# Peer-to-peer Sharing and Linking of Social Media based on a Formal Model of File-Sharing

Alan Davoust Babak Esfandiari

March 19, 2009

**Abstract**

We propose a formal model for peer-to-peer file-sharing systems, which shows the specificities of such an infrastructure to store and exchange data. We show how an extension of the model enables the support of multiple file-sharing communities, with user-defined characteristics. Finally, we introduce in our model the notion of linked data in peer-to-peer file-sharing systems, to allow navigation between documents. The navigation and user-contributed content are illustrated with a case study of a community where movies and actor descriptions are created, shared and connected. We base our discussion on a proof-of-concept implementation in in our prototype framework U-P2P.

## 1 Introduction

Many different directions can be identified in the current explosion of social media: first, "Web 2.0" relies on the existing Web paradigm but allows users to contribute content, tags and comments. While technologies such as AJAX have simplified this process, the web is still ultimately client-server based, with centralized control over content.

Another direction is the Peer-to-peer paradigm, which addresses the issue of centralized control, in its various incarnations:

- Peer-to-peer file-sharing [1], [2] is very popular when it comes to social media, as it relies on user-produced content, evades centralized control and relies on file replication (as a result of downloading) to ensure high availability despite a high churn of peers;
- Distributed HashTables (DHTs) [3], [4], [5] focus instead on transparent access to documents;

- and Peer Data Management Systems [6] [7] propose a scalable alternative to centralized Data Integration, and rely on schema mappings and cooperation among peers.

Meanwhile, the Semantic Web is an orthogonal concept that proposes to enrich and annotate content in a manner that can allow machine processing and reasoning, improved data navigation and inference of new knowledge. One main challenge is the creation of such knowledge, which either requires human expertise or automatic extraction.

We are interested in bridging the gap among the above alternatives and technologies. In particular we propose a way for users to share documents and tags, and to collaboratively create knowledge, without relinquishing control to a single entity, and without needing expertise in ontology design, all in the familiar paradigm that is file-sharing.

In this paper, we make the following contributions:

- We first propose, in section 2, a formal model of existing P2P file-sharing systems. This will help highlight their specific properties (openness, data redundancy, etc.) that make them suitable for social media and thus distinguish them better from PDMS and DHTs. We also explicitly represent the joining and leaving of peers.

- We then extend in section 3 the above model to support multiple file-sharing communities in a single framework, with the particularity that the creation and dissemination of such communities will be entirely initiated by the peers in a totally distributed fashion, exactly in the same way that the documents are shared. Our U-P2P framework [8], [9] is an implementation of this model.

- Finally, in section 4 we further extend the above model to provide support for document navigation across different file-sharing communities. This is achieved through the introduction of a URI scheme well adapted to the distributed and replicated documents found in a P2P file-sharing system, and a form of link based on this URI scheme. In effect, this is an implementation of the idea of *linked data* [10] using peer-to-peer file-sharing concepts and properties.

A case study of a community of peers sharing movies and descriptions of movie actors is used as a running example to illustrate our definitions and new concepts. When appropriate, we have also indicated how we implemented each concept in U-P2P, but without getting into detailed implementation considerations that would detract from our main purpose. Design and implementation details were the focus of our earlier related contributions [8], [9], [11], as opposed to the strong theoretical angle brought by this work, which was addressed in a very limited way in [12].

## 2 Formal Model of a File-Sharing System

### 2.1 P2P File-Sharing

Let us first recall informally how peer-to-peer file-sharing typically works. The user publishes a document by simply making it available on his/her personal hard drive. In some systems such as Napster this is followed by advertizing the document.

Searching for a document is accomplished by filling out a form that proposes various metadata specific to the kind of file that is being shared by the application (e.g. in the case of a music-sharing application, artist name, song name, bit rate, etc.). The search reaches either all machines that are accessible through a given protocol such as Gnutella [2], or a central registry where the machines have advertized their resources. Results are presented as links to the resource as well as metadata that helps preview the content and choose the most appropriate document.

This is then followed by the downloading phase. The act of downloading results in a copy of the file in the user's hard drive and is now also made available there: in effect the file has been replicated. Popular files will therefore benefit from massive replication, whereas unpopular or corrupt ones won't. Ultimately, the popularity and the usefulness of a file decides of its high or low availability.

This is different from DHT systems, where the user is given a means to publish a document, but the actual location of the document is decided by the system. The act of retrieving a document does not necessarily result in its replication, although redundancy schemes have been proposed in the literature. In any case, such replication is not a function of the popularity of the files, as is the case in P2P file-sharing systems.

This is also different from Peer-based Data Management Systems (PDMS) such as Piazza, where the goal is to present a unified access to data distributed over heterogeneous databases. File-sharing systems can perhaps be viewed as providing a subset of the features of a PDMS, or could be built as applications on top of a PDMS.

## 2.2 Data Model

The data items of interest to us are files, which can be seen as 'blobs' of binary data, to which are attached a number of meta-data attributes. These meta-data attributes include at least the owner and permissions, file name, size, and modification date, and in a more specialized setting attributes such as the author of a text, or the bit rate of a music file.

In the formal definitions introduced in this work we will call our data items *documents*, and, by analogy with database theory, we will model each document as a tuple $(x_1, \ldots, x_n)$. A typical file will be represented by a (large) "binary" element and by several elements of standard data types (character strings, numbers) listing meta-data attributes of the file.

Formally, the distinction in between the "binary" data types and "standard" data types is that the former do not admit any simple operators to manipulate and compare elements, which exist for the latter (e.g. string matching operators and numerical comparison operators).

## 2.3 Peers

A P2P system $\mathcal{P}$ is a set of peers, where each peer $p$ has an identifier *id* (typically its IP address) and some storage capability (a file system or a database).

Peers connect, communicate with one another and manage data through file-sharing *communities*, which we define in the following.

## 2.4 Communities

**Definition 1** *A P2P file-sharing community $C$ is constituted by:*

- A set $\mathcal{G}$ of states, where each state is an undirected[1] graph $G = (X, E)$ where each vertex $v \in X$ is associated (through an injective[2] function $f_G$) with a peer $p$, and the edges in $E$ represent the connections among the peers.

[1]one could also consider a directed graph, the distinction is dependent on the underlying protocol and is immaterial to our purpose

[2]Two nodes of the graph represent different peers, but there may exist peers outside the community; to these peers the function $f_G^{-1}$ cannot be applied. The fact that peers may exists while not being part of a community in a given state is the reason why nodes of the graph - which may appear and disappear - must be distinguished from the peers themselves.

State *transitions* are defined in the pseudocode algorithm 1, and a graphical example showing the states and transitions is shown in figure 1.

**Algorithm 1** Procedures defining state transitions for community $C$

```
procedure JOIN($p$):                                ▷ a peer $p$ joins the community $C$
    $X \leftarrow X \cup \{v\}$
    $f_G(v) \leftarrow p$
end procedure

procedure CONNECT($p_1, p_2$):        ▷ peers $p_1$ and $p_2$ establish a connection within $C$
    $E \leftarrow E \cup \{(f_G^{-1}(p_1), f_G^{-1}(p_2))\}$
end procedure

procedure DISCONNECT($p_1, p_2$):              ▷ peers $p_1$ and $p_2$ remove their connection within $C$
    $E \leftarrow E \backslash \{(f_G^{-1}(p_1), f_G^{-1}(p_2))\}$
end procedure

procedure LEAVE($p$):                               ▷ a peer $p$ leaves the community $C$
    for all $p_i$ such that $(f_G^{-1}(p), f_G^{-1}(p_i)) \in E$ do
        DISCONNECT($p, p_i$)              ▷ $p$ disconnects from its neighbors in $C$
    end for
    $X \leftarrow X \backslash \{f_G^{-1}(p)\}$          ▷ Remove the vertex associated with $p$ in $C$
end procedure
```

- A protocol $Prot$, which is a function that, given a graph $G = (X, E)$ and a vertex $v \in X$, returns a set of vertices *accessible* to $v$. For example, the Gnutella protocol with a time-to-live (TTL) of $k$ returns the vertices reachable from $v$ by a path of a length $l \leq k$.

- A schema $S_C$, which is limited here to 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 rows of the table (i.e. the tuples in the extension of $R_C$) correspond to the documents shared in the community. We have the following constraints on $R_C$:

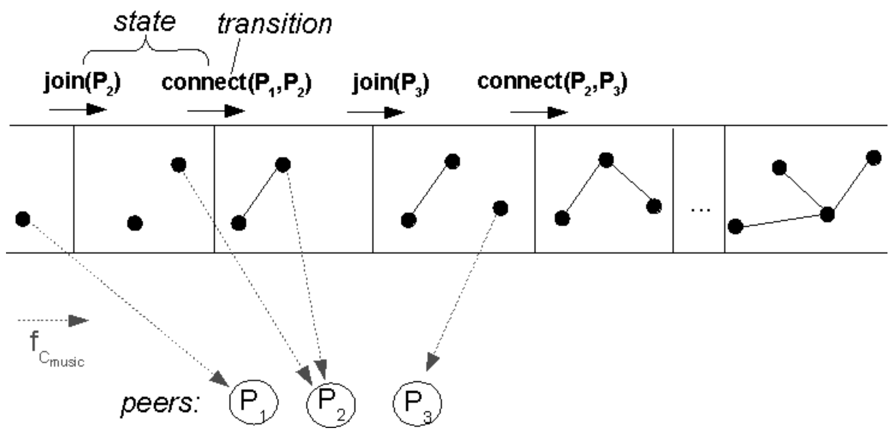

Figure 1: Example: states and transitions for example community $C_{movies}$

- $R_C$ has an attribute *name* that uniquely identifies[3] the document on peer $p$. We note that *name* will be the primary key of $R_C$.
- We define as *attachment* every attribute of $R_C$ that has a “binary” domain (in the sense discussed in 2.2), and *metadata*[4] is any attribute that is not an attachment.

- A *name*, which we will discuss in section 3.

Figure 2 illustrates this conceptual model of a peer-to-peer file-sharing community.

**Example** To illustrate this model, we can define the example community $C_{Movies}$, with the following properties:

- Name : “Movies”
- Schema: the relation $R_{movies}$, with the attributes *name, title, director, year*, which are metadata, and the additional attribute *trailer*, which is an attachment, and contains the binary video of the movie trailer. Table 2.4 shows an extension $R^{p_1}_{movie}$ stored by a peer $p_1$.

[3]Intuitively, if we consider that each element of the relation represents formally a file, then this attribute would be the filename, which is unique for a peer using a single directory to store all its shared data. We will discuss another option in section 2.7.

[4]Note that these *metadata* attributes have a *“meta-”* function with respect to the attachments, and do not refer to the schema of the relation $R_C$, which would be the metadata in the database sense.

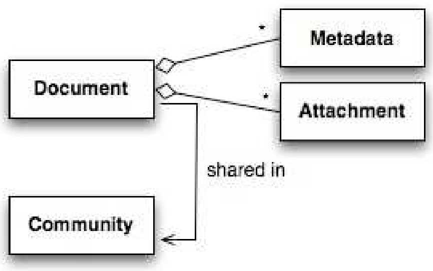

Figure 2: Conceptual model of a P2P file-sharing system

- Protocol: the Gnutella protocol.
- The set of states of the community is the set of graphs representing the network topology (i.e., the configuration of the peers connected within the community) as it evolves over time.

| **name** | **title** | **director** | **year** | **trailer** |
|---|---|---|---|---|
| h245 | "The Two Towers" | "Peter Jackson" | 2002 | [...] |
| h267 | "The Fellowship of the Ring" | "Peter Jackson" | 2001 | [...] |
| h321 | "Titanic" | "James Cameron" | 1997 | [...] |
| h145 | "Sleepy Hollow" | "Tim Burton" | 1999 | [...] |
| h234 | "Finding Neverland" | "Marc Forster" | 2004 | [...] |

Table 1: Example extension for the community "Movies". The attribute **trailer** is an attachment and we do not show its values, as they are binary.

## 2.5 Formal Definitions

**Definition 2** *Consider a community $C$ in state $G = (X, E) \in \mathcal{G}$. A peer $p$ is a* member of *community $C$ in that state $G$ iff:*

$$\exists v \in X \mid p = f_G(v)$$

**Definition 3** *We define a* document *of community $C$ as a tuple of the domain of $R$. See also section 2.2 for an informal discussion.*

**Definition 4** *The* content $\Omega(C,G)$ *of a community* $C$ *in state* $G=(X,E)$ *is a view obtained by the following relational query:*

$$\Omega(C,G) = \bigcup_{p\in f(X)} id(p) \times R_C^p$$

*where* $id(p)$ *is the identifier of* $p$ *and* $R_C^p$ *is the extension of* $R_C$ *over peer* $p$*. We note* peer-id *the first attribute of* $\Omega(C,G)$*.*

Intuitively, the content of a community is a view of all the documents stored by all the peers that are members of the community, with an additional column listing the identifier of the peer that hosts each document.

**Definition 5** *We define the following function* $prot_{C,G}$ *associated to the protocol* $Prot$ *of a community* $C$ *and to a state* $G$ *of* $C$*.*

$$prot_{C,G} : \begin{array}{ccc} \mathcal{G} & \longrightarrow & \mathscr{P}(\mathcal{P}) \\ p & \longmapsto & f_c(Prot(G, f_c^{-1}(p))) \end{array}$$

Intuitively, $prot_{C,G}$ returns the set of peers reachable from a peer $p$ in the state $G$ of community $C$, using the protocol $Prot$ of $C$.

**Definition 6** *We define the following operations available to a peer* $p$ *in a peer-to-peer system* $\mathcal{P}$*: PUBLISH(), SEARCH(expr), DELETE(document), and DOWNLOAD(*$j, name_0$*). The specifications of these procedures and functions are given in Algorithm 2.*

## 2.6 Formal Semantics of Queries

The previous definitions of the PUBLISH(), SEARCH(*expr*), DELETE(*document*), and DOWNLOAD($j, name_0$) operations are based on a set-theoretic view and relational algebra.

In this section we view SEARCH(*expr*) instead as a first-order logic query, and formalize the semantics of this query with respect to the predicate $R_C$.

We note $P_{CG}$ the set of peers of $\mathcal{P}$ that are *member-of* $C$ in state $G$.

We consider the set $\{I_p\}_{p\in P_{CG}}$ of interpretations for the predicate $R_C$ defined by the extensions $R_C^p$ for all the peers $p$ in $P_{CG}$.

**Definition 7** *We define the modal system* $M = (P_{CG}, prot_{C,G}, \phi)$ *where each peer in* $P_{CG}$ *defines a world,* $prot_{C,G}$ *defines an accessibility relation in between worlds, in the sense that* $(p_1,p_2)$ *is in the relation iff* $p_2 \in prot_{C,G}(p_1)$*, and* $\phi$ *is a valuation function that for a given world* $p$ *returns the set of tuples that are true in the interpretation* $I_p$*, i.e. we have:* $\phi : p \mapsto R_C^p$*.*

**Algorithm 2** Data manipulation operations available to a peer $p$ in a P2P system $\mathcal{P}$. (Note: $\sigma$ and $\pi$ refer to the relational SELECT and PROJECT operations)

```
  procedure PUBLISH($document$):
      $R_C^p \leftarrow R_C^p \cup \{document\}$
  end procedure

  procedure DELETE($document$):
      $R_C^p \leftarrow R_C^p \backslash \{document\}$
  end procedure

  function SEARCH($expr$):
      $Result \leftarrow \sigma_{(peerid \in prot_G(p))}(\Omega(C, G))$    ▷ Select content from reachable peers
      $Result \leftarrow \sigma_{expr}(Result)$    ▷ Filter the documents using $expr$
      $Result \leftarrow \pi_{(peerid, metadata(R_C))}(Result)$    ▷ Remove attachments
      return $Result$.
  end function
```

Here *expr* is an open formula involving conjunctions, disjunctions, and built-in predicates (e.g. $=$, $\leq$, application-specific string matching operators based on regular expressions). *expr* may have up to $k$ free variables, where $k$ is the number of meta-data attributes in $R_C$. The different queries accepted by a community are the different possible formulas *expr* that can be expressed using the built-in predicates of a P2P application.

```
  function DOWNLOAD($id, name_0$):
Require: $id \in prot_G(p)$
      $\{doc\} := \sigma_{(peerid=id \wedge name=name_0)}(\Omega(C, G))))$    ▷ This SELECT should return a single document
      PUBLISH($doc$)    ▷ Add document to local extension for the community.
      return $doc$.
  end function
```

**Definition 8** *The modal operator* **V** *(for "visible") is then defined as:*
$< M, p > \models \mathbf{V} R_C(x)$ *iff* $\exists p_1 \in P_{CG}, p_1 \in prot_{C,G}(p) \wedge x \in \phi(p_1)$.

In the following we will respectively note $m_1, \ldots, m_k$ (resp.$a_1, \ldots, a_j$) the indices of the meta-data (resp. attachments) attributes of $R_C$. In other words, if the attributes of $R_C$ are numbered as they appear in the predicate $R_C$, then $a_2$ denotes the second position of an attachment in $R_C$, etc. We have $j + k = n$ where $n$ is the arity of $R_C$.

A query SEARCH(*expr*) can be written in a declarative form as follows:
$\{(x_{m_1}, \ldots, x_{m_k}) \mid \exists (x_{a_1}, \ldots, x_{a_j}), R_C(x_1, \ldots, x_n) \wedge expr(x_{m_1}, \ldots, x_{m_k})\}$

We then have the following semantics for a search query, which follows immediately from the definitions of the modal system and from the relational algebra definition of the query (given in Algorithm 2):

**Theorem 1** *A tuple* $(y_{m_1}, \ldots, y_{m_k})$ *is an answer to the query SEARCH(expr) (applied to peer p in the community C in state G) iff:*
$expr(y_{m_1}, \ldots, y_{m_k}) \wedge (\exists (y_{a_1}, \ldots, y_{a_j}), < M, p > \models \mathbf{V} R_C(y_1, \ldots, y_n))$

Similarly, the semantics of the DOWNLOAD($j, name_0$) function can be given by the following pre- and post-conditions:
**pre-condition:** $j \in prot_{C,G}(p) \wedge R_C(name, \ldots, x_n) \in \phi(j)$
**post-condition:** $R_C(name, \ldots, x_n) \in \phi(p)$

## 2.7 Unique Names

In file-sharing systems, a tricky issue is to identify multiple copies of the same file, and, conversely, to distinguish files which appear to be the same, typically for having the same name. This is another view of the classic problem of maintaining consistency between several redundant sources of data.

In the file-sharing setting, it would be desirable to identify when several peers offer the exact same content, that has been downloaded from one to the other. This problem can be solved by generating for each document a one-way hash of the attribute values, and using this hash as a unique identifier of the file. We have implemented this in our prototype U-P2P, using the MD5 [13] algorithm.

We can then introduce the following assumption:

**Assumption 1** *The integrity constraint that the attribute* name *is a primary key for* $R_C$ *holds across all peers.*

This integrity constraint is a key foundation for the extensions to the model that we present in the following sections. In the following discussion we will thus assume that this constraint holds.

We note that this assumption allows us to simplify the formal semantics for the DOWNLOAD$(j, name_0)$ function, presented in the previous section, as follows:

**pre-condition:** $< M, p > \models \mathbf{V} R_C(name, \ldots, x_n)$

**post-condition:** $R_C(name, \ldots, x_n) \in \phi(p)$, or, equivalently: $I_p \models R_C(name, \ldots, x_n)$

## 3 Multiple communities

### 3.1 Motivation

We now extend our model to enable the creation of multiple communities within a single application. As per our formalization, defining a new community can be achieved by creating a new schema and choosing a protocol. The initial state of the community is an empty graph, and the following sequence of states will follow from the sequence of transitions (JOIN(), CONNECT(), etc.).

We consider the following requirements:

- Any peer should be able to create a new community,
- the creator should be able to advertize the existence of the community to peers which are not members of the community,
- conversely, any peer should be able to discover the new community.

### 3.2 The community of communities

For this purpose, we define a specific bootstrap community called the *Community* community, which we will note $C_{Community}$. This community is defined as follows:

- The protocol of $C_{Community}$ is $Prot_0$, which we arbitrarily choose (in our prototype U-P2P) to be the Gnutella protocol.
- The schema of $C_{Community}$ consists of a single relation $R_{Community}(.,.,.)$ of arity 3, with the following attributes: *name, schema , protocol.*

  The subdomain of for each attribute is as follows:

- *name* is the mandatory meta-data attribute defined in section 2.4 for any community schema, and its domain is a finite set of possible identifiers for communities[5].
- The domain of *schema* is a set of possible schemas for communities, described in some binary (or textual) representation. Each element of this set is a schema compliant with the definitions of section 2.4.
- The domain of *protocol* is a set of network protocols, described in some binary representation. Each element of this set is a protocol compliant with the definitions of section 2.4.

The domain for each attribute of $R_{Community}$ reflects the fact that each document shared in $C_{Community}$ will itself represent a community. In a way, $C_{Community}$is a directory of all existing communities, implemented itself as a community.

It is interesting to note that a consequence of this definition is that $C_{Community}$ (i.e. the *Community* community itself) must be represented as a document $doc_{Community}$ in this $C_{Community}$. $doc_{Community}$ has the following values for the attributes of $R_{Community}$:

- *name*: "Community"[6]
- *schema*: a representation of the schema described above.
- *protocol*: a binary representation of the Gnutella protocol.

A requirement to bootstrap the system is that each U-P2P peer $p$ is *member-of* $C_{Community}$. This is what allows peers to discover existing communities (using SEARCH() and DOWNLOAD() in $C_{Community}$).

In U-P2P, necessarily, we have :

- $\forall p \in \mathcal{P}, p$ *member-of* $C_{Community}$.
- $\forall p, doc_{Community} \in R^{p}_{Community}$.

[5]conceptually, this list of identifiers could be infinite denumerable, but in practice we are considering alphanumeric strings which will be used by peers, which are computers, and cannot handle identifiers of arbitrary length. So in practice this list is finite.

[6]In order to comply with assumption 1 we would in practice have a unique name generated by a hash function, but we prefer to use here such user-friendly names for better readability of our examples.

For any community, once the community document (i.e. the document describing the community, shared in $C_{Community}$) is downloaded, the interpretation of its contents by U-P2P allows the peer to join the community. This additional feature specific to $C_{Community}$ is reflected on the definition of the DOWNLOAD() function, as shown in algorithm 3.

**Algorithm 3** Peer $p$ downloads from peer $id$ a community $C_{name_0}$ represented by $doc_0$ in $C_{Community}$

```
  function DOWNLOAD($id, name_0$): First the document $doc_0$ with name
  $name_0$ is downloaded as any other document would:
Require: $id \in prot_{C_{Community},G}(p)$
      $\{doc_0\} := \sigma_{(peerid=id \wedge name=name_0)}(\Omega(C_{Community}, G))))$
      PUBLISH($doc_0$)
      JOIN($id, C_{name_0}$)   ▷ The downloaded document provides a handle for
  the peer to automatically join the community.
      return $doc$.
  end function
```

## 3.3 From documents to communities

This feature - the possibility of automatically joining new communities by simple DOWNLOAD() operations - relies on the fact that a community schema and a protocol can be represented in a binary format which can be interpreted by the application.

In U-P2P, schemas are represented as XSD (XML schema definition) files, and protocols are represented as java classes which implement a client for the protocol, and can be automatically used by U-P2P. We note that a proof-of-concept study for this representation of protocols was presented in [8], but in the most recent versions of U-P2P, this feature has been abandoned for simplicity's sake: all implemented communities are required to use the Gnutella protocol, which is now hard-coded into the application.

In addition to these fundamental attributes, we have extended the *Community* community in U-P2P to include additional meta-data attributes to improve the expressiveness of searches(e.g. a user-friendly description, keywords, etc.), and additional attachments which allow to customize the rendering of documents of the community in a web browser. We do not discuss these additional attributes in this formal model in order not to dilute the discussion, but it is important to see that the model has enough flexibility to accomodate a more complex schema than the one detailed here.

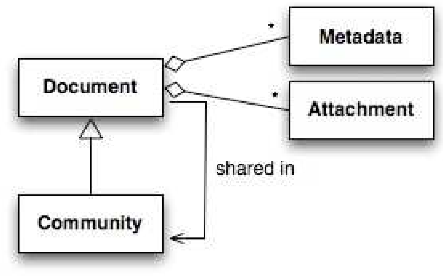

Figure 3: Conceptual meta-model of U-P2P. The inheritance relation expresses the fact that communities can be created and downloaded just as any other document.

### 3.4 Example

We present here a simple example with three peers and three communities. Using the examples shown in table 2.4 (we remind the reader that table 2.4 shows an extension for the community $C_{Movie}$ stored in the peer $p_1$ (in this example $p_1$ is the only member of $C_{Movie}$)) and table 3.4, which shows the *content* (cf. definition 4) of communities $C_{Community}$ and $C_{Actor}$.

The documents physically hosted by each peer would then be as shown in Figure 4.

## 4 Bridges

### 4.1 Motivation

Linked Data [10] aims to provide support for referencing data in a stable way (through the use of Uniform Resource Identifiers (URI)[14]) and connecting them using labelled connections. This should allow data to be more easily navigable. The use of labels, especially those with clear semantics, between resources also opens the door for knowledge representation that is consumable by software agents, for reasoning and inference purposes.

A central feature of the World Wide Web is the concept of *hyperlink*, which allows users to navigate from one document to the next. In a relational database, semantic links between data of two separate tables are materialized by *foreign key* constraints. Such constraints are the necessary basis to interpret complex queries across several tables.

Content of $C_{Community}$:

| **Peer-id** | **Name** | **Schema** | **Protocol** |
|---|---|---|---|
| 1 | Community | $[S_{Community}]$ | [Gnutella] |
| 2 | Community | $[S_{Community}]$ | [Gnutella] |
| 3 | Community | $[S_{Community}]$ | [Gnutella] |
| 1 | Movie | $[S_{Movie}]$ | [Gnutella] |
| 2 | Actor | $[S_{Actor}]$ | [Gnutella] |
| 3 | Actor | $[S_{Actor}]$ | [Gnutella] |

Content of community $C_{Actor}$:

| **Peer-id** | **Name** | **First Name** | **Last Name** | **Birth date** |
|---|---|---|---|---|
| 2 | h534 | "Johnny" | "Depp" | 09-06-1963 |
| 3 | h534 | "Johnny" | "Depp" | 09-06-1963 |
| 3 | h845 | "Elijah" | "Wood" | 28-01-1981 |
| 3 | h687 | "Kate" | "Winslet" | 04-12-1975 |
| 2 | h855 | "Liv" | "Tyler" | 10-07-1977 |
| 3 | h855 | "Liv" | "Tyler" | 10-07-1977 |

Table 2: Example content for the communities "Community" and "Actor".

Current peer-to-peer file sharing systems do not offer a natural way of linking files or data items to one another, and users normally extend searches by entering new queries using different keywords to the system's search interface.

We propose to further extend our model with a form of link based on the integrity constraint induced by our assumption 1. This assumption in effect guarantees that a single meta-data field (the *name*) can be used to uniquely identify a document, or any copy of this document that may have been replicated (without modification) across the P2P network.

We propose to introduce a new type of attribute, which will serve as links. As documents holding such links will make navigation possible, we will call them *bridges*, and the link attributes will be called *bridge endpoint attributes* (or simply *endpoints*).

## 4.2 URIs and Endpoints

As discussed by T.Berners-Lee on the topic of Linked Data [10], linking any type of resources requires a naming scheme to identify these resources,

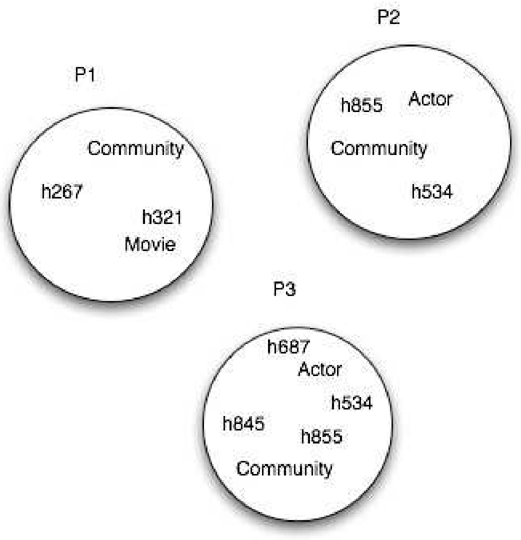

Figure 4: Distribution of documents in an example P2P system with three peers and three communities.

such as a URI [14]. 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*, 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`.

Formally, a bridge endpoint attribute can be 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 9** *An attribute* att *is a* bridge endpoint attribute *iff its domain* *is C-NAMES × 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.

## 4.3 Bridge Communities

Many schemas containing *bridge endpoint attributes* can be defined, and many communities can use these schemas. The class of such communities can be defined as follows:

**Definition 10** *A community B is a* bridge-type *iff the schema of B includes at least one* bridge endpoint attribute.

We call any document shared in a bridge-type community a *bridge*. Intuitively, a bridge is a navigable document.

Our model extended to include Bridges is illustrated in Figure 5.

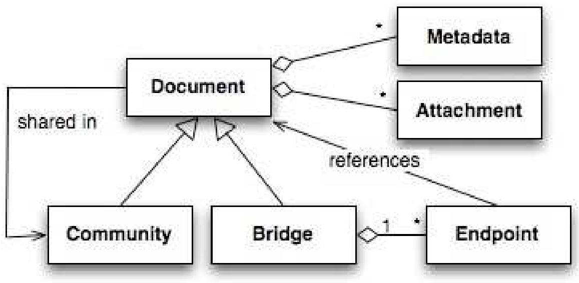

Figure 5: Bridges in the U-P2P Meta-Model.

This broad definition allows for any type of document to hold links towards other documents, and a navigation model very similar to that of the WWW, a browsable graph of documents with one-way links. However, properties of unstructured P2P file-sharing networks, where users produce and download content in a completely decentralized manner, *à la* Web 2.0, and arbitrarily replicate documents across the network, suggest a different approach. A simple use-case scenario could be as follows:

Suppose a user finds an error in a document downloaded from U-P2P and wishes to correct that error for the benefit of the other users. Making changes to that particular document and publishing it won't have much of an effect: one would need to make changes to all copies of the document. One solution could be to create an updated version and link it via a bridge labeled `owl:priorVersion`[7] to the older version. However, using a navigation model of one-way bridges, only users who find the new version can navigate to the older, possibly obsolete version, which seems quite useless.

Similarly, a review posted and linked to an existing film would be linked "backwards", since most users would typically need to navigate from a movie to its reviews rather than the opposite. Modifying a single film document to add the link towards the review would not help very much, since it still would leave numerous copies of the film unrelated to the review; furthermore the modified document would require a new *name* to satisfy assumption 1.

Any user should be able to publish bridges, and link documents stored by any peers. These simple examples show that there is a strong case for decoupling bridges from the documents that they link, and secondly to offer two-way navigation of the bridges.

This link-centric approach follows the model of RDF, where an RDF link can be added to any existing graph, and navigated in both directions through queries.

With such considerations in mind, we propose a first approach where we consider only the simplest bridges, which model binary relations between two entities, themselves modeled by existing documents.

All bridges can then be shared in a single Bridge Community, which we define as follows:

- the name of the community is "BinaryBridge"
- the schema of the community consists of a single relation $R_{BinaryBridge}$, of arity 4, with the following attributes :
  - name : the name of the bridge
  - endpoint1 : the URI (as defined above) of a document $d_1$ in community $C_1$, e.g. `up2p:fa8bcc5650`[...]`/ca94aff50`[...].
  - endpoint2 : the URI of a document $d_2$ in community $C_2$.
  - label : a label for the bridge.
- the protocol of the community is the Gnutella protocol.

[7]Using the OWL namespace does not immediately imply that the system is capable of interpreting the semantics of the label. It does, however, suggest a direction for future development.

**Example** Using this simple model, we extend the example presented in section 3.4, and we introduce the additional community $C_{BinaryBridge}$.

We assume that peer $p_3$ has joined the community $C_{Movie}$, and downloaded the movies “The Fellowship of the Ring” and “The Two Towers”. The same peer $p_3$, who is a movie fan, then creates the community $C_{BinaryBridge}$ to improve the navigation between movies and actors. He/She then creates two bridges, one to indicate that the “The Two Towers” is the sequel of “The Fellowship of the Ring”, and another to relate actors to the movies they have played in.

Table 4.3 shows the extensions for communities $C_{Community}$, $C_{Movie}$, $C_{BinaryBridge}$, and $C_{Actor}$ stored by $p_3$.

As we have mentioned earlier, this type of simple bridges can be used to model RDF links. If we represent these links between movie and actor documents as a graph, we obtain the graph shown in Figure 6.

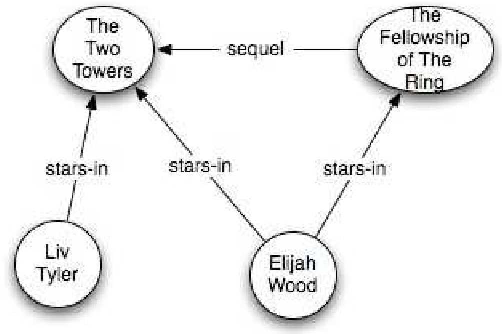

Figure 6: Graph showing the bridges between documents stored by $p_3$.

## 4.4 Document Navigation using Simple Bridges

With arbitrary bridges, URIs such as `up2p:<CommunityName>/<DocumentName>` allow for simple browsing (one-way navigation): users can follow a link by searching the community `<CommunityName>` for the document `<DocumentName>`, then downloading that document if it is available, and repeating the process from the new document.

However, as we have pointed out in section 4.3, a preferable approach would be two-way navigation, which is more easily permitted, and even automated, by the simple model considered in the previous discussion and example.

Assuming the existence of community BinaryBridge, we introduce a function NAVIGATE defined by algorithm 4. The purpose of this function is to perform a search of all documents linked by simple BinaryBridges to a given document identified by its URI $uri_0$, such that the bridge label matches a given input $bridgeLabel$.

**Algorithm 4** Navigation using simple bridges

```
function NAVIGATE($uri_0, bridgeLabel$):
    $btable \leftarrow$ BinaryBridges.SEARCH($Endpoint1 = uri_0 \wedge match(label, bridgeLabel)$)
    $btable \leftarrow$ BinaryBridges.SEARCH($Endpoint2 = uri_0 \wedge match(label = bridgeLabel)$)    ▷ Perform search on both endpoints of bridges
    $btable$.RemoveDuplicates($name$)  ▷ Only keep one instance of each document
    for all ($pid, name, label, endpoint1, endpoint2$) in $btable$ do
        BinaryBridges.DOWNLOAD($pid, name$)  ▷ Download all the bridges
    end for
    for all ($pid, name, label, endpoint1, endpoint2$) in $btable$ do
        if $endpoint1 = uri_0$ then
            $otherend \leftarrow endpoint2$
        else
            $otherend \leftarrow endpoint1$
        end if
        cname $\leftarrow$ GetCommunity($otherend$)  ▷ Extract the community name from the URI $otherend$.
        docname $\leftarrow$ GetDocumentName($otherend$)  ▷ Extract the document name from the URI $otherend$
        if $p$ member-of $C_{cname}$ then
            $results \leftarrow results \cup C_{cname}$.SEARCH($name = docname$)
        end if
    end for
    return $results$
end function
```

Here we assume that the peer originating the query ignores Bridges to documents in new communities (i.e. that the peer is not *member-of*). We could extend this algorithm to include such Bridges, by including a step where Bridges linking to documents in new communities would trigger the automatic search and download of this community from $C_{Community}$.

**Example** Starting from the previous example, we now assume that peer $p_2$ has downloaded both the "Movie" and the "BinaryBridge" communities. $p_2$ then calls the function NAVIGATE with inputs `up2p:Actor/h845` (the URI of the document describing Elijah Wood) and `*`, which we assume will be a wildcard to match any label.

The NAVIGATE function will retrieve both bridges `up2p:/BinaryBridge/h678` and `up2p:/BinaryBridge/h457`, then follow them to obtain the table of search results shown in Table 4.

Peer $p2$ could then repeat the search originating from the movie "The Fellowship of the Ring" (`up2p:/Movie/h267`), and find, for instance, that Liv Tyler plays in the same film, and that the film has a sequel called "The Two Towers".

This shows how labeled links can be found and followed in order to navigate the greater graph of documents distributed across peers and communities.

As we have previously pointed out, BinaryBridge documents can be mapped to RDF Links, and we give here a more complete mapping, in table 5.

Based on this analogy, we could consider automating the navigation process in order to obtain a query engine comparable to a SPARQL engine.

If wildcard matching is used for both inputs of the function NAVIGATE, then this function can be used to emulate basic searches for RDF triples, i.e. searches using templates of the form `(?s, pred, obj)`, or `(ubjs, ?p, obj)`.

We can then map such atomic queries to the functions NAVIGATE, or SEARCH, following the basic dichotomy of RDF triples using literals, and RDF links. From there it seems a reasonable step to consider plugging an existing SPARQL engine into U-P2P. Our purpose here is not to discuss the details of this step, but rather to show how our simple model of Bridges could allow us to support expressive queries over a graph of linked documents, distributed in a P2P File-Sharing network.

## 4.5 A Community of Bridge-Types

In the simple representation proposed in the previous section, the "label" associated to a bridge is used to *type* the link represented by the bridge, but as such it is simply a character string, that is up to the user to interpret.

A richer way of describing bridges is to partition the abstract notion of Bridges, or navigable documents, into classes of Bridges representing the same abstract relation in between documents. Each such abstract relation is a *bridge-type* (cf. definition 10) that can be modeled by a community, and the document describing one such community, in addition to the standard attribute set (name, protocol, schema), should list the properties of the abstract relation.

These *bridge-type* communities form a subclass of the greater class of all communities, and they should be shared in another community materializing this subclass[8], the community $C_{BridgeType}$.

An important property of a bridge-type is the specification of which communities the documents linked by the bridges may belong to. With respect to the abstract relation, these communities represent a similar concept to the *domain* and *range* of an RDF Property.

For each endpoint attribute of a bridge-type, we can thus define a *Range Community*, and by extension consider that the *Range Communities* of all Endpoint attributes of a bridge-type are the *Range Communities* of that bridge-type itself.

Formally, we define the Range Community of an attribute as follows:

**Definition 11** *For each Bridge Endpoint Attribute* att*: there exists a community $C_D$ such that the subdomain of* att *is equal to the subdomain of the attribute* name *of $R_{C_D}$. $C_D$ is then called the* range community *of* att*.*

We now define the community $C_{BridgeType}$:

- name: "BridgeType"
- protocol: Gnutella.
- schema: As an extension of $C_{Community}$, $C_{BridgeType}$ must have at least the same attributes, plus some additional attributes:
  - *name*, *protocol* and *schema*
  - *label*, a user-friendly label associated with any Bridge in this community, i.e. with the abstract relation that the bridge-type models.
  - *endpoints*, a complex attribute listing the endpoint attributes of the bridge-type, and for each endpoint, its range community[9].

[8]By analogy with the RDF model presented in table 5, we could somehow model the *subclass* relation between $C_{BridgeType}$ and $C_{Community}$. We will set this problem aside for the moment.

[9]Such a complex attribute does not necessarily contradict our first definition of a community schema (a single n-ary relation), as long as this attribute can be an attachment, that the implementing system (UP2P) can be programmed to handle, just as it can handle protocols and schemas encoded in attachments.

**Example** We extend our running example of Movies and Actors, introducing two Bridge-Types.

The graph of documents represented in Figure 6 can be modeled using two bridge-types, associated to the abstract relations of "Sequel" and "Stars-In". The documents stored by $p_3$ in this model would then be as shown in Table 4.5.

### 4.6 Navigation using the BridgeType community

The BridgeType community introduced in the previous section is a way to easily find bridge-types, and through their *Range Communities*, to immediately identify which bridge-types may apply to our starting point community.

We define the following navigation function associated with the BridgeType community: NAVIGATEBETTER, with inputs $(uri_0, bridgeLabel)$ similar those of NAVIGATE, except that *bridgeLabel* should now match the *label* associated with a bridge-type rather than an individual bridge. NAVIGATEBETTER is specified in algorithm 5.

Note that in our open model where any user can create communities, we could imagine both the simple BinaryBridge bridges and more complex bridge-types supported by the BridgeType community to coexist in a UP2P network.

## 5 Conclusion

The central contribution of this paper is a new formal model for existing P2P file-sharing systems, rooted in a database-theoretic view of data items.

This general model provides a basis to analyse how data can be exchanged among peers who join and leave communities, may lose their connection, and have full control over their local system, as is the case in the highly dynamic and unstructured networks which account for most of P2P activity.

Furthermore, by "reifying" the notion of a *community*, we have created a highly flexible model where new concepts, new content can be easily created by ordinary end-users of our file-sharing system, which is very much in line with the fundamental shift towards user input associated with social media such as blogs and wikis.

In addition, we have shown that our multiple-community structure can be the basis of a new URI scheme to address data in a P2P file-sharing network, which tends to move, appear and disappear at great speed. This

**Algorithm 5** Navigation using the BridgeType community

```
function NavigateBetter($uri_0, bLabel$):
    $comm_0$ ←GetCommunity($uri_0$)    ▷ Get Starting community and document
    $doc_0$ ←GetDocumentName($uri_0$)
    $btable$ ← BridgeType.search($hasRange(endpoints, comm_0) \wedge match(label, bLabel)$)
    $btable$.RemoveDuplicates($name$)    ▷ Only keep one instance of each BridgeType
    $results$ ← [emptyTable]    ▷ Start with blank list of results
    for all $(pid, name, label)$ in $btable$ do    ▷ The other attributes of BridgeType are attachments
      ▷ Download each relevant bridge-type, which is also a community and will be automatically joined:
        BT ← BridgeType.download($pid, name$)
      ▷ Get the list of endpoints that have $comm_0$ for Range Community...
        $allRelevantEndpoints$ ← BT.GetEndpoint($range = comm_0$)
          ▷ ...and the other endpoints, which will link to other documents:
        $allOtherEndpoints$ ← BT.GetEndpoint($range \neq comm_0$)
       ▷ select any bridge that links to the original document, through any relevant endpoint:
        $bridges$ ←BT.search($\bigvee_{ep \in allRelevantEndpoints}(ep = uri_0)$)
        for all $(pid, bname, \dots) \in bridges$ do
            $br$ ← BT.download(pid, bname)
              ▷ Get all documents linked by other endpoints of the bridge:
            for all $ep \in allOtherEndpoints$ do
                $comm_2$ ← BT.GetCommunity($br.ep$)
                $doc_2$ ← BT.GetDocumentName($br.ep$)
                if $p$ member-of $comm_2$ then
                    $results \leftarrow results \cup comm_2$.search($name = doc_2$)    ▷ Collect all documents as search results
                end if
            end for
        end for
    end for
    return $results$
end function
```

URI scheme opens the possibility of linking data in the file-sharing context, and of navigating the resulting graph of connected documents. Users can continue creating new connections and new documents, and this graph can therefore *emerge* from the participation of users, and evolve with the popularity of what is being shared.

Our future work will concentrate on the addition of semantics to our bridges. Simply associating to a bridge-type an indication of whether the relation it models is transitive, reflexive, or symmetric, can allow a simple navigation to be extended by recursive processing, possibly generating new knowledge.

This would also enable the consumption of the graph of documents by software agents for reasoning purposes. We would then in effect be evolving toward a Semantic Web-type overlay on a P2P file-sharing platform, which might be a better deployment for the vision of a writable and machine-consumable web, as envisioned by Tim Berners-Lee: peers contribute low-granularity nuggets of knowledge and the emerging knowledge evolves with the popularity of the concepts that compose it.

## References

[1] Napster. http://www.napster.com/ (2000). URL http://www.napster.com/

[2] Gnutella: The gnutella protocol specification 0.6. http://rfc-gnutella.sourceforge.net (2002). URL http://rfc-gnutella. sourceforge.net

[3] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, vol. 31, pp. 161–172. ACM Press (2001). DOI http://dx.doi.org/10.1145/383059.383072. URL http://dx.doi.org/10.1145/383059.383072

[4] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. Networking, IEEE/ACM Transactions on **11**(1), 17–32 (2003). DOI http://dx.doi.org/10.1109/TNET.2002.808407. URL http://dx.doi.org/10.1109/TNET.2002.808407

[5] Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proc. of the IFIP/ACM Int. Conf. on Distributed Systems Platforms Heidelberg, pp. 329–350. Springer-Verlag (2001)

[6] The piazza peer data management system. IEEE Trans. on Knowl. and Data Eng. **16**(7), 787–798 (2004). DOI http://dx.doi.org/10.1109/TKDE.2004.1318562

[7] Adjiman, P., Chatalic, P., Goasdou?, F., Rousset, M.C., Simon, L.: SomeWhere in the Semantic Web . In: International Workshop on Principles and Practice of Semantic Web Reasoning (2005)

[8] Mukherjee, A., Esfandiari, B., Arthorne, N.: U-p2p: A peer-to-peer system for description and discovery of resource-sharing communities. In: ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems, pp. 701–705. IEEE Computer Society, Washington, DC, USA (2002)

[9] 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)

[10] Berners-Lee, T.: Linked data. World Wide Web design issues (2006). URL http://www.w3.org/DesignIssues/LinkedData.html

[11] Arthorne, N., Esfandiari, B.: Peer-to-peer data integration with distributed bridges. In: CASCON '06: Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research, p. 14. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1188966.1188985

[12] Davoust, A., Esfandiari, B.: Towards semantically enhanced peer-to-peer file-sharing. In: R. Meersman, Z. Tari, P. Herrero (eds.) OTM Workshops, *Lecture Notes in Computer Science*, vol. 5333, pp. 937–946. Springer (2008)

[13] Rivest, R.: The MD5 Message-Digest algorithm. RFC 1321, Internet Engineering Task Force (1992). URL http://www.rfc-editor.org/rfc/rfc1321.txt

[14] Berners-Lee, T., Fielding, R., Masinter, L.: Uniform resource identifier (URI): generic syntax. RFC 3986, Internet Engineering Task Force (2005). URL http://www.rfc-editor.org/rfc/rfc3986.txt

$R^{p_3}_{C_{Community}}$:

| **Name** | **Schema** | **Protocol** |
|---|---|---|
| Community | $[S_{Community}]$ | [Gnutella] |
| Movie | $[S_{Movie}]$ | [Gnutella] |
| Actor | $[S_{Actor}]$ | [Gnutella] |
| BinaryBridge | $[S_{BinaryBridge}]$ | [Gnutella] |

$R^{p_3}_{C_{Actor}}$:

| **Name** | **First Name** | **Last Name** | **Birth date** |
|---|---|---|---|
| h534 | “Johnny” | “Depp” | 09-06-1963 |
| h845 | “Elijah” | “Wood” | 28-01-1981 |
| h687 | “Kate” | “Winslet” | 04-12-1975 |
| h855 | “Liv” | “Tyler” | 10-07-1977 |

$R^{p_3}_{C_{Movie}}$:

| **name** | **title** | **director** | **year** | **trailer** |
|---|---|---|---|---|
| h245 | “The Two Towers” | “Peter Jackson” | 2002 | [...] |
| h267 | “The Fellowship of the Ring” | “Peter Jackson” | 2001 | [...] |

$R^{p_3}_{C_{BinaryBridge}}$:

| **Name** | **Label** | **Endpoint1** | **Endpoint2** |
|---|---|---|---|
| h678 | “sequel” | up2p:Movie/h245 | up2p:Movie/h267 |
| h678 | “stars in” | up2p:Actor/h845 | up2p:Movie/h245 |
| h457 | “stars in” | up2p:Actor/h845 | up2p:Movie/h267 |
| h923 | “stars in” | up2p:Actor/h855 | up2p:Movie/h267 |

Table 3: Documents stored by $p_3$, including simple Bridges.

| | | | | |
|---|---|---|---|---|
| 1 | h245 | “The Two Towers” | “Peter Jackson” | 2002 |
| 3 | h245 | “The Two Towers” | “Peter Jackson” | 2002 |
| 1 | h267 | “The Fellowship of the Ring” | “Peter Jackson” | 2001 |
| 3 | h267 | “The Fellowship of the Ring” | “Peter Jackson” | 2001 |

Table 4: Example: Results obtained by NAVIGATE(‘up2p:Actor/h845’, ‘*’)

| **RDF concept** | **UP2P data model** |
|---|---|
| RDF Resource | document |
| RDFS Class | Community |
| RDF Link (statement where subject and object are resources) | BinaryBridge |
| RDF Property (of RDF link) | BinaryBridge.label |
| RDF Property (of triple linking a resource to a literal) | metadata attribute |

Table 5: Correspondence between the U-P2P metamodel and RDF(S)

$R^{p3}_{C_{Community}}$:

| **Name** | **Schema** | **Protocol** |
|---|---|---|
| Community | $[S_{Community}]$ | [Gnutella] |
| Movie | $[S_{Movie}]$ | [Gnutella] |
| Actor | $[S_{Actor}]$ | [Gnutella] |
| BridgeType | $[S_{BinaryBridge}]$ | [Gnutella] |

$R^{p3}_{C_{BridgeType}}$:

| **Name** | **Schema** | **Protocol** | **{Endpoint/Range}** |
|---|---|---|---|
| StarsIn | $[S_{StarsIn}]$ | [Gnutella] | {'actor'/Actors, 'movie'/Movies} |
| Sequel | $[S_{Sequel}]$ | [Gnutella] | {'prequel'/Actors, 'sequel'/Movies} |

$R^{p3}_{C_{StarsIn}}$:

| **Name** | **actor** | **movie** | **Role** |
|---|---|---|---|
| h678 | up2p:Actor/h845 | up2p:Movie/h245 | "Frodo" |
| h457 | up2p:Actor/h845 | up2p:Movie/h267 | "Frodo Baggins" |
| h923 | up2p:Actor/h855 | up2p:Movie/h267 | "Arwen Evenstar" |

$R^{p3}_{C_{Sequel}}$:

| **Name** | **prequel** | **sequel** |
|---|---|---|
| h678 | up2p:Movie/h267 | up2p:Movie/h245 |

Table 6: Documents stored by $p_3$. $R^{p3}_{C_{Actor}}$ and $R^{p3}_{C_{Movie}}$ are not shown, as they are identical to the previous example (table 4.3).