Collaborative Building, Sharing and Handling of Graphs of Documents Using P2P File-Sharing

Alan Davoust and Babak Esfandiari

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, Ontario, Canada

Abstract. We are interested in creating a peer-to-peer infrastructure for the collaborative creation of knowledge, with no centralized point of control. We show how documents in a P2P file-sharing network can be interlinked, using a naming scheme based on the document schema and content, rather than on the document location. The interlinked documents can be seen as a distributed graph of documents, for which we define a class of graph queries supported by our file-sharing system.

Keywords: Peer-to-peer, file-sharing, graph queries, Semantic Web.

1 Introduction

Collaborative online knowledge construction efforts have gained momentum with the arrival of "Web 2.0" technologies, and they represent a crucial step toward realizing Tim Berners-Lee's vision of the Semantic Web [1]. Projects such as [2] rely on a centralized wiki repository where ontologies can be created and modified. The issue with such approaches is that there is ultimately a central decision point (and a single point of failure) and only one "version of the truth" can be held. Distributed efforts relying on peer data management systems [3], [4] still aim to provide a single consistent view of data through data integration mechanisms such as ontology mappings.

Our work is based on the principles of peer-to-peer (P2P) file-sharing systems such as such as Napster[1], or eMule[2], where peers act both as clients and servers, and where documents downloaded by one peer are also immediately made available by that peer. This means that more popular documents will be more heavily replicated and will be more available regardless of the turnover (arrival and departure) of peers. It also means that each peer will provide a subset of all available documents, a subset that is potentially different from those served and used by other peers.

U-P2P [5,6] is a schema-based file-sharing system, where communities of peers share documents structured according to a particular metadata schema, comparable to the music-specific metadata schema of MP3 files. Each peer can manage

[1] http://www.napster.com, no longer existing in its original form
[2] http://www.emule-project.net

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 888–897, 2009.

multiple schemas, but also create new schemas which can be themselves shared on the network, and discovered and downloaded by other peers. Peers who use the new schema thus form a new community.

However, in this model, each community defines an independent and unrelated collection of documents. The next step that we see towards representing knowledge is the possibility of interlinking documents. Interlinked documents can be seen as a navigable graph, that users may "browse", navigating from one concept to another related concept. The side-effect of this navigation is the creation of a personal sub-graph or view of the world, made available for others to build upon.

Our approach to links, previously presented in [7,8] is based on a P2P-specific URI scheme with a search-based dereferencing protocol: documents are identified by their community and content, rather than by their location, as in web-based systems. In this paper we show how the resulting graph of documents can be queried and shared using a query language similar to SPARQL.

The rest of this paper is organized as follows. We first introduce our naming scheme in Section 2, then present a class of graph queries supported by U-P2P in section 3.

2 URIs and Endpoints in U-P2P

2.1 Formal Foundations

We briefly recall the formal model of U-P2P, that was previously presented in [9].

Communities. Our existing prototype U-P2P provides a framework with multiple communities, where each community has its specific schema, which defines the format of the documents shared within that community, its protocol, which defines how (and therefore *which*) peers can be reached by a query from another peer of the community, and finally its own name, which is simply an identifier of the commmunity.

By analogy with the formal model of relational databases, we define each document shared in a community as a tuple $(x_1, \ldots, x_n)$.

The schema S_C of a community C is a single n-ary relation R_C made up of n ($n \geq 2$) attributes A_i. The domain of R_C is a fixed infinite set $\mathcal{D}_C$, and we will detail further the subdomains of the attributes A_i.

Intuitively, the tuples in the extension of R_C correspond to the documents shared in the community. Each Peer p member of the community C hosts an extension R_C^p of R_C.

We distinguish two types of attributes for R_C: *metadata* attributes and *attachments*. Metadata attributes are attributes with standard types such as strings and numbers, whereas attachments are attributes that have a "binary" domain, in the sense that there is no simple human readable representation for these attributes.

In particular, R_C has an attribute *name* that uniquely identifies each document in R_C, i.e. in each community schema there is a primary key attribute that

we will call *name*. In U-P2P, this attribute is generated using a one-way hash function using the metadata attributes of the document.

Intuitively, as this formalization is meant to model traditional file-sharing communities, attachments are attributes containing the binary payload of a file. In our view a file and its metadata form a single data item, which we call a document.

The set of peers $\{p_i\}$ reachable from a given peer p can be represented by a function $Prot_C(p{:}\text{peer})$ (the procedural aspects of the protocol are not modeled). For example, the Gnutella protocol with a time-to-live of 7 can be modeled by a function that returns all the peers connected to p by a path shorter than 7 hops.

File-Sharing Operations. Each peer p member of a community C defines an interface to C with the following fundamental file-sharing operations:

- Publish(*doc*: document): add the document *doc* to R_C^p
- Delete(*doc*: document): if *doc* is in R_C^p, remove *doc* from R_C^p.
- Search(*expr*: expression): the expression *expr* is a first-order logic formula of at most n free variables, involving the attributes of R_C, this search returns tuples $\{(p_i, name_j, x_{m_1}, \ldots x_{m_k})\}$, where $(name_j, x_{m_1}, \ldots x_{m_k})$ are the metadata attributes of a document d_j, p_i is a peer such that $p_i \in Prot_C(p)$ and $d_j \in R_C^{p_i}$.

  Intuitively, these search results describe documents matching the search query and available in the network (reachable via the protocol), with the identifiers of the peers that store these documents, which can then be used to download the file.
- Download($name_0$: document identifier, p_1 : peer identifier) : copies the document identified by $name_0$ from $R_C^{p_1}$ to R_C^p.
- Lookup($name_0$: document identifier, *attr*: attribute): returns the value of attribute *attr* in the document doc_{name_0} uniquely identified by $name_0$.

Multiple Communities. In U-P2P, communities are themselves described in structured documents, shared in a specific community called the "Commmunity of Communities", which we will note $C_{Community}$.

The schema of this community is the relation $R_{Community}$ with the attributes $(name, schema, protocol)$. Each document shared in this community thus represents a community, with its unique *name*, and representations of its *schema* and *protocol*. In U-P2P, we have added additional attributes to this community, including community-specific presentation templates, and a description of the community. On downloading such community definition documents, a peer automatically *joins* the described community, and thus acquires the capability of sharing documents of that community.

2.2 URI Scheme

Within our multiple-community framework, all the instances of a particular document (replicated across the P2P network) are interchangeable and uniquely

identified within their community by their *name* (which we recall, is a hash obtained from the document metadata), hence fully identified within a P2P system by the pair (*community name*, *document name*).

We can thus introduce a URI scheme specific to our framework, by concatenating the identifier "up2p" of our scheme, with the community and document names, separated by a slash character.

In UP2P, we generate unique names of documents using an MD5 hash function, and represent its output as a 32-character hexadecimal string. We obtain 70-character document URIs such as the following example:
`up2p:b6b6f4dd9cd455bc647af51e03357156/0192c7eb042bae9aaafa3b0527321938`.

2.3 Endpoint Attributes

In order to make use of this URI in our model, as a way to link documents, we now introduce a new type of attribute, defined by its domain, similarly to *metadata* and *attachments*.

We define the fixed, infinite set C-NAMES of possible community names (i.e. the domain of the attribute *name* in the community $C_{Community}$), and the fixed, infinite set D-NAMES as the union, for all possible communities, of the domains of their *name* attributes.

Definition 1. *An attribute* att *is an* endpoint attribute *iff its domain is C-NAMES* $\times$ *D-NAMES. (cartesian product).*

We note that this definition is of purely theoretical interest. In practice the schema of a community can be annotated to specify that a given attribute is an endpoint (or an attachment, or a metadata attribute); and a system using such endpoints would only validate the syntax of the URI.

2.4 Dereferencing Protocol

Dereferencing a URI, i.e. obtaining a document from its URI, can be done using the basic P2P File-Sharing operations *search* and *download* defined in Section 2.1.

The procedure is given by the pseudocode algorithm 1.

This search-based dereferencing mechanism may have a high cost, but we note that the search is specific to a community, which reduces the search space, as all peers in a P2P system will not necessarily be members of all communities. Furthermore, identifying a document by its content rather than its location provides the opportunity of getting any one of multiple copies, which makes the dereferencing more robust to churn, in the sense that when a peer leaves a community, the documents that it has published in the community may still be available from other peers.

2.5 Example

We introduce two communities, one community of documents describing movie actors, and a community of documents describing films.

Algorithm 1. URI dereferencing algorithm

The peer performing the dereferencing is denoted by p.
function DEREFERENCE(up2p:id1/id2):
  if p not member of C_{id1} **then** ▷ C_{id1} is the community uniquely identified by the name $id1$
    $\{(p_i, id1)\}_{i=1...m} \leftarrow C_{Community}$.SEARCH(name=$id1$)
    $j \leftarrow$ select in $[1 \dots m]$ ▷ select a peer
    $Community$.DOWNLOAD(p_j , $id1$); ▷ p joins C_{id1}
  end if
  $\{(p_i, id2)\}_{i=1...w} \leftarrow id1$.SEARCH(documentId=$id2$);
  $k \leftarrow$ select in $[1 \dots w]$ ▷ select a peer
  $id1$.DOWNLOAD(p_k , $id2$);
  return doc_{id2} ▷ doc_{id2} is the document of C_{id1} uniquely identified by the name $id2$.
end function

The schema of the community C_{film} would be the relation R_{film} with attributes $(name, title, director, year, binaryfilm)$.

The schema of C_{actor} could be R_{actor} with attributes: $(name, actorname, filmography)$, where $filmography$ could be a multi-valued endpoint attribute, referencing films that this actor has played in, themselves documents shared in the community C_{film}.

We could then model the actors Elijah Wood and Liv Tyler, who played in several films of the "Lord of the Rings" trilogy. The documents representing Elijah Wood and Liv Tyler could then be[3]:

(livtyler, 'Liv Tyler', `up2p:film/twotowers`*)*
(elijahwood, 'Elijah Wood', `up2p:film/twotowers`, `up2p:film/fellowship`*)*

The documents describing the films would be:

(twotowers, 'The two Towers', 'Peter Jackson', [binary film representation])
(fellowship, 'The Fellowship of the Ring', 'Peter Jackson', [binary film representation])

The graph induced by these example documents can be represented in the UML object diagram of Figure 1. Note that the diagram does not include all the attributes of the documents, as it would overload it.

3 Graph Queries in U-P2P

In the graph of data induced by endpoints, where vertices represent documents and edges represent endpoint attributes, we consider the class of graph queries defined by *path patterns*: given an input document d and a path pattern p, the query returns all documents connected to d by a path matching p.

[3] The *name* attribute of each document should be generated by a MD5 hash, but we use reader-friendly names here to make the examples more readable.

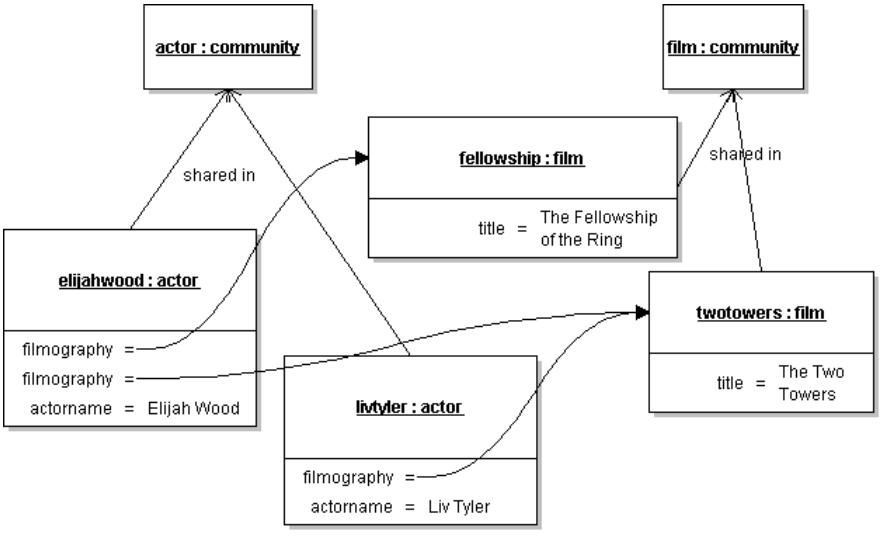

Fig. 1. An Example graph of related U-P2P documents

The simplest patterns p define paths of length 1, i.e. a single edge. We first describe how such elementary queries can be answered, then we build on this basis to show how arbitrary path pattern queries can be answered.

3.1 Edge Pattern Queries

In the graph of data, we define a predicate to describe the edges of the graph.

Definition 2. *We define the predicate* edge *of arity 4, as follows:*

$edge(doc_1, comm, endpoint, doc_2)$ *is true if the attribute* endpoint *of document* doc_1 *in community* comm *has the value* uri_2*, where* uri_2 *is the URI of document* doc_2*.*

We define queries based on the predicate *edge* by replacing one, both, or none of doc_1 and doc_2 by free (i.e. universally quantified) variables or by bound (i.e. existentially quantified) variables. We do not consider the possibility of the community or endpoint name being variables.

We note that the reason to constrain the community of doc_1 (i.e. to include the community in the edge definition), rather than the community of doc_2, or both, is that the endpoint attribute that characterises the edge is defined within the context of the community schema of doc_1.

For the sake of simplicity, we focus on queries with one free variable, and constants, as these will be the basis of our following generalization in section 3.2. Queries with a single free variable return documents, extending the functionality of simple SEARCH queries, which is a fundamental part of file-sharing.

We consider the following simple declarative queries. The lower-case names denote constants, and the upper-case X and Y denote variables.

$$\{X \mid edge(doc_1, comm, endpoint, X)\} \quad (1)$$

$$\{X \mid edge(X, comm, endpoint, doc_2)\} \quad (2)$$

The above queries can be respectively interpreted as: "find all documents referenced by doc_1 through the attribute *endpoint*", and "find all documents in community *comm* referencing doc_2 through the attribute *endpoint*".

We will thus focus on those queries, for which U-P2P implements a query answering algorithm.

The answers to other types of queries can easily be derived from minor modifications of our query answering algorithm, but we do not expand on this here for a lack of space.

In Algorithms 2 and 3, we detail the query answering procedures for queries 1 and 2, using the basic P2P File-Sharing operations *search* and *download*, and *lookup*. We assume that *doc* is given as a URI.

Algorithm 2. Algorithm for answering query 1.

```
function AnswerQuery2(doc, comm, endpoint)
    dereference(doc)                          ▷ ensure that doc is stored locally
    uri-list ← comm.lookup(doc, endpoint)     ▷ endpoint may be multi-valued
    for all uri ∈uri-list do
        results ← results ∪ dereference(uri)
    end for
    return results.
end function
```

Algorithm 3. Algorithm for answering query 2.

```
function AnswerQuery1(comm, endpoint, doc)
    if p not member-of comm then
        {(p_i, comm)}_{i=1...m} ← Community.search(communityId=comm)
        j ← select in [1...m]                                  ▷ select a peer
        Community.download(p_j , comm);
    end if
    results ← comm.search(endpoint = doc)
    return results.
end function
```

3.2 General Graph Queries

Based on these elementary *edge* queries, we now define a class of graph queries, representing sequences of edges, i.e. paths of arbitrary length in the graph of documents. This is the class of queries supported by U-P2P.

Preliminaries. Intuitively, an *edge* query is a pattern that matches directed edges between two documents, and for each matching edge, query 1 returns the document at the *start* of the edge, whereas query 2 returns the document at the *end* of the edge.

In order to unify these two types of queries, we introduce a parametric form of the *edge* predicate, where the boolean parameter will indicate which document the query should return. We define the parametric predicate $edge_{dir}$, where the parameter *dir* can take the values r (for "right") or l (left), as follows:

$$edge_l(X, c, e, Y) =_{def} edge(X, c, e, Y) \quad (3)$$

$$edge_r(X, c, e, Y) =_{def} edge(Y, c, e, X) \quad (4)$$

We can now rewrite our queries 1 and 2 as follows:

$$\{X \mid edge(doc, comm, endpoint, X)\} = \{X \mid edge_r(X, comm, endpoint, doc)\} \quad (5)$$

$$\{X \mid edge(X, comm, endpoint, doc)\} = \{X \mid edge_l(X, comm, endpoint, doc)\} \quad (6)$$

Path Pattern Graph Queries

Definition 3. *The class of* path pattern queries *is defined by the following general declarative form:*

$$\begin{aligned} \{X \mid \exists(Y_0, \ldots, Y_n), \quad & edge_{b_0}(X, comm_0, endpoint_0, Y_0) \\ \textstyle\bigwedge_{i=1}^{n-1} \; & edge_{b_i}(Y_{i-1}, comm_i, endpoint_i, Y_i) \\ \wedge \quad & edge_{b_n}(Y_n, comm_n, endpoint_n, doc)\} \end{aligned}$$

where the parameters of the query are:

- $(b_0, \ldots, b_n)$ *are boolean parameters (i.e. they take one of the two values 'r' or 'l') denoting one of the two forms of* $edge_{dir}$ *queries (queries 5 and 6);*
- $(comm_0, \ldots, comm_n)$ *are communities (may contain multiple occurences of a given community);*
- $(endpoint_0, \ldots, endpoint_n)$ *are endpoint attribute names, with the constraint that* $endpoint_i$ *must be an attribute of the schema of community* $comm_i$*;*
- *doc is a document*

Such queries define paths of length n in a graph of documents, starting from the input document *doc*.

Example. Based on the example of Section 2.5, involving Movies and Actors, we give the declarative form of a query for an actor who have played in the same film as the actor Kevin Bacon, given as an input to the query in the form of the URI up2p:film/kbacon:

$$\begin{aligned} \{X \mid \exists Y_0, \quad & edge_l(X, actor, filmography, Y_0) \\ \wedge \; & edge_r(Y_0, actor, filmography, \text{up2p:actor/kbacon}) \end{aligned}$$

We note that this query definition could easily be extended by some additional edges to represent the famous "six degrees of separation" concept between movie actors, as made popular by the "Kevin Bacon Game".

Query Answering Algorithm. We now detail the general algorithm to answer a *path pattern query*, using the notations of the above definition.

The general methodology is based on the linear form (paths) of the considered graph patterns: intuitively, the edges defined by each *edge* predicate form a path, i.e. a sequence with a natural order.

The query can be answered by starting from the "end" of the path, indicated by the input variable *doc* in the last occurence of the $edge_{dir}$ predicate, i.e. the occurence associated with the index $i = n$. The query is answered by answering each of the subqueries associated with the index values $i = 1 \dots n$, starting with n then iterating in reverse to 1. Each subquery is answered using either Algorithm 2 or 3.

The answers to the subquery associated with index i are documents that can then be input to the query at index $i - 1$, and so on until the last step, where the answers are returned.

This algorithm is listed in Algorithm 4.

Algorithm 4. Path Pattern Graph Query Answering Algorithm

function AnswerPathQuery($[b_0, \dots, b_n]$, $[comm_0, \dots, comm_n]$, $[endpoint_0, \dots, endpoint_n]$, *doc*)
  if $dir = l$ **then**
    $Y_{list} \leftarrow$ AnswerQuery1($comm_n$, $endpoint_n$, *doc*)
  else if $dir = r$ **then**
    $Y_{list} \leftarrow$ AnswerQuery2(*doc*,$comm_n$, $endpoint_n$)
  end if
  if $n = 0$ **then**
    return Y_{list}
  else
    for all $Y \in Y_{list}$ **do**
      $answer_{list} \leftarrow answer_{list} \cup$ AnswerPathQuery($[b_0, \dots, b_{n-1}]$, $[comm_0, \dots, comm_{n-1}]$, $[endpoint_0, \dots, endpoint_{n-1}]$, Y)
    end for
    return $answer_{list}$
  end if
end function

4 Conclusions

We presented in this paper a graph data model for documents in a P2P file-sharing network, and we showed a way to contribute to such a graph in a collaborative manner using basic P2P functions and a query language that builds on those functions.

Our URI scheme nicely fits with the principles of file-sharing, as it does not assume a single static location for a document, but rather identifies a document by its schema and content.

Our graph data model could be compared with RDF: its main difference is that we do not see attributes as nodes in the graph: our model corresponds to an RDF graph showing only RDF links between resources.

Due to space restrictions, we left out a description of our case study that validates our approach, as well as a description of our approach for reusing queries, which makes use of a custom U-P2P Community of Queries.

References

1. Berners-Lee, T., Hendler, J.A., Lassila, O.: The semantic web. Scientific American 284, 34–43 (2001)
2. Martin Hepp, D.B., Siorpaes, K.: Ontowiki: Community-driven ontology engineering and ontology usage based on wikis. In: Proceedings of the 2005 International Symposium on Wikis, WikiSym 2005 (2005)
3. Halevy, A., Ives, Z., Madhavan, J., Mork, P., Suciu, D., Tatarinov, I.: The piazza peer data management system. IEEE Transactions on Knowledge and Data Engineering 16, 787–798 (2004)
4. Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.C., Simon, L.: SomeWhere in the semantic web. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 1–16. Springer, Heidelberg (2005)
5. Mukherjee, A., Esfandiari, B., Arthorne, N.: U-P2P: A peer-to-peer system for description and discovery of resource-sharing communities. In: ICDCSW 2002: Proceedings of the 22nd International Conference on Distributed Computing Systems, Washington, DC, USA, pp. 701–705. IEEE Computer Society, Los Alamitos (2002)
6. Arthorne, N., Esfandiari, B., Mukherjee, A.: U-P2P: A peer-to-peer framework for universal resource sharing and. In: Discovery, USENIX 2003 Annual Technical Conference, FREENIX Track, pp. 29–38 (2003)
7. Davoust, A., Esfandiari, B.: Towards semantically enhanced peer-to-peer filesharing. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 937–946. Springer, Heidelberg (2008)
8. Davoust, A., Esfandiari, B.: Towards semantically enhanced peer-to-peer filesharing. Journal of Software 4 (to appear, 2009)
9. Davoust, A., Esfandiari, B.: Peer-to-peer sharing and linking of social media based on a formal model of file-sharing. Technical Report SCE-09-04, Department of Systems and Computer Engineering, Carleton University (2009)