check the descendant-ancestor relationship between the specific instance w and the class v by traversing the taxonomy tree. The constraint checking algorithm is shown in Algorithm *CheckConstraint*.

Algorithm *CheckConstraint*(R, C_S, T)
1. **for** each variable $\star$ in S
2. find the value v in R and the constraint (op, c)
3. return isTrue(v, op, c, T)

Algorithm *isTrue*(v, op, c, T)
1. **if** $op = LT$ **return** isNodeDescendant(v, c, T)
2. **if** $op = GT$ **return** isNodeDescendant(c, v, T)
3. **if** $op = EQ$ **return** (c.equals(v))

For example, in Figure 5(d), for subscription s_2, $\star_2$ is matched by node "2005" since $2005 > 2004$ and $\star_1$ is matched by node "Home Page ♯325" since it is descendant of class "Academia."

4.3 Analysis

Space Complexity: The space cost mainly includes two parts: hash tables and linked lists associated with each edge to store the subscription ids that contain this edge. The size for the hash tables is determined by the number of unique edges among all the subscriptions. The length of the linked list depends on the average number of subscriptions each edge is associated with. Therefore, the space complexity is

$$O(|G_M.edgs| + |G_M.edgs| \times N_{s_e})$$

where $|G_M.edges|$ is the number of unique edges in matrix G_M and N_{S_e} is the average number of subscriptions each edge is associated with.

Time Complexity: For the procedure of insert a subscription into the system, he insert(G_S) algorithm iterates for every edge in the coming subscription, locate the corresponding list associated with the edge and add an entry of the coming subscription into the list. Thus, the insert algorithm depends on the number of edges for each subscription and the time complexity is

$$O(|G_S.edges|).$$

To form the graph G_M which contains all subscriptions, we have to insert subscriptions one by one. Therefore, the time to load a batch of subscriptions at a time is $\sum_{s_i} |G_{S_i}.edges|$. Since the number of edges in each subscription is very small, the time complexity of loading subscription is

$$O(number\_of\_subscriptions).$$

The matching algorithm consists of two stages. First is edge matching. By checking each edge in the publication, we determine all the subscriptions that have at least one edge matched by the publication. The time of the first stage depends on the size of the completed graph G'_P and the number of edges in the publication. Since each graph G'_P contains all the stars in G_M plus $E.v$ and $E.w$, the number of edges in G'_P is $\binom{k+2}{2}$. Suppose k is the number of stars in G_M, m is the number of edges in the publication, we have

$$O(m * 2\binom{k+2}{2}) \sim O(mk^2).$$

In the second stage, for each subscription in $SubSet$, if all the edges of it are matched, we perform a join operation on the binding tables to determine whether there is a satisfying binding for the variables, then we check the constraints. To join two tables, the time is linear with the size of the smaller table. The time complexity to find satisfying bindings of variables for each subscription is

$$O(k * l)$$

where k is the number of stars in G_M and l is the size of the smallest binding table for variables.

The time to check whether the constraint for the variable is satisfied according to the class taxonomy is dependent on the complexity of the taxonomy tree. Since multiple parents are allowed in the class taxonomy tree, the time is $O(d^t)$ where d is the depth of the tree and t is the average number of parents each node may have.

Overall, the matching time to evaluate all subscriptions is

$$O(mk^2) + O(n * k * l + n * k * d^t)$$

where n is the number of subscriptions in $SubSet$. In real applications, the class taxonomy tree is fixed, the number of variables in one subscription is small (usually 1 to 3, at most 5), $m << n$, and n is around the number of matched subscriptions. Therefore, the overall matching time is linear with the number of matched subscriptions:

$$O(ratio_{match} * number\_of\_subscriptions).$$

5 Evaluation

Table 1
The workload parameters in experiments

parameters	**default values**	**description**
$Size_P$	(35,90)	size of publication
$Size_S$	(5,35)	size of subscription
N_{sub}	30,000	number of subscriptions
$ratio_{match}$	0.1%	ratio of matched subscriptions among all
N_{stars}	2	number of stars (variables) in one subscription
N_{sub^*}	27,000	number of subscriptions containing stars
$overlap_s$	50%	ratio of overlap among subscriptions

We have implemented the algorithm in Java. We experimentally evaluate the rate of matching and the memory use. We run the experiments on a Linux system with 1GB RAM and a 1GHz microprocessor. We are using a synthetic workload so that we can independently examine various aspects of G-ToPSS. We report the results for the two most important metrics from a user's perspective, namely the rate of matching and the memory requirements. The workload parameters are shown in Table 1.

$Size_P$ and $Size_S$ are decided by (number of nodes, number of edges) the publication graph and the subscription graph. The number of edges must be larger than the number of nodes in order to obtain a connected graph. We use $ratio_{match}$ to control the number of matched subscriptions that are generated as subgraphs from the publication graph.

We generate the test workload using the parameter values from Table 1. A publication is generated first. For example, for publication of size (k,m) we

first generate a simple path of length $k-1$ and then we generate $m-k+1$ edges between random pairs of the k nodes.

Subscriptions are generated in four steps. 1.$ratio_{match}$ subscriptions that match the publication are generated by randomly selecting a subgraph of the publication. 2. Using same technique, overlapped subscriptions are generated as subgraphs from one big graph. 3. $N_{sub} * (1 - overlap_s)$ non-overlapping subscriptions are generated randomly in the same way that the publication was generated. 4. N_{stars} vertices are selected from all N_{sub^*} subscriptions and replaced with a variable ($\star$). Alternatively, we limit values that can be bound to a variable by adding constraints.

All measurements are performed after G-ToPSS has loaded all the subscriptions. We look at the effect of the number of subscriptions, subscription size and matching ratio (number of subscriptions matched by a publication). Finally, we compare G-ToPSS with two alternative implementations. For each experiment, we vary one parameter and fix the others to their default values as specified in Table 1.

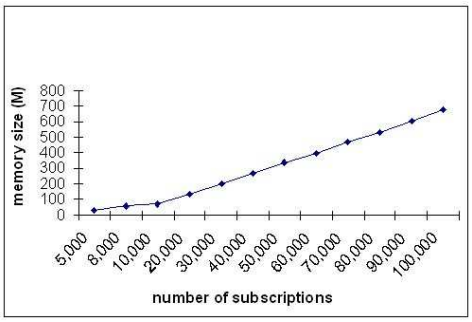
Number of subscriptions: Figure 9(a) shows the memory use with increasing number of subscriptions. We see that the memory size grows linearly as the number of subscriptions increase. Since all subscriptions in our experiments are of the same size and the overlap factor is constant, the memory increase per subscription is also a constant.

Figure 9(b) shows the time to find all matches for a publication given a fixed set of subscriptions. As the set of subscriptions increases, so does the time. The number of subscriptions that match the publication is relative to the total number of subscriptions in the set. Consequently, the number of matches increases as the number of subscriptions increases.

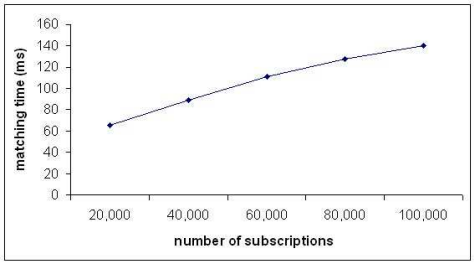
The time to match a publication is split between structure matching phase and constraint evaluation phase. As the number of subscriptions increases, both of these times increase by a fixed amount because the number of matches increases constantly.

Subscription size: Figure 9(c) shows how the space used by the subscriptions decreases as the overlap between them increases. We present this to validate our workload. The matrix space is the size of G_M, while *whole memory* is equal to the size of G_M plus the space used to store all the subscriptions.

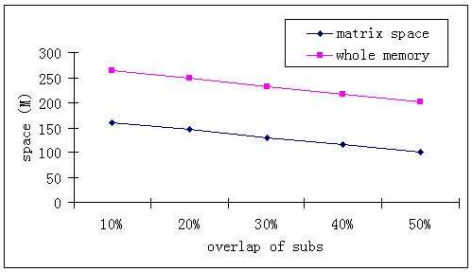
Figure 9(e) shows the effect of increasing subscription size on the matching time. We see that the time increases more rapidly as the number of edges increases (e.g., from 4 to 8), the time almost doubles. On the other hand, as the number of edges increases slowly, so does the increase of matching time, hence the matching time is not affected by the number of nodes, but by the



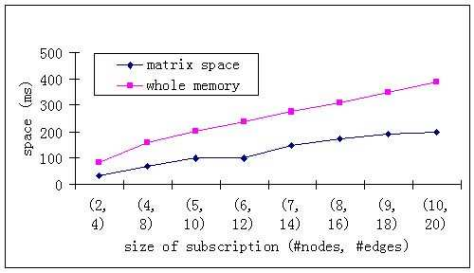
(a) Memory vs. #subscriptions



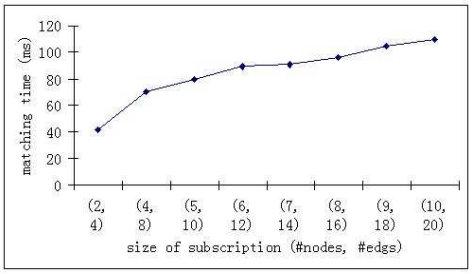
(b) Matching time vs. #subscriptions



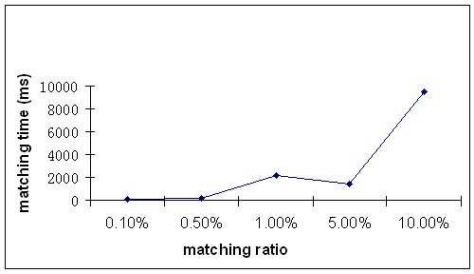
(c) Memory vs. subscription overlap



(d) Memory vs. subscription size



(e) Matching time vs. subscription size



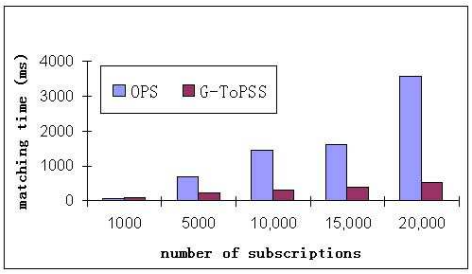
(f) Matching time vs. matching ratio

Fig. 9. Experimental performance results

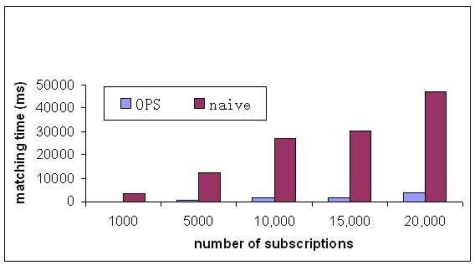
number of edges in the subscription.

Matching ratio: Figure 9(f) shows the effect of increasing the number of subscriptions that match the publication. As this number grows, the time to match grows very rapidly. This is mainly due to increase in time to calculate all the bindings for each subscription.

G-ToPSS vs. Alternatives: In Figure 10(a) we compare the performance of



(a) G-ToPSS vs. OPS



(b) OPS vs. naive

Fig. 10. Compare G-ToPSS with other algorithms

our algorithm to the OPS algorithm [25]. As the graph shows, OPS matching time increases very rapidly with the number of subscriptions. The main reason for the significant difference in matching times comes from the differences in basic assumptions. The OPS algorithm makes the same basic assumption as do other, traditional, subgraph isomorphism algorithms [24], namely that every node in a subscription is a variable. In other words, any node of a publication can match with any other node in the subscription graph. However, this assumption unnecessarily increases the matching complexity, as we see in the evaluation. We make a more realistic assumption that the number of variables in any subscription is low as compared to the total number of nodes in a subscription graph and the nodes in a RDF publication are unique.

Figure 10(b) illustrates that, even though OPS is less scalable than G-ToPSS, it is still far better then a naive approach which sequentially checks all subscriptions to find the matching ones.

6 Application

Recent years have seen a rise in the number of unconventional publishing tools on the Internet. Tools such as wikis, blogs, discussion forums, and web-based content management systems have experienced tremendous rise in popularity and use; primarily because they provide something traditional tools do not: easy of use for non computer-oriented users and they are based on the idea of "collaboration." It is estimated, by `pewinternet.org`, that 32 million people in the US read blogs (which represents 27% of the estimated 120 million US Internet users) while 8 million people have said that they have created blogs.

Web-based collaboration is the common idea for this new breed of content-management tools. The center piece of such tools is a web page that is being used as an area where multiple users participate in content creation. More significantly, the collaboration enabling tool used is the web page itself (accessed

through the all-pervasive web browser).

With these new web applications, there rouse a need for users to stay informed about changes to the content. In general, users want to be updated about daily news headlines of interest to them, or be notified when there is a reply in a discussion they participate in, or their favorite web personality has updated his/her blog (online diary etc.).

RSS [4] is quickly becoming the dominant way to disseminate content update notifications on the Internet. `pewinternet.org` reports that 6 million people in the US use RSS aggregators (a service/application that monitors large numbers of RSS feeds). [5]

Web-based content management systems (CMS) have also grown in popularity mainly because they are based on the publishing tools just described, but also because they are much easier to use and maintain than traditional CMS. [6] Like traditional CMS systems, they provide content access control, user profiles, persistent storage, web access, RSS authoring, advanced content management, content routing and taxonomic content classification.

In this section, we describe an extension to content management systems, CMS-ToPSS, for scalable dissemination of RSS documents, based on the publish/subscribe model. To illustrate the effectiveness of the system, we extend an existing open-source web-based content-management system, Drupal (`drupal.org`) to use CMS-ToPSS in a manner that is transparent to end users, yet provides an efficient content-routing architecture.

CMS-ToPSS consists of three main components: The Drupal module (a conent management system), the G-ToPSS filtering service and connector between them. The overall architecture is shown in Figure 11. The Drupal module acts as a client to the filtering service. The module does not require any changes to Drupal, and any Drupal installation can experience the benefits of CMS-ToPSS by simply retrieving and installing the module.

G-ToPSS filtering service is accessible via XML-RPC and can be accessed by the XML-RPC client. The CMS-ToPSS connector reads RSS feeds and serializes them into publications and subscriptions as input to G-ToPSS. In Figure 13 we show an example of an RSS feed (ie.e, a publication) and a subscription is shown in Figure 12. Both publications and subscriptions are

[4] `web.resource.org/rss/1.0/spec`

[5] Reported by Pew Internet & American Life Project (www.pewinternet.org), an organization that produces reports that explore the impact of the Internet on families, communities, the daily life. Also reported by "RSS at Harvard Law" (blogs.law.harvard.edu/tech/)

[6] Mid Market Web CMS Vendors Pull Ahead. Brice Dunwoodie. `CMSwire.com`

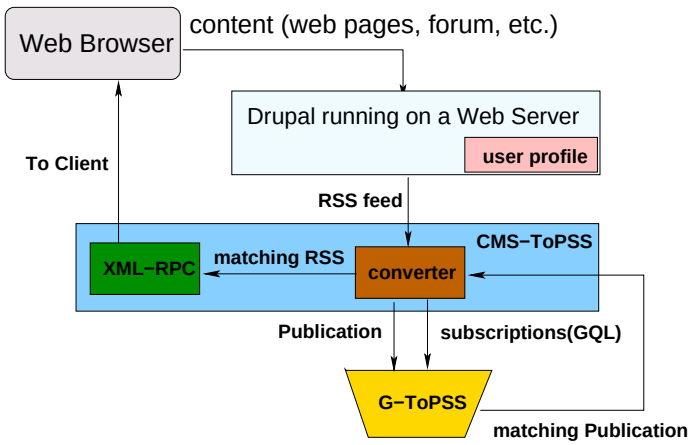


Fig. 11. CMS-ToPSS system architecture

RSS feeds. And subscriptions are differentiated by the key word *GQL* in *title* and the query can be taken out from *description*.

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://purl.org/rss/1.0/">
    <channel rdf:about="http://www.xml.com/xml/news.rss">
        <title>XML.com</title>
        <link>http://xml.com/pub</link>
        <description> XML.com features a rich mix of information and services for
the XML community. </description>
        <image rdf:resource="http://xml.com/universal/images/xml_tiny.gif"/>
        <items>
            <rdf:Seq>
              <rdf:li resource="http://xml.com/pub/2000/08/09/xslt/xslt.html"/>
              <rdf:li resource="http://xml.com/pub/2000/08/09/rdfdb/index.html"/>
            </rdf:Seq>
        </items>
    </channel>
    <item rdf:about="http://xml.com/pub/2000/08/09/rdfdb/index.html">
        <title>GQL</title>
        <link>http://xml.com/pub/2000/08/09/rdfdb/index.html</link>
        <description>
            SELECT ?x
             WHERE (Homepage325, Project, G-ToPSS), (G-ToPSS, Supervisor,
Arno Jacobsen), (G-ToPSS, YEAR, ?x)
            SUCHTHAT (?x > 2000) </description>
    </item>
</rdf:RDF>
```

Fig. 12. Subscription example

Upon receiving a publication and subscriptions, G-ToPSS performs the matching between them and the outputs are notifications which are also serialized as RSS feeds over the converter back to the client via XML-RPC. Each subscription that a user submits is, in fact, a distinct RSS feed (containing items matching the user's subscription).

The Drupal module performs both subscribing and publishing based on user interaction with Drupal CMS. User can easily generate an RDF document using our template and publish to G-ToPSS. Also user can form a subscription with specified constraints from the interaction panel and send it to G-ToPSS. The module serializes all content changes in Drupal using RSS and sends them

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns="http://purl.org/rss/1.0/">
    <channel rdf:about="http://www.xml.com/xml/news.rss">
        <title>XML.com</title>
        <link>http://xml.com/pub</link>
        <description> XML.com features a rich mix of information and services for
the XML community. </description>
        <image rdf:resource="http://xml.com/universal/images/xml_tiny.gif"/>
        <items>
            <rdf:Seq>
              <rdf:li resource="http://xml.com/pub/2000/08/09/xslt/xslt.html"/>
              <rdf:li resource="http://xml.com/pub/2000/08/09/rdfdb/index.html"/>
            </rdf:Seq>
        </items>
    </channel>
    <image rdf:about="http://xml.com/universal/images/xml_tiny.gif">
        <title>XML.com</title>
        <link>http://www.xml.com</link>
        <url>http://xml.com/universal/images/xml_tiny.gif</url>
    </image>
    <item rdf:about="http://xml.com/pub/2000/08/09/xslt/xslt.html">
        <title>Processing Inclusions with XSLT</title>
        <link>http://xml.com/pub/2000/08/09/xslt/xslt.html</link>
        <description> Processing document inclusions with general XML tools can
be problematic. This article proposes a way of preserving inclusion information
through SAX-based processing. </description>
    </item>
    <item rdf:about="http://xml.com/pub/2000/08/09/rdfdb/index.html">
        <title>Putting RDF to Work</title>
        <link>http://xml.com/pub/2000/08/09/rdfdb/index.html</link>
        <description> Tool and API support for the Resource Description Frame-
work is slowly coming of age. Edd Dumbill takes a look at RDFDB, one of the most
exciting new RDF toolkits. </description>
    </item>
</rdf:RDF>
```

Fig. 13. RSS feed example

to the G-ToPSS filter service. The filtering service forwards the document to the interested clients which could be other XML-RPC clients as well as other Drupal modules. Note that the G-ToPSS filtering service can serve multiple Drupal sites.

In addition to publishing all content changes in RSS, the Drupal module also extends different kinds of Drupal content with change notification capabilities. For example, users can subscribe to receive notifications when they have replies on the discussion forum, or when a certain web page in Drupal has been updated. The Drupal module registers these kinds of subscriptions with the G-ToPSS filtering service transparently to the user.

A user, using a web browser, accesses a Drupal site that is extended with the module described in this paper. The user can choose to receive notifications for content of her choice (e.g., discussion forum replies, web page updates etc.) Drupal supports convenient taxonomic content classification, which can be directly mapped to a G-ToPSS ontology. In this case, the user will get notifications only when both the content and taxonomic constraints of her subscription are satisfied. The users can also create content (e.g., participate in a discussion form or create/update a web page) to trigger notifications. The users' subscriptions are stored as part of their Drupal profile. Via the profile web page, users can review their notification requests as well as see all notifications received for those requests.

We also allow users to subscribe directly on the RSS content by expressing

Fig. 14. CMS-ToPSS User API

their subscriptions in G-ToPSS's SQL-like subscription language (GQL). The subscriptions and their results are also shown as part of the user profile. The screenshot of the user interface is shown in Figure 14.

7 Conclusions and Future Work

Use of RDF as a language for representing metadata is growing. Applications such as RSS and content management are exhibiting use patterns that current systems were not designed for.

The G-ToPSS prototype shows that a data-centric, push-based architecture such as a publish/subscribe system is a very good fit for just such applications (as illustrated by CMS-ToPSS described in Section 6). G-ToPSS is able to support high matching rates for very complex subscriptions. In practice, we expect these subscriptions to be simpler (i.e., have smaller number of edges and stars) on average than the ones used in our experiments.

Being based on RDF, G-ToPSS can be easily extended to use additional semantic information expressed in languages built on top of RDF, such as RDFS and OWL. We show how a RDFS taxonomy can be used to increase the expressiveness of the G-ToPSS query language. Our implementation uses an efficient traversal of the class hierarchy with support for multiple inheritance, which adds more expressiveness to the language without unduly affecting the matching rate. On the other hand, more powerful inference techniques such as those of Descriptions Logics (on which OWL is based) could augment the constraint filtering without significant changes to the matching engine.

In the future, we will work on extending G-ToPSS with full RDF language

features (such as bags and sequences), which we have left out since their implementation does not affect the matching rate but merely adds syntactic sugar.

Extending G-ToPSS to support variables on predicates is straight forward since the same techniques for supporting variables on subjects and objects can be used. Consequently, matching time complexity is not affected by this extension.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching events in a content-based subscription system. In *Symposium on Principles of Distributed Computing*, pages 53–61, 1999.

[2] M. Altinel and M. J. Franklin. Efficient filtering of XML documents for selective dissemination of information. In *Proceedings of the 26th VLDB Conference*, 2000.

[3] I. Burcea, H.-A. Jacobsen, E. de Lara, V. Muthusamy, and M. Petrovic. Disconnected operation in publish/subscribe middleware. In *Mobile Data Management*, pages 39–, 2004.

[4] M. Cai, M. R. Frank, B. Yan, and R. M. MacGregor. A subscribable peer-to-peer rdf repository for distributed metadata management. *J. Web Sem.*, 2(2):109–130, 2004.

[5] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[6] C. Y. Chan, P. Felber, M. N. Garofalakis, and R. Rastogi. Efficient filtering of XML documents with XPath expressions. *The VLDB Journal*, 11:354–379, 2002.

[7] P.-A. Chirita, S. Idreos, M. Koubarakis, and W. Nejdl. Publish/subscribe for rdf-based p2p networks. In *ESWS*, pages 182–197, 2004.

[8] M. Cilia, C. Bornhoevd, and A. P. Buchmann. CREAM: An Infrastructure for Distributed Heterogeneous Event-based Applications. In *Proceedings of the International Conference on Cooperative Information Systems*, pages 482–502, 2003.

[9] G. Cugola, E. D. Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 27:827–850, sep 2001.

[10] Y. Diao, P. Fischer, M. Franklin, and R. To. Yfilter: Efficient and scalable filtering of XML documents. In *Proceedings of ICDE2002*, 2002.

[11] K. G. C. (ed). RDF Data Access Use Cases and Requirements. *W3C Working Draft*, 2004.

[12] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. Ross, and D. Shasha. Filtering algorithms and implementation for very fast publish/subscribe systems. In *SIGMOD Conference*, 2001.

[13] E. Fidler, H. A. Jacobsen, and G. Li. The padres distributed publish/subscribe system. In *8th International Conference on Feature Interactions in Telecommunications and Software Systems*, Leicester, UK, 2005.

[14] A. K. Gupta and D. Suciu. Stream processing of xpath queries with predicates. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 419–430, New York, NY, USA, 2003. ACM Press.

[15] V. Haarslev and R. Moller. Incremental Query Answering for Implementing Document Retrieval Services. In *Proceedings of the International Workshop on Description Logics*, 2003.

[16] G. Li, S. Hou, and H. A. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. *International Conference on Distributed Computing Systems (ICDCS'05)*, 2005.

[17] G. Li and H. A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *ACM/IFIP/USENIX 6th International Middleware Conference*, Grenoble, France, 2005.

[18] H. Liu and H.-A. Jacobsen. A-ToPSS - a publish/subscribe system supporting approximate matching. In *Very Large Databases (VLDB'02)*, University of Toronto, August 2002.

[19] V. Muthusamy, M. Petrovic, and H.-A. Jacobsen. Effects of routing computations in content-based routing networks with mobile data sources. In *the Eleventh Annual International Conference on Mobile Computing and Networking*, Cologne,Germany, 2005.

[20] M. Petrovic, I. Burcea, and H.-A. Jacobsen. S-ToPSS - a semantic publish/subscribe system. In *Very Large Databases (VLDB'03)*, Berlin, Germany, September 2003.

[21] M. Petrovic, H. Liu, and H.-A. Jacobsen. CMS-ToPSS - efficient dissemination of rss documents. In *Proceedings of 31st International Conference on Very Large Data Bases (VLDB). (demo)*, September 2005.

[22] M. Petrovic, H. Liu, and H.-A. Jacobsen. G-ToPSS - fast filtering of graph-based metadata. In *the 14th International World Wide Web Conference*, Chiba, Japan, May 2005.

[23] M. Petrovic, V. Muthusamy, D. Gao, and H.-A. Jacobsen. Publisher mobility in distributed publish/subscribe systems. In *DEBS Workshop at ICDCS,Columbus, Ohio*, 2005.

[24] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.

[25] J. Wang, B. Jin, and J. Li. An Ontology-Based Publish/Subscribe System. In *Middleware*, 2004.

[26] Z. Xu and H. A. Jacobsen. Efficient constraint processing for location-aware computing. In *6th International Conference on Mobile Data Management (MDM'05), Ayia Napa, Cyprus*, 2005.