Collaborative Knowledge Construction in a Peer-to-Peer File Sharing Network

by

Alan Davoust, M.Eng.

A Thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute for
Electrical and Computer Engineering (OCIECE)

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
September 2009

Your file Votre référence
ISBN: 978-0-494-60258-4
Our file Notre référence
ISBN: 978-0-494-60258-4

Abstract

The Semantic Web is a vision of a web of machine processable data, encoding knowledge, that agents can reason about to process complex tasks. This vision relies on large amounts of data describing interrelated resources. One approach to producing such data is the "Web 2.0" approach, of collaborative production of content by end users.

We seek an alternative to the centralized control model of web-based solutions.

Our approach uses the principles of peer-to-peer file sharing, where data is also contributed by end users, with decentralized control, and a data circulation model where files are replicated by downloads, causing popular or useful files to spread in the network.

Building on earlier work where this paradigm was applied to structured documents, we extend this approach to support a graph data model. We introduce a formal model for schema-based file sharing systems, then extend this model with links between shared documents. We propose a query language for this graph of interlinked documents. Finally, we describe the design of our prototype implementation.

To people who think differently...

Acknowledgments

I am indebted to my supervisor, Dr. Babak Esfandiari, for his invaluable help. He has guided my learning process through much constructive dialogue, taken time beyond reasonable working hours to read my ramblings or discuss the deeper semantics of semantics, and last but not least, has been generally very pleasant to work with.

The encouragement and support of my loved ones was crucial to get me through this new studying adventure; this goes first and foremost to my better half, Sylviana, but also to my family and friends, who are too many to list here.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

1.1.1 The Semantic Web

With the growing volume of information available on the web, human users are confronted to an information overload, and many see the future of the web in machine-processable data.

The Semantic Web, first proposed by Tim Berners-Lee [1], is an effort of many research communities to develop a layer of machine-processable data on top of the existing web of documents.

The idea of the Semantic Web is to annotate Web resources, as well as resources of our world, such as places, people, or organizations, with structured, semantically rich annotations. This would produce a global knowledge base, that Semantic Web agents could automatically query and reason about in order to perform complex tasks.

Ongoing efforts at the W3C[1] have produced a roadmap to the Semantic Web, involving functional layers starting with basic syntax, going on towards semantics, reasoning, proof, and trust.

[1]World Wide Web Consortium

The Semantic Web *Layer Cake* is illustrated in Figure 1.1.

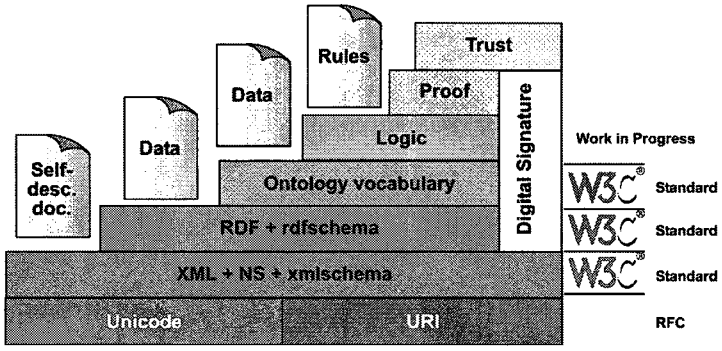


Figure 1.1: The Semantic Web Layer Cake.

Many technologies involved in these different layers have been the object of W3C standards and recommendations.

The first level of this "cake" simply characterizes standards of the Web infrastructure. The second level is where the Semantic Web branches off from the traditional Web of documents: the now standard XML (Extended Markup Language), is the language of choice for structured data. The purpose of the following level, constituted by the Resource Description Framework, is to allow data on the web to be organized into a graph data model. Annotations can be related to documents, or to other annotations, forming a global graph of interrelated data.

The next level up, of ontology vocabularies, adds a semantic level on top of the graph of data. Controlled vocabularies support interoperability across systems and languages, by providing semantic bridges between different data representations. The envisioned agents of the Semantic Web will understand complex semantic queries, and use logic to reason over data, using ontologies as their knowledge base.

Finally, these agents will be required to be able to prove their findings, and we will need a framework to establish our trust with respect to agents, and to the data that they will reason from.

Despite many years of efforts put into this cake, usage of the Semantic Web seems

to be still confined to marginal, highly technical communities. Casual web users are left waiting for the Semantic Web "killer application".

1.1.2 User-contributed Knowledge

These casual end-users of the web are at the heart of a different revolution, dubbed the "Web 2.0", that can be described as a shift in the dynamics of content creation, towards collaborative contribution by casual end-users [2]. New web technologies such as AJAX have enhanced the interactiveness of web pages, making browsers more of a two-way communication tool.

Companies have come up with interactive concepts, such as online storage for digital photographs, and users have answered *en masse*, collectively building millions of pages of content, and terabytes of data.

Wikipedia[2], a community-edited encyclopedia featuring millions of articles in dozens of languages, has sprung up from nothing in less than ten years. Facebook[3], at time of writing the most popular social networking site, has attracted over 200 million registered users in less than five years. Other successful initiatives have amassed large amounts of photographs, genealogical data, cooking recipes, factual information and opinions about virtually any imaginable topic.

An obvious challenge to the broad Semantic Web research community is to apply the Web 2.0 paradigm of user-contributed content to *semantic* data, as a means to collaboratively build the Semantic Web.

This challenge has been taken up by many projects, some in a quite straightforward manner, such as *semantic wikis*, or "databases about everything", such as Freebase[4]; and others attempting to indirectly produce formal knowledge from Web 2.0 data repositories, e.g. by extracting formal knowledge from Wikipedia.

[2]http://www.wikipedia.org
[3]http://www.facebook.com
[4]http://www.freebase.com

We can see two important reasons for the tremendous success of the Web 2.0 revolution: first, users want to express themselves, share all sorts of information on the web, as opposed to simply consuming information. This has been possible from the early days of the Web: many people owned personal pages, where they could publish whatever they wanted. Communication now goes further, as annotations such as comments generate a new sense of dialogue.

The second factor is the lightweight aspect of the Web 2.0 paradigm: users only need a browser, which is obviously a change from traditional personal pages, which required some technical skills, and non-trivial (and possibly costly) processes to secure hosting services, and maintain the pages.

On the Web 2.0, sites now provide user-friendly tools for users to edit and upload content, such as Wikis, and typically host the content for free.

1.1.3 Decentralized Control and Peer-to-peer

However, while the Web 2.0 paradigm appears to place users in a controlling position, the host sites ultimately retain full control of the users' data : wikis may block users from contributing, and the registration to most sites comes with terms and conditions established by the host; typically administrators retain the right to edit or remove any data at their discretion.

An interesting problem is then to provide a contribution model similar to the Web 2.0, but where end-users retain the control of their data. This may be desirable for many reasons: we could cite privacy concerns, as in the case of social networks, or simply because a centralized knowledge repository may not be able to accomodate all different viewpoints: Wikipedia frequently sees controversies erupting from opposing viewpoints on an article. In addition, if a service disappears or is unavailable, the data disappears along with it.

In order to achieve such decentralized control, and data availability without a

central point of failure, users must host their own data: the solution is then a peer-to-peer architecture.

The peer-to-peer (P2P) paradigm is precisely characterized in opposition to the client-server paradigm: all the participants in the network have interchangeable roles of *peers*, as opposed to *clients* consuming information hosted by *servers*.

The most popular application of the P2P paradigm is file-sharing, an application where users connect and exchange files, typically media files such as music and films.

P2P File-Sharing applications typically come in the form of a piece of software that each user installs on her computer, requiring minimal technical skills. P2P File-Sharing applications can thus be described as a lightweight solution to sharing data, a characteristic that was key to the success of the Web 2.0. In addition, file-sharing networks are characterized by the "darwinian" emergence of certain files : as users download files of interest and replicate them in the network, popular files become highly available, despite high levels of churn.

However, existing P2P file-sharing solutions are much less versatile than the Web 2.0 when it comes to different types of content. Traditional file-sharing applications offer only very limited support of structured data, usually in the form of built-in metadata for media files.

This limitation is clearly an obstacle to share data representing knowledge in a P2P file-sharing network. With respect to the Semantic Web "layer cake", support for the layer of structured data has been addressed by several projects. The work presented in this thesis focuses on the next layer, that we could call the *linked data* layer. This layer enables data items to be annotated and interconnected, in what we see as the next step towards knowledge representation.

1.2 The Problem of Collaborative Knowledge Construction

We base this work on an existing prototype file-sharing application called U-P2P, first presented in [3], which allows users to share structured XML documents. Users may contribute new documents of a given schema, but also define new document types, or concepts, in the form of XML schemas and customized presentation templates. As in traditional file-sharing applications, these new documents and schemas can also be downloaded by other peers, and thus propagated across the network, to be reused by others.

U-P2P thus addresses the problem of building collections of structured documents describing various concepts, using the collaborative and decentralized P2P file-sharing paradigm.

Our challenge is to further extend this principle, in order to enable the construction of knowledge, i.e. data describing interrelated concepts. We thus formulate our research problem as follows:

> **Research problem:** How can the decentralized contribution model of peer-to-peer file-sharing be applied to the collaborative construction of knowledge ?

More specifically, we have identified the following requirements :

- The contribution model of P2P file-sharing includes both the aspect of decentralized production and control, and the file-sharing aspect, where useful contributions can be downloaded and propagated in the network;
- The problem of constructing *knowledge*, in our view, requires a way to annotate, and to relate documents to one another;

- Such a collection of interrelated resources also requires query mechanisms, which are necessary both to discover relationships, and to retrieve the related documents.

1.3 Overview of our Solution and Contributions

Our solution to this problem is a graph data model suitable for interlinking documents shared in a P2P file-sharing network.

Our approach is based on a formal model of peer-to-peer file-sharing, which lays the foundations of P2P file-sharing for structured documents.

We define the fundamental concept of a community, and show how a community can be used as a meta-model to represent a concept, where documents shared in the community will represent instances of this concept. We model the transient relationship of a peer participating in a file-sharing community, and formalize the data management operations available to peers. Finally, we show how multiple communities can be managed in a single network.

The second contribution of this thesis is an extension to this formal model, that defines a graph data model in a P2P file-sharing network. This data model relies on a naming scheme suitable for documents shared in a multiple-community network. This naming scheme makes use of system-generated identifiers for the communities, and for documents within each community, such that a name identifies any copy of a document within the file-sharing network. This naming scheme allows links between documents, which in turn induce a graph of interlinked documents.

Our third main contribution is a class of graph queries for this data model, and algorithms to answer such graph queries in the file-sharing network, using the fundamental operations available to peers in each community. In addition, we apply again the file-sharing principle to these queries, and show how they can themselves

be shared in the network, in order to support the reuse of queries between peers.

Finally, we present an extensible design for a file-sharing application supporting these features, which reuses and extends the previously existing prototype U-P2P.

Our design is based on a tuple space, and demonstrates the use of this computation model for processing complex tasks with asynchronous characteristics, such as graph queries over a file-sharing network.

1.4 Organization of the thesis

Following the introduction, Chapter 2 gives a brief overview of RDF, the main graph data model in use in Semantic Web research, and of SPARQL, the main query language for RDF.

Chapter 3 reviews existing approaches to collaborative construction of knowledge for the Semantic Web.

Our formal model for P2P file-sharing systems is presented in Chapter 4. Our extension of this model for graph data is presented in Chapter 5, along with the graph queries supported by this model.

The design of our prototype file-sharing application is presented in Chapter 6. Processing of queries in a tuple-space is presented, along with a complexity analysis.

Chapter 7 presents a brief case study where we executed graph queries over a collection of interlinked documents built from Freebase data.

Finally, Chapter 8 presents our conclusions and some directions for future work.

Chapter 2

Background: RDF and SPARQL

In the layered view of the Semantic Web Layered Cake, our research problem focuses on the layer concerned with relating data elements to one another. The W3C standard for this layer is the Resource Description Framework (RDF, [4]).

Although our approach does not focus on the specific data model of RDF, parallels can be drawn between our work and RDF, so it is worthwhile giving a brief overview of RDF, which is done in Section 2.1.

The query language for RDF that is now an official W3C recommendation is SPARQL, and our approach to graph queries can also be compared to SPARQL. We give a brief introduction to SPARQL in Section 2.2.

2.1 The Resource Description Framework

The Resource Description Framework, [4], is a W3C standard data model, that can be used to annotate resources, or to relate resources to one another by *links*.

The fundamental concepts of RDF are *resources* and *properties*. The fundamental RDF construct is a *triple*, which represents a logical statement about a resource. A triple has a *subject*, a *predicate*, and an *object*. The subject of the statement is a resource, the predicate is a property, and the object may be a resource or a literal

value[1].

2.1.1 Resources

In the context of the Internet, virtual objects accessible to users, such as files, were originally designated by the term *resource*. In the context of the Semantic Web, this term has come to be a standard way of designating any identifiable entity that one may want to describe or refer to [5]. This includes both web-accessible documents, and other concepts such as persons, places, artifacts, that are not directly accessible through the web. The former are known as *information resources*, and the latter are conversely termed *non-information resources* [6].

2.1.2 Uniform Resource Identifiers

In order to unambiguously identify *resources* on the Semantic Web, a formal identifier syntax was defined as part of Web Architecture standards: Uniform Resource Identifiers (URIs) [5].

The standard URI syntax includes a protocol by which the resource may be retrieved: this retrieval process, via the protocol specified in the URI, is called *dereferencing* a URI. This protocol is typically HTTP.

2.1.3 RDF as a Graph Data Model

RDF data can be viewed as a directed labeled graph: resources are represented by vertices, and statements describing these resources are represented by directed edges from the subject to the object of the statement, labeled with the predicate.

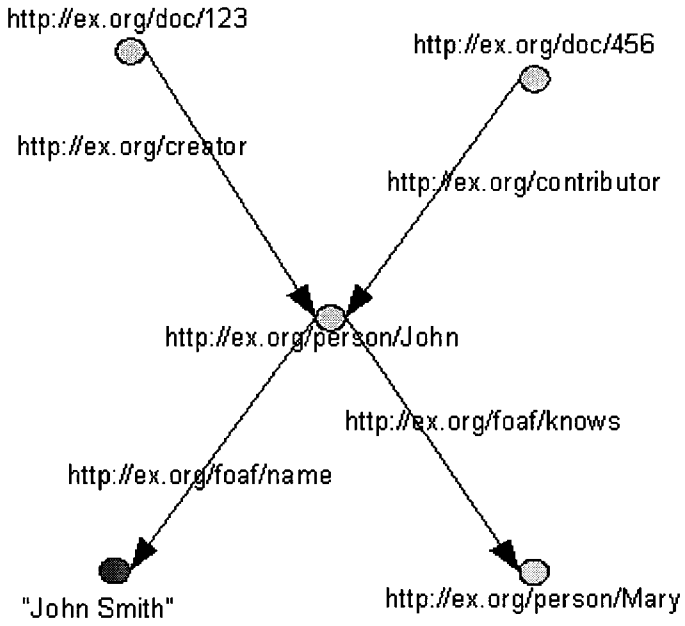
Table 2.1 provides an example of four triples describing relationships between four resources.

[1] we ignore the case of blank nodes for the sake of simplicity

Subject	Predicate	Object
http://ex.org/doc/123	http://ex.org/creator	http://ex.org/person/John
http://ex.org/doc/456	http://ex.org/contributor	http://ex.org/person/John
http://ex.org/person/John	http://ex.org/foaf/name	"John Smith"
http://ex.org/person/John	http://ex.org/foaf/knows	http://ex.org/person/Mary

Table 2.1: Example RDF triples.

The graph of data represented by these triples is illustrated in Figure 2.1.3.



Aside from the literal "John Smith", all the nodes in the graph are identified by URIs, which unambiguously identify multiple occurences of a resource. For example, the URI `http://ex.org/person/John` repeated in all the triples is represented in the graph by a single vertex, whereas multiple occurences of a literal such as `"John Smith"` would not be considered related.

2.2 SPARQL

Graphs, such as the one illustrated in the previous section, support a number of query languages. The official W3C recommendation language is SPARQL.

We give here a brief overview of the main constructs of SPARQL. See [7] for the W3C specification, and [8] for an in-depth analysis of the language and its semantics.

SPARQL is a declarative query language for RDF. The two main types of queries that can be made are `SELECT` queries and `CONSTRUCT` queries.

2.2.1 SELECT Queries

The following example shows the usage of the `SELECT` construct:

```
SELECT ?person
WHERE
{
  <http://ex.org/doc/123>  <http://ex.org/creator> ?person .
}
```

The question mark in the string `?person` indicates a free variable of the query: answers to the query are bindings for this variable.

The second part of the query, after the `WHERE` keyword, constitutes the body of the query, and expresses a graph pattern. Graph patterns are expressed as a sequence of triples including constants (which may be URIs or literals), and variables.

If we apply this query to our example graph (Table 2.1), the graph pattern would only match the first triple, by binding the variable `?person` to the URI `http://ex.org/person/John`: this URI would thus be the only answer of the query.

2.2.2 CONSTRUCT Queries

In `CONSTRUCT` queries, both parts of the query — before and after the keyword `WHERE` — specify graph patterns by sequences of triples, as in the following example:

```
CONSTRUCT
{
?thing <http://myprefix/artist> ?person
}
WHERE
{
  ?thing  <http://ex.org/creator> ?person .
}
```

The query is again answered by matching the second pattern (after `WHERE`) to an RDF graph. For each match, the bound variables are used to construct an RDF graph based on the template provided by the first pattern. The answers to the query are thus RDF graphs.

2.2.3 Graph Pattern Matching

The core principle of SPARQL queries is thus to provide templates that match patterns in RDF graphs, by binding free variables to literals or URIs of the graph.

The simplest graph patterns are sequences of triples, which are implicitly conjunctive queries: all the triples of such a pattern must be matched to a graph in order to produce an answer. In addition to triples, constraints can be expressed over the variables by the keyword `FILTER`. SPARQL supports a number of boolean conditions over literals, such as numerical operators (=, <, etc.) or regular expression matching for character strings.

Finally, simple conjonctions of triples and constraints can be further combined in disjunctions, by using the operator `UNION`. Optional patterns can be added using the operator `OPT`

Note that a number of more advanced features of SPARQL are not covered here. However, we expect this introduction to be sufficient for the reader to appreciate the comparisons which will made between SPARQL and our approach to graph queries.

Chapter 3

State of the Art: Collaborative Construction of Knowledge Bases

3.1 Introduction

The general setting of our research problem is the situation where a repository of knowledge is generated from several sources, either by putting together several pre-existing collections of data, or by several users collectively (with more or less coordination) creating a single repository.

In this review we thus present approaches based on decentralized infrastructures, as well as decentralized contribution approaches for centralized repositories.

In Section 3.2, we review different approaches to collaborative data contribution, comparing Web 2.0 approaches with peer-to-peer approaches. In Section 3.3 we first present different graph data models supported by these knowledge repositories, and secondly review approaches to querying such data models in a decentralized setting.

3.2 Data Contribution for the Semantic Web

The different infrastructures which have been used for distributed construction of knowledge can be organized according to their contribution model. We distinguish the following broad categories:

- Web repositories, i.e. centralized, web-based infrastructures to publish and store data, that many remote users contribute to via web browsers. This category can further be subdivided into two sub-categories:
  - Collaboratively edited collections, such as Wikis, where any user can generally access and edit any data in the repository (with some limits, which we will discuss);
  - "Social" Web 2.0 sites, where different users have separate accounts storing specific portions of the data;
- Peer-to-Peer (P2P) data management systems, which support more decentralized control: the data storage infrastructure is physically distributed in a P2P architecture; each contributing peer also stores part of the data. We can distinguish two sub-categories of systems, which differ by their approach to distributed storage:
  - Systems of interconnected autonomous peers, where each peer is in full control of its local system;
  - Systems with protocol-determined distributed storage, such as anonymous storage systems or Distributed Hash Tables.

3.2.1 Collaboratively Edited Data Repositories

In this section we review approaches based on the idea of collaborative edition. The data is stored in a centralized, web-accessible repository, and casual users (in some case any user, in some case a list of registered users), may edit any part of the data through interactive functions of an ordinary web browser. This contribution model is illustrated in Figure 3.1[1].

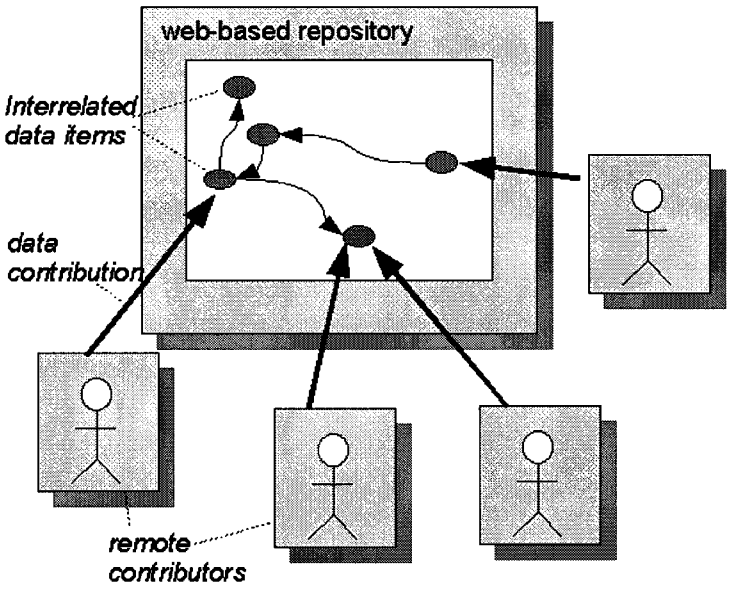


Figure 3.1: Architecture and contribution model of collaboratively edited repositories.

The simplest embodiment of this principle is the concept of a *wiki*, invented by Ward Cunningham for his WikiWikiWeb site[2].

Wikis are web sites where users can modify existing web pages, and create new ones. This principle has been mostly applied to web pages containing text, but also to structured data.

[1]Note that the data items represented in this diagram, as in others in this chapter, are related by links, forming a graph data model. Although a graph data model is the focus of our research problem, we also review here repositories that do not support explicit links, for the purpose of illustrating particular contribution models, for example.

[2]http://c2.com/cgi/wiki

Wikipedia

The most famous Wiki is probably now the online encyclopaedia Wikipedia [3], which contains a combination of structured data ("infoboxes") and unstructured data (the text of articles). While Wikipedia does not host formally structured knowledge (as in an ontology), the collaborative construction of this knowledge is a good illustration of the principles that Wikis for structured data rely on; furthermore, several efforts have focused on extracting formal structured knowledge from Wikipedia, which we will briefly review below.

The central idea of Wikipedia is that, at least for objective facts such as those described in an encyclopaedia, a consensus should emerge from collaborative edition. A first user places data on a page, and this page is read by many other users, who may modify it if they disagree. The expectation is that the construction of an article comes like a discussion, and the content eventually reflects a consensus and stabilizes.

Overall, this principle seems to be successful: despite the fact that many non-expert users edit articles, the scientific accuracy of the articles is generally considered to be good, as illustrated by a comparison of Wikipedia and Encyclopaedia Brittanica published in Nature magazine [9].

However, this principle sometimes breaks down, and a consensus fails to materialize. The administrators of Wikipedia must sometimes block specific editors, or protect controversial articles from repeated edition ("wiki-vandalism").

Indirect Knowledge Extraction from Wikipedia

The DBpedia project[4] [10] is an effort to export the structured knowledge available in Wikipedia infoboxes to a web-accessible RDF knowledge base. While DBPedia is not directly collaboratively edited, the knowledge it represents reflects the result of

[3] http://www.wikipedia.org
[4] http://dbpedia.org

the collaborative edition of Wikipedia, taken at a snapshot moment.

DBPedia is now an important component of the Linked Data On the Web initiative (see Section 3.2.3), and accounts for a large majority of the semantic data available on the web, in terms of potential returns to user queries [11].

Other projects have explored the idea of extracting structured knowledge from Wikipedia text using natural language processing techniques, such as Wanderlust [12].

Semantic Wikis

Several projects have taken up the idea of wikis to collaboratively edit ontologies. Such *semantic wikis* include SweetWiki [13], OntoWiki [14], which we review here, and many others.

OntoWiki is a tool to browse and collaboratively edit an RDF knowledge base. Users can browse the ontology of concepts, and, exactly as in Wikis, edit or add properties to the concepts that they are viewing, or add new concepts.

While most properties are shown in a generic tabular form, map and calendar views are built-in to OntoWiki in order to show resources based on geographical or time-related properties.

OntoWiki also supports an additional meta-data layer of "social annotations" on top of the RDF ontology, based on the concept of reification: RDF statements are themselves data that can be annotated, for example to comment on their validity. In addition, the history of modifications is saved, as in a traditional Wiki, so that the rollback of any editing action is possible.

Freebase

Freebase [15], while not exactly a wiki, is a collaboratively edited database, where end users can input and edit structured content stored in tables. Freebase requires registration, but users may modify data contributed by others.

Users may create tables listing instances ("topics" in Freebase terminology, similar to the RDF concept of a "resource") of particular concepts, and list properties of the topic in a tabular form. Properties can in turn be topics, and have properties of their own.

Just like Wikipedia, Freebase data can be the subject of vandalism, and administrators have set up some protection mechanisms, such as automatic consistency checks and protecting certain data from edition [16].

3.2.2 Social Web Sites

We review here the account-based contribution model characteristic of Web 2.0 sites where the general public may contribute data after registration, often known by the general denomination of social networking sites. The basic principle of these sites is that casual web users create accounts, and are allocated some storage space on a centralized domain, where they may upload data which remains associated with their username, and, in the cases that we have reviewed, technically remains the property of the user. As a consequence, users cannot modify one another's data.

This contribution model is illustrated in Figure 3.2.

Social Networking and Annotation

The principle of social networking sites is usually to provide a personal space on the web, where users store and / or describe resources of interest to them. Users may also connect to friends by means of various forms of messages, or simply by looking into these virtual windows into one another's online lives.

In the context of such web sites, users upload various forms of content — photographs, bookmarks, blog posts, etc. — and can annotate these resources, with some form of meta-data. The most common form of annotation is *tags*, short keywords that can be associated to some resource, and variations include reviews and

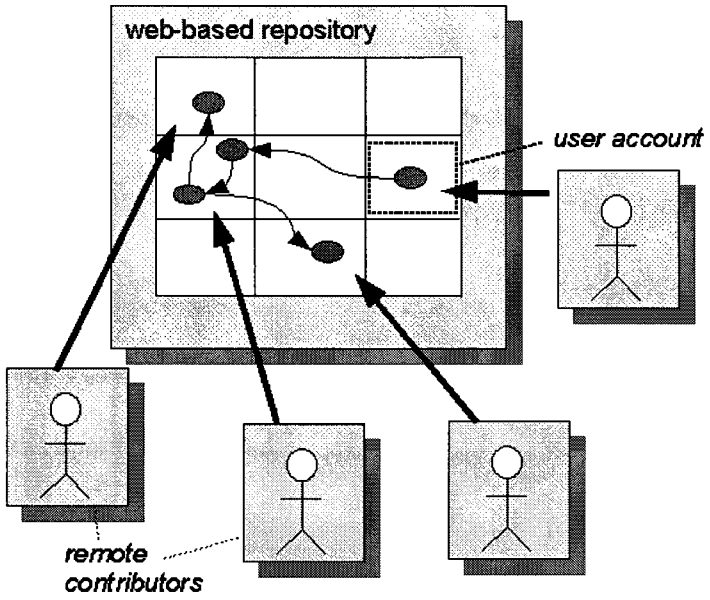


Figure 3.2: Architecture and contribution model of Web 2.0 account-based data repositories.

comments (containing longer freeform text), ratings (restricted to a numerical interval, such as 0-5 stars), and boolean "approval" or "recommendation".

Tags are used in many web sites hosting specialized collections of resources such as photographs (Flickr[5]), web bookmarks (del.icio.us[6])), or academic citations (CiteULike[7]). Social Networking sites such as Facebook[8], LinkedIn[9], MySpace[10] and hundreds more, host complex data in the form of user profiles.

As an example, we describe the social networking site Flickr, which is focused on digital photography.

Users register and are allocated some storage space to upload and store photographs. User profiles contain a number of properties such as name and geographic

[5]http://flickr.com
[6]http://delicious.com
[7]http://www.citeulike.com
[8]http://www.facebook.com
[9]http://www.linkedin.com
[10]http://myspace.com

location. The photographs uploaded by each user can be annotated with a title, description, tags (simple keywords), a geographic location, a date, and with privacy and license settings. Photographs may also be commented, and a graphic tool allows users to select specific areas of photographs, which they can then apply their comments to. For example, a user may select a rectangular area and suggest, in a textual comment, that the photograph could be reframed to that area.

The ownership status of all this data is much clearer than in the case of collaboratively edited knowledge, since no single data item belongs to multiple users[11] (e.g. comments are authored by clearly identified users, and cannot be edited by others).

Indirect knowledge extraction from Social Annotations

A number of projects have attempted to build ontologies from existing collections of social annotations.

The main approaches (reported in [17] and [18]) used the structure of the social annotation knowledge base to infer the semantics of the tags.

In this case, as in the project DBPedia (see section 3.2.1), an ontology is created in a single effort from a separate, collaboratively built, knowledge base with different characteristics.

Google Base

Google Base[12] is a "database about everything" where users can upload all kinds of structured data, or describe "items" through web forms.

Each entry in the database ("item") is described by an item *type*, and a number of key-value attributes. Google base supports a number of predefined item types[13]

[11] although the problem of online information where several users have a stake, such as privacy concerns related to photographs where two users appear, presents very interesting philosophical problems. In the context of this thesis we will however ignore such issues.

[12] http://base.google.com

[13] the predefined types are: Course Schedules, Events and Activities, Footprint, Housing, Jobs,

with specific schemas (a suggested list of attribute keys). Users are free to create new item types, and to add attributes with custom names to any item.

In describing items, users can use some predefined attribute names which have semantics, i.e. the values entered by the user are interpreted in a certain domain by the application. For example, *date* and *location* attributes can be viewed directly on calendar and geographic map views, respectively.

In addition, users may upload "attachments", which are files that can be associated to identified items. However, no specific attribute name can be given to relate the item to the attachment.

3.2.3 Networks of Autonomous Peers

In parallel to the tremendous development of the World Wide Web, in its standard client-server paradigm, the era of digital music brought the explosion of Peer-to-peer (P2P) file-sharing networks, spearheaded by Napster[14].

The principle of peer-to-peer data exchange is the idea that hosts are not confined to a client or server role, but can arbitrarily connect to one another and exchange data in any direction.

While the most popular application of the P2P paradigm has been file-sharing between casual users, P2P architectures have also been applied to knowledge repositories.

In this section we briefly describe the original P2P file-sharing principle, then present extensions of this principle that have focused on sharing knowledge.

We focus here on extensions of the P2P file-sharing model that have retained the original principle of *autonomous interconnected peers*; as opposed to structured P2P networks, which we will cover in section 3.2.4.

Local Products, News and Articles, Personals, Products, Recipes, Reference Articles, Reviews, Services, Travel Packages, Vacation Rentals, Vehicles, Wanted Ads

[14]http://www.napster.com, no longer functioning in its original form

The principle of P2P data exchange by autonomous peers is illustrated in Figure 3.3.

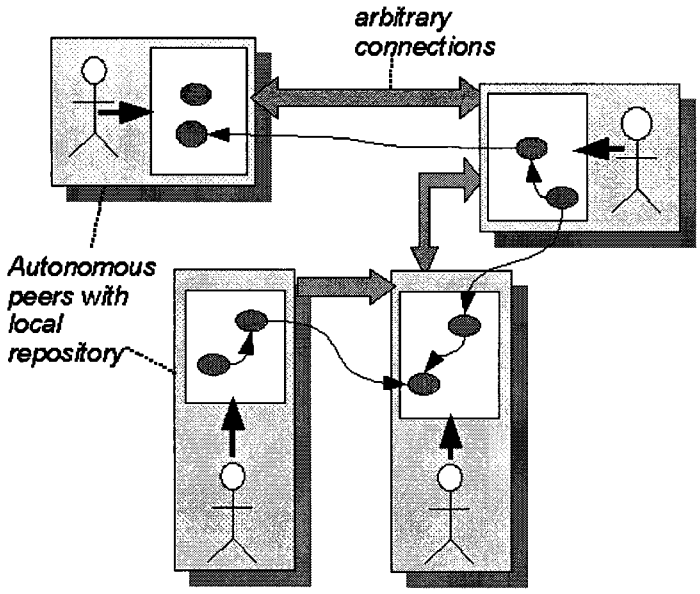


Figure 3.3: Contribution model in Peer-to-Peer networks (autonomous peers).

Peer-to-peer File-Sharing

Traditional P2P file-sharing applications such as Napster, Kazaa[15], the Gnutella [19] network (including applications such as Limewire[16] or Bearshare[17]), are mostly geared towards audio and video files with built-in metadata.

The popular music file format MP3, for example, includes built-in placeholders called the *ID3 tags*, for music-specific annotations identifying the artist, title, genre, and a few other attributes of the piece.

What we consider to be the defining principles of P2P file-sharing are the following characteristics: an unstructured network, with highly transient peers, where each peer is in full control of its local storage space, and may freely connect to any other peer.

[15]http://www.kazaa.com, no longer exists in its original form
[16]http://www.limewire.com
[17]http://www.bearshare.com

Peers in a P2P file-sharing network download files from one another, in such a way that data becomes available from multiple locations.

To the general data contribution principle illustrated above, which characterizes networks of autonomous peers, the particular application of file-sharing adds the data circulation and replication generated by downloads. As popular files are more often downloaded, these items end up having a higher availability despite the transient nature of most connections [20].

Bibster

An application that has retained the characteristics of traditional file-sharing systems, and been extended to support more expressive knowledge representation, is Bibster [21], which is a platform built using the JXTA protocol, to share bibliographic data (Bibtex entries).

Bibster is supported by two built-in ontologies of the described domains: an ontology of the concepts related to bibliographic citations (e.g. author, title, URL, etc.), and an ontology of topics covered by the listed publications. The latter ontology was built by the ACM[18].

Users of Bibster make their bibliographic data available to other users, as in a traditional file-sharing application, and this data is automatically aligned to the central ontologies of Bibster. Users may query the network based on concepts of the ontologies, e.g. browse computing topics for bibliographic entries related to that topic.

Users do not modify the existing ontologies: they simply populate existing concepts with instances, and use the ontologies as a support for semantic queries.

[18]Association for Computing Machinery, a scientific society focused on computer science and closely related subjects

U-P2P

We present briefly our prototype file-sharing system U-P2P [3], in its existing form prior to this thesis.

U-P2P is essentially a schema-based file-sharing system, where the shared data items are XML documents, to which we can add binary attachments, following the principle of email attachments.

A single U-P2P client may manage several XML schemas, and can connect to other peers that manage the same schemas. The documents shared in the network are thus shared within schema-specific communities, and peers may simultaneously be members of several such communities.

With respect to data contribution, the distinguishing feature of U-P2P, as opposed to other schema-based systems, is that U-P2P peers may publish and download from one another new schemas to model any domain. A peer who downloads a new schema, which comes with a set of presentation templates to search and view data structured according to the schema, automatically joins the community of peers using this schema.

Example communities implemented in our prototype manage bibliographic data, molecule descriptions, annotated images, shakespeare plays, and more.

U-P2P is an implementation of the formal model for multiple-community file-sharing systems presented in Chapter 4.

Edutella

Edutella is a P2P network built using the JXTA framework, where the different peers manage RDF data describing educational material (e.g. books, on-line course notes, etc).

In Edutella, each peer autonomously manages its database of RDF data, and

connects to a network of Edutella super-peers, through which the peer may query the global collection of RDF data.

We note that while the ultimate purpose of the system is to allow end-users to discover and share educational resources (motivated by the idea of P2P file-sharing), queries only return URIs for the resources, and the actual dereferencing and downloading of these resources is not directly covered by Edutella. The RDF metadata managed by Edutella is not meant to be downloaded and replicated as in a file-sharing system.

Peer Data Management Systems

Peer Data Management Systems (PDMS) are networks of autonomous interconnected Database Management Systems.

The principles of PDMS have been applied to relational data, XML data, and are similar to that of Edutella in the sense that each peer maintains an independent database, and can query the global collection of data through P2P connections. The main difference with Edutella is that in PDMS, the databases of the different peers are assumed to have different schemas, and thus processing queries over the global collection of data requires semantic integration between the different peer schemas.

PDMS for relational data were formalized by Calvanese et al. in [22], and their principles can be summarized as follows. Each peer is a relational database management system, with a schema formed by a set of relations. The relations in this *Peer Schema* are mapped through *schema mappings* to local data sources, or to the schemas of other peers.

Queries received by a peer, formulated according to its *peer schema*, are rewritten as subqueries over the peer's data sources and over other peers' schemas, using the mappings. Query processing thus involves a series of rewritings, where the original query is decomposed into subqueries, which are recursively propagated across the

network through mappings, and answers are propagated back to the initiator of the query.

Piazza [23] is a PDMS that can manage relational and XML data. Case studies have been presented for relational data describing a network of emergency services (hospitals, fire-fighting services, etc.) [24], and for XML data describing scientific publications in various repositories [25].

SomeRDFS [26] is an implementation of a Peer Data Management System for sharing ontologies, which are expressed in a limited fragment of description logics.

The queries managed by SomeRDFS involve T-Box reasoning in this distributed network. For example, a typical query will be to retrieve the instances of a certain class C, which involves determining all the subclasses of C that can be inferred by the different T-Boxes and mappings.

Query processing in SomeRDFS follows the same general principle of recursive rewriting and propagation, as in PDMS for relational or XML data, with different constraints due to the different logical models.

The Linked Data On the Web Initiative

The Linked Data On the Web initiative (LDOW) is a recent collective effort by a number of organizations to publish the contents of large databases to web-accessible RDF repositories, and to interconnect these repositories.

A view of the global "data cloud" generated by this initiative can be seen in Figure 3.4.

One of the first datasets published in this context was DBPedia, that we have briefly reviewed in Section 3.2.1. While DBPedia itself is the result of the collective edition of Wikipedia, it can be seen in the context of LDOW as a single contributor to the greater data cloud.

All the participants of this initiative, organizations that have published their data

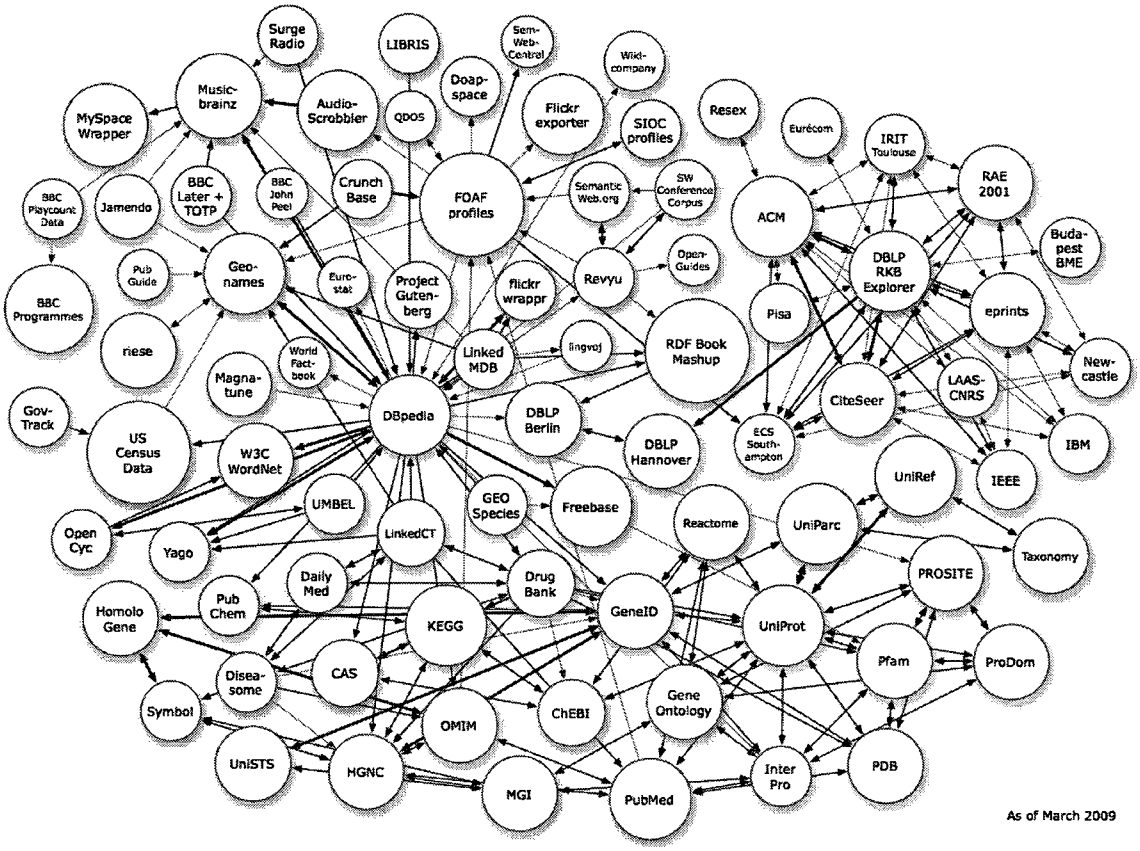


Figure 3.4: The Linked Data Cloud as of March 2009

in the form of a publicly accessible RDF repository, can be seen as autonomous contributors to a greater knowledge cloud, seen by some as a first realization of the Semantic Web. Each participant organization offers a query interface to query the subgraph of data hosted by this organization, but no single interface provides access to the entire collection of data, unlike PDMS, where the idea is that each peer provides an access point to query the global collection.

3.2.4 Structured P2P networks

In structured P2P networks, peers contribute both data and storage space, and the collection of contributed data is distributed over the global storage space according to a defined protocol. Connections between peers (and hence the network structure) are also usually determined by the protocol.

Most structured P2P networks are based on the principle of Distributed Hash Tables, but we also include in this category anonymous distributed storage services such as Freenet [27].

The contribution model of structured P2P networks is illustrated by Figure 3.5.

Distributed Hash Tables

In Distributed Hash Tables (DHT), each peer contributes some storage space, and the full collection of data is distributed in the network in a way that retrieval of any data item is optimized.

The main purpose of distributed hash tables is to use a large number of machines to produce a scalable infrastructure, which can then be used as if it were a single seamless shared space.

Issues such as data ownership and access control, induced by the multiple contributors to the DHT are considered to be a logical problem to be addressed at an

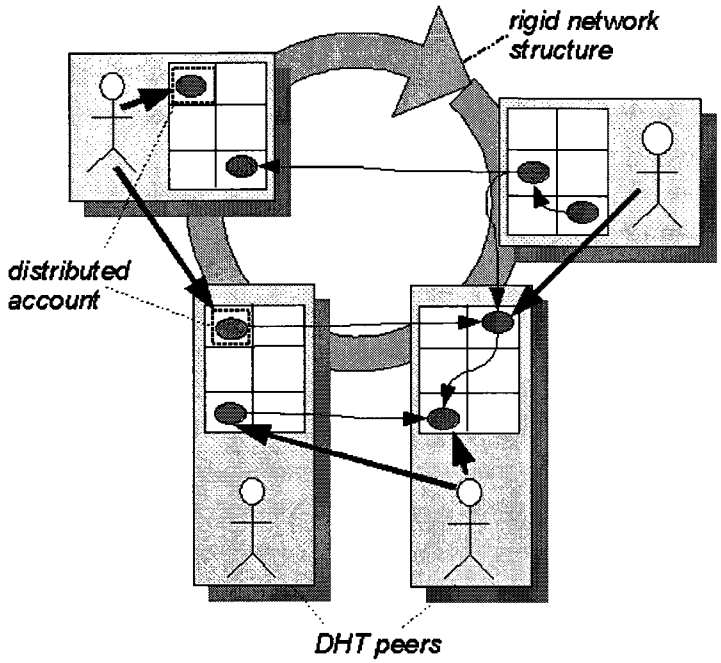


Figure 3.5: Contribution model in DHT-based repositories.

application level, on top of the DHT infrastructure. However, some DHT such as OpenDHT [28] natively include access control protocols.

In other words, the DHT defines a centralized logical layer, that abstracts from the underlying physical distribution.

The physical distribution is handled by the following principles. For each data item to be stored, a unique key is generated using a one-way hash function. The key space (i.e. the numerical interval containing the possible keys) is divided up among the peers, who store the documents related to their portion of the key space. In some cases, data is replicated and stored by several peers.

In addition, the network of peers is structured in such a way that peers storing a specific key can be found in a small number of hops, as network hops are often the main factor of time complexity in distributed storage architectures.

In Chord [29], for example, the peers are organized in a ring, and neighboring peers host contiguous intervals of the key space. With this arrangement, the search

for a peer storing a particular key can be done by a dichotomic search. The average number of hops necessary for a search is thus a logarithmic function of the total number of peers.

Examples of generic DHT infrastructures include CAN (Content Addressable Network) [30], Chord [29], Pastry [31], or Tapestry [32].

Anonymous Storage Facilities

A number of research projects have explored the possibility of offering anonymous storage online. Such projects include Freenet [27], Free Haven [33], Publius [34], Anderson's Eternity Service [35], and more. These services make data available on the web but are based on distributed storage in space provided by the users of the system (i.e. a peer-to-peer infrastructure). Various cryptographic techniques are used to make all users and publishers anonymous, and to obfuscate the published data: the storage space appears to contain random data.

The purpose of anonymous publication is to make it difficult to identify the author or the host of any content, thus preventing users from being liable for their content, and encouraging free speech. The lifetime of a published document in the network is determined either at publication time by the publisher (Free Haven, the Eternity Service), or else by its popularity in the network (Freenet); the systems are designed so that no other user (such as administrators) can delete content for any reason. In many cases, once a document is published, even the publisher of that document loses control, so that this user cannot be coerced into removing it.

3.2.5 Summary

The different solutions that we have reviewed are infrastructures for data contributed from multiple users and sources. We analyze here the level of decentralization that these different solutions offer, in terms of *control*.

New technologies of the Web 2.0 have opened up web sites to content contribution from end-users, who then obtain a limited control over the data.

Collaboratively edited repositories, where multiple users access the same data, offer a form of *collective control*. "Social" sites, where the repository is partitioned into user accounts, provide some decentralized control over the data, where each user manages a specific portion of this data.

However, despite the distributed contribution mechanisms, organizations hosting the data retain the ultimate control over the data. The administrators of Wikis may block certain contributors, protect some data from edition altogether, and in social networks and other registration-based sites, terms of service, agreed upon as a prerequisite for registration, typically state that the host organization may remove or edit data at their sole discretion[19].

Hence we can conclude that centralized web-based repositories do not support decentralized control of data.

Peer-to-peer systems can be divided into two broad areas: autonomous data repositories in decentralized architectures, and DHT-based systems, which are collaborative systems for distributed data storage. DHT and autonomous peers provide decentralized control of the data, but in DHT peers give up control of their connections and their local storage space, in return for optimized data retrieval and processing.

A special case of P2P systems, conceptually similar to DHT, are distributed anonymous publication systems, which use cryptographic methods to remove the peers' control over the data. In this extreme case, control is not only decentralized but anonymous.

[19]See for example the Facebook terms of service at http://www.facebook.com/terms

3.3 Support of a Graph Data Model

Within the Semantic Web Layer Cake, we are interested in the layer where resources can be related to one another, forming a graph of data. In this section we review different graph data models, and different approaches to querying the graph of data.

3.3.1 Alternative Graph Models to RDF

The main data model recommended for the Semantic Web is RDF, which we have briefly presented in Chapter 2. We now describe the alternative graph data models of Freebase and of social annotations.

Freebase

From the user point of view, Freebase (Section 3.2.1) data appears to be tabular data in the sense of conventional relational databases, but the values stored in the tables can be references to other "topics", i.e. entities modeled in the database.

The links to Freebase topics are supported by GUID (Globally Unique Identifiers), similar to hash-generated keys. These keys uniquely identify each row in each table, and are intended as unique identifiers for the "real world" entities modeled by the rows.

In this sense, Freebase supports a graph data model, that can be navigated through a browser: one can for example view an actor, browse to a film that the actor played in, then browse to the director of that film, and so on.

In addition to navigation through a browser interface, Freebase offers a custom query API to query the graph of data. Users may create Freebase applications using this API; such applications define *views* on the data of Freebase.

Freebase administrators also create views with custom presentation (similar to Freebase applications), hosted on the site.

This graph data model can be compared with Google Base, which is similar to Freebase in its purpose of being a "database about everything". Google Base does not directly support a graph data model. Each item (resource) stored in the base is identified by an HTTP URL, which effectively enables the retrieval of Google Base resources.

As users may use any keys and values to describe the items, it is in practice possible to reference another Google Base item as a attribute value; however, such references do not seem to have been an intended part of the data model, nor does the Google Base API directly support a graph query language.

Social Annotations

Social annotations made popular by social web sites of the Web 2.0 are essentially based on the three basic concepts of Resources (which can be of various subtypes), Users, and Tags, and can be modeled by graphs: users, resources, and tags can be modeled as vertices, and the relationships between these concepts can be modeled as edges interconnecting these vertices.

As an example, we describe the structure of the knowledge base induced by social annotations on Flickr.

Relations between the different items include the following:

- Users are related to one another by acquaintance ("connection") relationships;
- Users may enter *groups*, which are n-ary relations between users;
- An uploaded photograph is related to the user who owns the picture, but also to users who comment or bookmark the photo as a favorite.
- Photographs can be grouped in *sets* (photographs of a single user) or in *pools* (photographs contributed by multiple users), which are n-ary associations.

- If we consider Tags (i.e. the terms used as tags) to be first-class resources, then the tagging action by a user is a ternary association between a user, a tag, and a photograph.

The graph[20] formed by this data can be queried using a custom API.

Folksonomies

Collections of tagged resources are known as *folksonomies*, a term coined from "folk" and "taxonomy" by T. VanderWal in 2004 [36], and have been the object of much research in the past few years.

In the context of social annotations, tags are seen as instances of a single concept ("tag"), and the knowledge is constituted more by the relationships between different users, resources, and tags, than among concepts denoted by the tagging terms themselves.

In contrast, the "folksonomy research" we describe in this section aims to organize the collection of tags into (ultimately) an ontology of terms, thus adding an additional semantic layer to the graph of tags, users, and resources.

For this purpose, some authors have proposed to extend the concept of tags and explicitly let users annotate resources with RDF statements (as in [37]), or annotate tags with other tags [38], with the idea of letting an ontology indirectly emerge among the tags. Similarly, in the semantic wiki SweetWiki [13], the semantic wiki is augmented by a tagging system that administrators may edit in order to organize tags into an ontology.

Tagonto [39] is a project addressing the same problem from a slightly different angle: users can choose concepts from existing ontologies to use as tags, or relate tags to ontology concepts in order to disambiguate them. The ontology is in a way

[20] or more precisely *hypergraph*, since we have n-ary relationships, which can then be modeled by hyperedges

integrated with the tag collection, creating a larger knowledge base.

3.3.2 Queries in Distributed RDF repositories

In the case of centralized RDF repositories, the problem of answering queries is immediately addressed by query languages such as SPARQL.

Our second problem of interest is thus the problem of answering queries in decentralized RDF repositories.

DHT-based knowledge bases

DHTs (see section 3.2.4) are more of an infrastructure than data repositories *per se.*

The most straightforward application of DHTs is to atomic documents, for which the main query task considered is retrieving documents by their identifier. However, with some modification, such structured overlays were successfully adapted to store structured data, such as triples (RDFPeers [40], GridVine [41]), or relational data (PIER [42], AmbientDB [43]).

We present here the example of RDFPeers [40].

RDFPeers is an RDF knowledge base built on top of a DHT.

For each RDF triple stored in the knowledge base, a hash is generated from each of the subject, predicate, and object. The triple is then stored in exactly three locations defined by the three hash values.

SPARQL queries can be decomposed into atomic queries, conceptually similar to triples with wildcards (e.g. triples of the form $(?, middle, ?)$), and to answer such queries, query agents need only query the peer associated with the hash generated from the provided value *middle*: all triples with this value in the predicate position will be stored on that peer (and other copies will be stored on two other peers).

As for any DHT, all the peers may contribute data to the DHT infrastructure,

and the DHT protocol determines where each atomic data item (each triple, in this case) is stored, logically building a unified triple store.

Edutella

In Edutella (see section 3.2.3), the global collection of RDF data is viewed as a single graph, expressed by distributed data. The main challenge is thus to manage complex queries over this graph, in the distributed setting. Edutella supports a query language based on datalog, and the distributed processing of these queries is managed by a network of Edutella super-peers[21].

The super-peers decompose queries into subqueries, distribute subqueries to the connected "leaf" peers, and aggregate the responses to obtain the final answer to the query.

Edutella queries are more expressive than simple graph queries as the datalog-like features of the language allow inference rules to be included in the query, supporting recursively defined queries such as transitive closure queries. In addition, additional rules are hard-coded into the query processing system to perform basic reasoning based on the semantics of the most fundamental RDF / RDFS constructs. For example, queries take into account the transitivity of the RDFS construct `rdfs:subclass`.

Linked Data On The Web

The different participating organizations of the Linked Data On the Web initiative form a decentralized network, supporting a global collection of RDF data.

Interconnections of these knowledge bases are realized by RDF links, i.e. RDF triples where URIs belong to different domains.

[21] In the super-peer architecture, most peers connect to a small number of central peers, typically with higher processing power, that form the backbone of the P2P network. These peers are called the super-peers.

For example, a concept described in Freebase[22], identified with a Freebase URI, could reference other related concept described in DBPedia, thus connecting the two graphs of data.

These connections make the greater RDF graph of this data cloud a connected graph, but no single interface provides access to the entire collection of data.

The global Linked Data cloud can be explored by means of external tools, which access the interfaces of the different databases.

Tools for this purpose include semantic web browsers such as the Tabulator [44] and mashup applications, such as those that can be created using Yahoo Pipes (see below). A generic language to define mashups on top of the Linked Data cloud was proposed in [45]. This language, called MashQL, is supported by a graphic interface comparable to that of Yahoo Pipes. The main difference with Yahoo Pipes is that the interactive MashQL interface automatically queries the RDF source to discover its schema and available query parameters, whereas constructing a mashup normally requires knowledge of the data schema.

Yahoo Pipes

Yahoo Pipes[23] is a Web 2.0 "social site" of particular interest here, not so much for the graph of data that its social annotations might form, but rather for the particular resources that users share on the site.

The data shared on Yahoo Pipes are mashups ("pipes"), i.e. applications to query web-accessible data. An AJAX-powered interface allows users to interactively create mashups by selecting data sources, filtering and processing the data, then "piping" various such data filtering modules in order to recombine the data and obtain a complex view over the different data sources.

[22]Note that Freebase data is also exposed in RDF as Linked Data on the Web.
[23]http://pipes.yahoo.com

For example, a pipe could retrieve census data listing the population of geographical regions from a web-accessible government repository, and combine this data with geographical data from Google Maps, to organize the population figures on a map.

In effect, the interface allows users to visually program query applications, which are then hosted on their Yahoo account. Pipes may be annotated with a few properties, including tags, and can be "cloned" by other users (i.e. copies of the pipe are then stored on the other users' accounts), so that these users can reuse the pipe and modify it to suit their particular needs.

The interesting aspect of Yahoo Pipes is that these Pipes provide user-friendly query interfaces for casual users to query the web of data constituted by other web-based repositories.

3.3.3 Summary

The main graph data models that appear to be suitable for the Semantic Web are RDF, and the Freebase data model, where any concept can be described and related to other concepts described in the database. Social annotations can also be seen as a graph of data, for which several research projects have attempted to add a semantic layer.

Peer Data Management Systems based on relational or XML data do not support a graph data model. While the internal database of each peer may describe interrelated resources (e.g. if the data base was designed after an entity-relationship model, the database can be seen as describing a graph of entities related by relationships), a PDMS does not define a global graph of data, but rather a set of disjoint graphs, that must be integrated at the semantic level.

The same observation can be made of Bibster, the semantically enhanced application to share bibliographic data. The bibliographic data shared on Bibster does not form a graph of data. Navigation and discovery of related data items (citations, in

this case) is done at the semantic level, based on purpose-designed ontologies.

The different Web-based approaches participating in the Linked Data cloud form a global cloud of data, but the distributed infrastructure represented by the multiple web servers is not meant to support queries through a unique access point. The Linked Data approach rather relies on the availability of specific query services for each sub-graph, and separate tools and applications capable of accessing the different query APIs of the participating organizations.

3.4 Conclusion

From this analysis, it appears that none of the existing solutions to the collaborative construction of knowledge satisfies our requirements in terms of contribution model, and support for interconnection of resources.

Many Web 2.0 repositories present a data model well suited to knowledge representation, particularly sites managing RDF data. In addition, we note that the "cloning" of mashups offered by Yahoo Pipes can be compared to data downloads in a P2P file-sharing approach, and motivates our approach to making graph queries reusable (see Section 5.5).

However, the main drawback of web-based solutions is that they do not support decentralized control.

The main solutions to provide decentralized control can be described as *peer-to-peer* approaches.

Structured P2P networks, such as distributed hash tables, present a logically monolithic infrastructure, hiding the physical distribution of the peers by protocols that manage data storage and retrieval. Over such infrastructures, centralized repositories with various forms of access control can be established. However, the participating peers are not fully *autonomous*, with respect to their connections to other

peers and their local storage.

The related work most relevant to our research problem are the applications Edutella and Bibster, which are file-sharing applications extended with semantic annotations, and semantic integration in the case of Bibster.

However, in both cases, the principle of propagating data through downloads is not applied to this semantic data.

In Edutella, users query a global repository of RDF metadata to discover educational resources to download. The problem of downloads is not addressed by Edutella itself: downloads are considered a separate problem, and the structured metadata is statically used for query answering.

In Bibster, the bibliographic data is shared and propagated in the network as in traditional file-sharing applications. The semantic queries in Bibster are supported by two static, built-in ontologies, to which end-users do not contribute.

Chapter 4

A Formal Model for P2P File-Sharing

4.1 Introduction

The solution presented in this thesis is a P2P file-sharing system for structured data, supporting a graph data model.

Our solution is a conceptual extension of the traditional model of file-sharing, implemented over the existing framework of U-P2P (as presented in Section 3.2.3). The purpose of this chapter is to establish a formal basis for this extension, by formalizing the concept of a P2P file-sharing system.

We first informally present the basic principles of file-sharing, made popular by first-generation systems such as Napster or Kazaa, in Section 4.2.

In Sections 4.3 and 4.4 we formalize the different characteristics of these first-generation file-sharing applications.

In Section 4.5 we show how this model can be extended to simultaneously manage multiple communities, as in the original U-P2P prototype.

4.2 Informal Presentation of P2P File-Sharing

Let us first recall informally how peer-to-peer file-sharing typically works.

4.2.1 A Typical File-Sharing Scenario

Chloe is a university student who composes and plays folk music in her free time. She listens to many independent folk artists and spends a lot of time downloading music from the Gnutella P2P file-sharing network, in order to discover new artists.

For this she uses Lemonwire, an application that she downloaded and installed on her home computer in a few minutes.

The interface of Lemonwire provides a special search interface for audio files. As most MP3-formatted music files contain built-in metadata, in the form of so-called "ID3 tags", the search form lets her specify her search criteria according to the ID3 metadata fields, which are artist, title, album, bitrate, genre, track number, and year.

Chloe selects "folk" from the *genre* drop-down menu, enters "192" in the *bitrate* field and submits her query.

Chloe's search message goes out into the Gnutella network. It reaches all the peers that Chloe is directly connected to, who in turn propagate this search message to all their neighbours, decreasing the message time-to-live so that it doesn't circulate for ever in the network.

Each peer reached by the search message searches its public sharing directory for MP3 audio files tagged with the genre "folk", and with a bitrate of at least 192 kbps.

After three network hops Chloe's search message reaches Mark's home computer. Mark is a folk musician whose band has decided to make their music available for free under the Creative Commons license; a few songs are currently in his shared directory, and match Chloe's search criteria.

Mark's P2P client returns a "QueryHit" message listing the songs, with for each file the file name, and the contents of the ID3 tags. The message also references Mark's IP address.

Chloe sees all those songs by a band she has never heard of: they're intriguing.

She clicks on the "download" button next to one of the songs. Her Lemonwire client opens a direct connection to Mark's computer, which stores the file, and the first song is downloaded in a few seconds, because they both have high bandwidth connections.

Chloe listens to the song, and likes it very much. She downloads all the songs by Mark's band and keeps them in her shared directory, so that other people can download them from her as well.

A few weeks later, Chloe searches for more songs by Mark's band and finds that she gets more results than last time. It seems that this band is getting popular. Incidentally, Mark is offline at the moment, so all of these results come from other users!

4.2.2 Characteristics

This simple scenario illustrates a few characteristics of file-sharing systems.

Our first observation, which is at the root of this formalization, is the following: Each user's local repository can be seen as a database with a single table, where each file is listed as a row. File-sharing queries can be seen as simple "select" queries, which return the file name and metadata annotations of entries matching the query.

However, a file-sharing network is very different from a distributed database. We can identify a few key differences:

- The answers to a query depend on the current connections in the network, which are not expected to be permanent : peers may join or leave the network at any time.

- Within a given network, protocols may not allow queries to reach all the connected peers. The above example illustrates this aspect of the Gnutella protocol: query messages have a limited time-to-live, and are only propagated through that many hops in the network.

- Downloads replicate files in the network, creating redundancy;
- The number of instances of a file available in the network can be seen as an indicator of the file's popularity and usefulness.

The formalization presented here is meant to capture these different characteristics, thus establishing a formal *file-sharing* model, different from existing models of distributed databases or other P2P systems (e.g. Distributed Hash Tables).

4.3 File-Sharing Concepts

4.3.1 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 (often large) "binary" element and by several elements of standard data types (character strings, numbers) listing metadata attributes of the file.

Formally, the distinction 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).

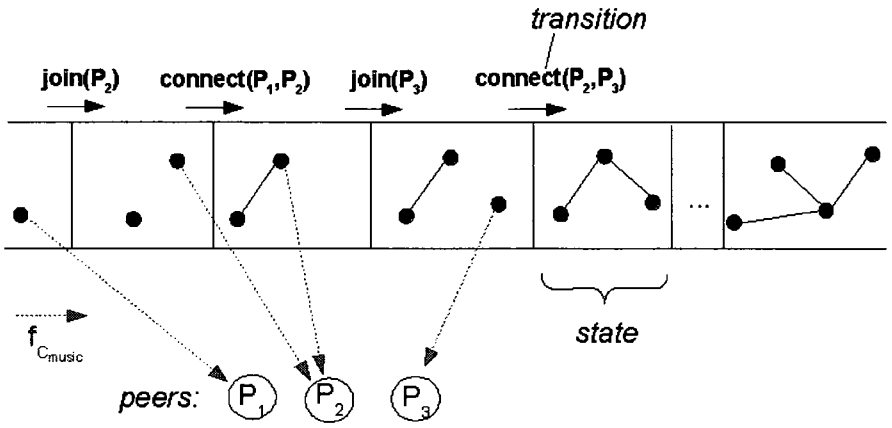


Figure 4.1: Example: states and transitions for example community C_{movies}

4.3.2 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 share *documents* through file-sharing *communities*, which we define in the following.

4.3.3 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.

  State *transitions* are defined in the pseudocode Algorithm 1, and a graphical example showing the states and transitions is shown in Figure 4.1.

[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 exist 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.

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:

  - 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 4.3.1), and *metadata* is any attribute that is not an attachment.

- A *name*, which we will discuss in Section 4.5.

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 4.1 shows an

[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 4.3.4.

extension $R^{p_1}_{Movies}$ stored by a peer p_1. The attribute **trailer** is an attachment and we do not show its values, as they are binary.

- 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**
twotowers	"The Two Towers"	"Peter Jackson"	2002	[...]
fellowship	"The Fellowship of the Ring"	"Peter Jackson"	2001	[...]
titanic	"Titanic"	"James Cameron"	1997	[...]
sleepy	"Sleepy Hollow"	"Tim Burton"	1999	[...]
neverland	"Finding Neverland"	"Marc Forster"	2004	[...]

Table 4.1: Example extension for the community "Movies".

4.3.4 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 [46] algorithm[4].

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 Section 4.5. In the following discussion we will thus assume that this constraint holds.

4.3.5 A Conceptual Model of File-Sharing

To the definitions formulated in the previous sections we add the following relationships between the concepts of file-sharing:

Definition 2 *We 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 *A* document d *is* shared in *the community* C *iff* d *is a tuple of the domain of* R_C, *and there is a peer* p_0 *such that* d *is in* $R_C^{p_0}$.

This conceptual model of a P2P file-sharing system, i.e. the concepts of communities, documents, and peers, and the relations between these concepts, is illustrated by a UML class diagram in Figure 4.2.

As our example related to movies suggests, this model of file-sharing, considering only the subset of concepts Community, Document, and Attribute, can also be used as a *meta-model* to model concepts and entities.

[4]Note that the names used in our examples, such as Table 4.1, are instead reader-friendly names to preserve the clarity of the examples.

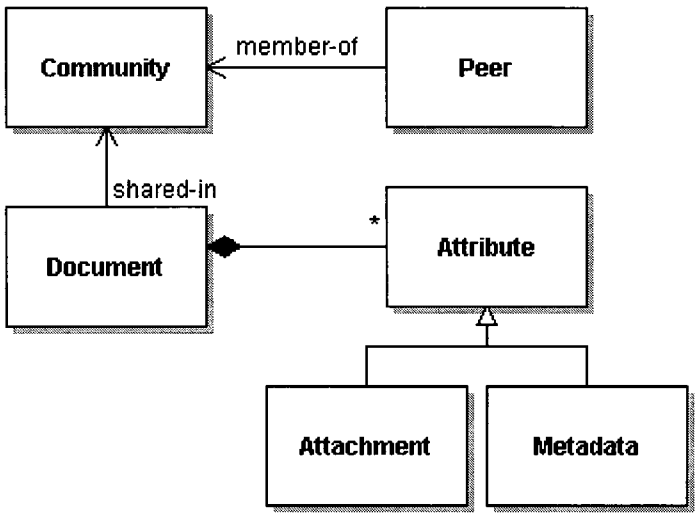


Figure 4.2: Conceptual model of a P2P file-sharing system.

A specific community can be used to model a concept: for example, the community C_{Movies} models the abstract concept of a movie, while each document shared in that community models an instance of this concept, i.e. a particular movie. This aspect of describing concepts and instances is an important step towards representing knowledge.

The main research problem of this thesis can be seen as addressing the next step, of relating entities to one another.

4.4 Queries and Operations

4.4.1 Preliminary Definitions

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, and the symbol $\times$ denotes a cartesian product.

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.*

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

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.

4.4.2 File-Sharing Operations

We define here the data manipulation operations available in a P2P file-sharing community. We first provide a simple definition of these functions and procedures, then a declarative definition of the functions in first-order logic, and finally a specification of the functions and procedures using relational algebra.

Definitions

A peer p in a peer-to-peer system $\mathcal{P}$, *member of* a community C, defines an interface to C with the following 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 logic formula that is matched to the metadata attributes of documents; this function returns the metadata arributes of matching documents, and the identifiers of the peers that store these documents.
- 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 $name_0$.

Declarative Definition of SEARCH and LOOKUP

In the following we will respectively note $m_1, \ldots, m_k$ (resp.$a_1, \ldots, a_j$) the indices of the metadata (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.

The functions SEARCH and LOOKUP can be defined by the following declarative queries:

SEARCH(*expr*):

$$\{(p_i, name_j, x_{m_1}, \ldots x_{m_k}) \mid \exists(x_{a_1}, \ldots, x_{a_j}), R_C^{p_i}(name_j, x_2, \ldots, x_n) \wedge expr(x_{m_1}, \ldots, x_{m_k})\} \tag{4.1}$$

The expression *expr* used as input 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 metadata attributes in R_C.

LOOKUP($name_0, attr$): we note i_{attr} the index of attribute *attr* in the schema of C.

$$\{X \mid \exists(x_2 \dots x_n), R_C^p(name_0, x_2, \dots, x_{i_{attr}}, \dots, x_n)\} \tag{4.2}$$

Relational Algebra Specifications

The formal specifications of these procedures and functions, using relational algebra, are given in Algorithm 2.

Algorithm 2 Data manipulation operations available to a peer p in a P2P system $\mathcal{P}$. (Note: σ and π 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*):
  $R_C^p \leftarrow R_C^p \backslash \{document\}$ ▷ If *document* is not in the local extension R_C^p, then nothing happens.
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

procedure DOWNLOAD($id, name_0$):
Require: $id \in prot_G(p)$
  $\{doc\} := \pi_{R_C}(\sigma_{(peerid=id \wedge name=name_0)}(\Omega(C, G)))$ ▷ This query returns a single document
  PUBLISH(*doc*) ▷ Add document to local extension for the community.
end procedure

function LOOKUP(*document*, *attribute*)
Require: $document \in R_C^p$
  return $\pi_{attribute}(document)$
end function

4.4.3 A Modal Logic View of a File-Sharing Community

We propose an alternative view of a community based on modal logic. The idea of this view is that beyond the tuples that we have described in the previous formal definitions, searches are meant to locate documents. The documents accessible to a peer in a certain community can be seen, in the predicate logic formalism commonly used in database theory, as different assertions, i.e. models for the considered predicate.

We might consider that the peer chooses some of these assertions, and by downloading the corresponding tuples, propagates the view that this assertion is indeed true.

We give here a modal logic characterization of these ideas.

Definitions

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 6 *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 between worlds, in the sense that* (p_1, p_2) *is in the relation iff* $p_2 \in prot_{C,G}(p_1)$*, and* ϕ *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:* $\phi : p \mapsto R_C^p$.

Definition 7 *The modal operator* $\mathbf{V}$ *(for "visible") is then defined as:*
$< M, p > \models \mathbf{V}R_C(\mathbf{doc})$ *iff* $\exists p_1 \in P_{CG}, p_1 \in prot_{C,G}(p) \wedge \mathbf{doc} \in \phi(p_1)$*, where* ***doc*** *is a document* $(x_1 \ldots x_n)$ *of community* C.

We now associate to the input expression *expr* of a SEARCH query a predicate Q_{expr}, that applies to documents, and is true if *expr* is true when its free variables are bound with the metadata of the document:

Definition 8 *The predicate* $Q_{expr}(\mathbf{doc})$ *is true iff* $expr(x_{m_1}, \ldots, x_{m_k})$ *is true.*

Modal Semantics

The documents **doc**, answers to a search query with the input expression *expr*, can be then given by the following modal semantics:

$$\{\mathbf{doc} \mid < M, p > \models \mathbf{V}R_C(\mathbf{doc}) \wedge Q_{expr}(\mathbf{doc})\} \tag{4.3}$$

Similarly, the semantics for downloading a document **doc** can be given by the following pre- and post-conditions:

pre-condition: $< M, p > \models \mathbf{V}R_C(\mathbf{doc})$

post-condition: $I_p \models R_C(\mathbf{doc})$

Note that we also indirectly use the assumption 1 introduced in Section 4.3.4, in the sense that our download operation takes a *name* for input, and our precondition directly associates a single document in the community to this name.

4.5 Multiple Communities

4.5.1 Motivation

We now extend our model to enable the creation of multiple communities within a single application. In a knowledge representation perpective, we see communities as defining concepts, of which instances can be modeled by the documents shared in that community. This extension thus allows a file-sharing system to describe multiple concepts.

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.

4.5.2 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 $C_{Community}$ is the relation $R_{Community}(.,.,.)$ of arity 3, with the following attributes: *name, schema , protocol.*

  The subdomain of each attribute is as follows:

  - *name* is the mandatory meta-data attribute defined in Section 4.3.3 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 4.3.3.

[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.

– 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 4.3.3.

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

Figure 4.3 illustrates the conceptual model of a file-sharing system, augmented with this additional sub-class of documents. This model is implemented by our prototype file-sharing application U-P2P[6].

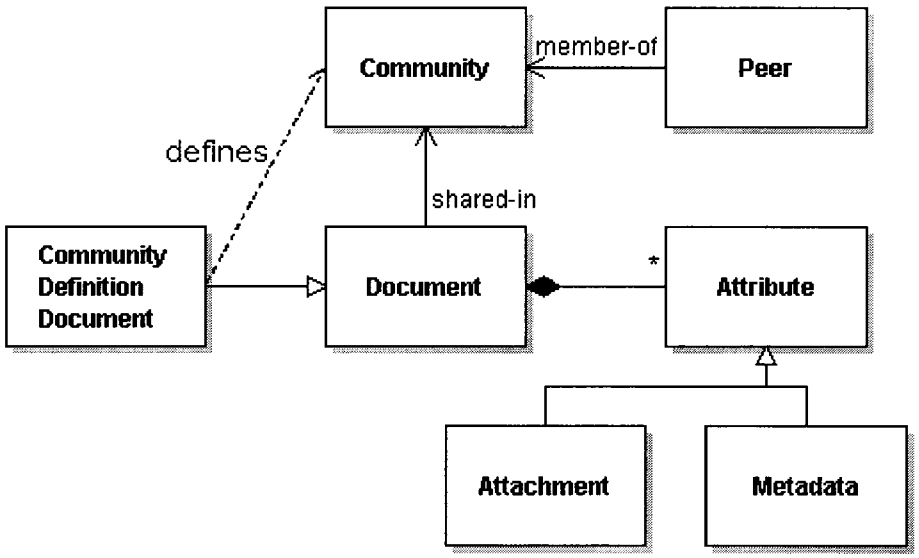


Figure 4.3: Conceptual model of U-P2P.

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

[6]We note that although the model is implemented by versions of the prototype application that existed prior to this work (discussed in 3.2.3), the formal foundations presented here are new contributions of the present thesis.

this $C_{Community}$. $doc_{Community}$ has the following values for the attributes of $R_{Community}$:

- *name*: "Community"[7]
- *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* $C_{Community}$.
- $\forall p, doc_{Community} \in R^{p}_{Community}$.

4.5.3 Semantics of Community Definition Documents

In a knowledge representation perspective, as discussed in Section 4.3.5, we are here using the meta-model of communities and documents to represent a concept, and instances of that concept. The concept modeled by the *Community* community is, precisely, the abstract concept of a file-sharing community as defined in Section 4.3.3.

The foundational property of U-P2P is that this application is programmed to interpret *community definition documents* in order to *join* the communities that these documents define.

The PUBLISH and DOWNLOAD procedures specific to the *Community* community must thus be redefined as shown in algorithm 3.

[7]In order to comply with assumption 1 of Section 4.3.4 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.

Algorithm 3 PUBLISH and DOWNLOAD functions of the interface to $C_{Community}$ defined by peer p

procedure PUBLISH($document_C$):
$R^p_{Community} \leftarrow R^p_{Community} \cup \{document_C\}$
C.JOIN(p) ▷ Where C is the community defined by $document_C$.
end procedure

procedure DOWNLOAD($id, name_0$):
Require: $id \in prot_{C_{Community},G}(p)$
$\{doc_C\} := \pi_{R_{Community}}(\sigma_{(peerid=id \wedge name=name_0)}(\Omega(C,G)))$
PUBLISH(doc_C) ▷ Includes the JOIN defined above
C.CONNECT(p, id)
▷ The peer that the downloaded document comes from provides a connection within the new community.
end procedure

4.5.4 Example

We present here a simple example with three peers and three communities. Using the examples shown in Table 4.1 (we remind the reader that Table 4.1 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 4.2, 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. Each document is represented by its unique name.

4.5.5 Implementation in U-P2P

This feature – the possibility of automatically joining new communities by simple PUBLISH() 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, documents are formatted in XML, the community schemas are stored as XSD (XML Schema Definition) files, and protocols are represented as java classes

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	johnnydepp	"Johnny"	"Depp"	09-06-1963
3	johnnydepp	"Johnny"	"Depp"	09-06-1963
3	elijahwood	"Elijah"	"Wood"	28-01-1981
3	katewinslet	"Kate"	"Winslet"	04-12-1975
2	livtyler	"Liv"	"Tyler"	10-07-1977
3	livtyler	"Liv"	"Tyler"	10-07-1977

Table 4.2: Example content for the communities "Community" and "Actor".

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 [3], 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

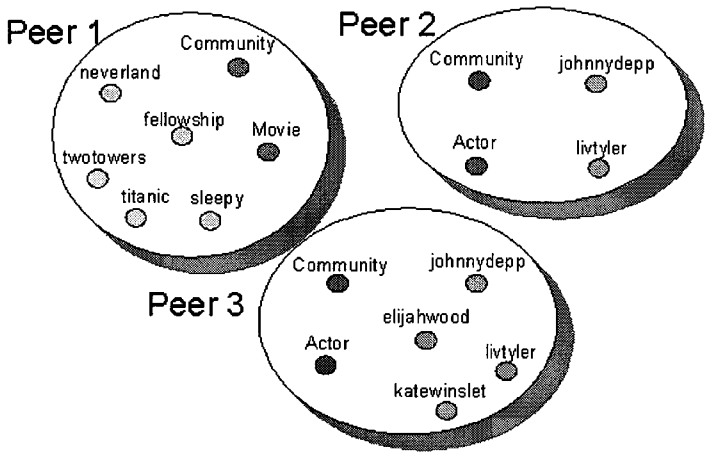


Figure 4.4: Distribution of documents in an example P2P system with three peers and three communities.

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.

Chapter 5

Graph Data in a P2P Network

5.1 Introduction

In Chapter 4 we defined a P2P file-sharing system supporting a data model of structured documents based on arbitrary schemas defined by communities, and presented U-P2P, a prototype application capable of managing multiple communities.

In U-P2P, a peer can be a member of several communities, publish new communities, and discover and join communities created by others. 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.

We present here an extension to U-P2P to support such links, and queries over the resulting graph of documents.

We first analyze the requirements for linking data in a P2P network in Section 5.2. In Section 5.3 we present a naming scheme for U-P2P documents, and its associated dereferencing mechanism.

Finally, we present a class of graph queries supported by U-P2P, in Section 5.4.

5.2 Requirements

5.2.1 Related Work

The design of our graph data model is based on the analysis of related work. We recall here the principles of Freebase and the Linked Data On the Web initiative, already described in Sections 3.2.1 and 3.2.3, respectively.

Freebase

Freebase [15] is meant to be a collectively built and edited "database about everything". In Freebase, a concept is described by a table listing instances of this concept and their attributes. Each instance (a "topic" in Freebase terminology) is a row, and the attributes of this topic may be references to other topics. Attributes may be multi-valued.

For example, movie actors are described in a table with columns listing their name, date of birth (and death, if relevant), nationality, and a list of movies they have played in. This list of movies is precisely a list of references to movies listed in the Freebase table listing movies. Each movie in that table is in turn described by its title, year, director, screenplay author, etc.

Freebase data forms a graph of interconnected resources: one can view an actor, browse to a film that the actor played in, then browse to the director of that film, and so on, navigating the graph. The links to freebase topics are supported by GUID (Globally Unique Identifiers), similar to hash-generated keys.

The Linked Data On The Web (LDOW) initiative [47] aims to provide a framework for organizations to publish RDF data in Web-Accessible repositories, and interconnect these repositories.

Organizations participating in this initiative name the resources described by their data using HTTP URIs (Uniform Resource Identifiers), and publish RDF statements

linking these resources to resources described in other data collections. Such connections make the global graph of linked data *connected.*

The LDOW approach is supported by the following "good practices" [47] promoted by the W3C:

- The resources should be named by URIs;
- The URIs should be HTTP URIs, so that they can be dereferenced;
- The dereferencing of URIs should return useful information using the W3C standards (RDF, SPARQL);
- The information should include links to other URIs, in order to connect the graph.

Analysis

The Linked Data On The Web initiative is based on the RDF data model, which is inherently a graph data model. Navigating the graph in a distributed setting relies on well established naming and dereferencing mechanisms associated with the HTTP protocol.

The data model of a Freebase table taken in isolation matches the schema structure that we have defined for a file-sharing community (see Section 4.3.3).

The identifiers of Freebase "topics", that are used to reference one topic from another, are hash-generated unique keys. Freebase shows how this data model associated with a naming scheme supports a graph data model that could nicely extend the basic U-P2P model.

However, U-P2P documents are distributed in a P2P file-sharing network, which makes the U-P2P setting very different from the Freebase setting, which is centralized, or from the Linked Data on the Web setting, where the different data providers are static web hosts (i.e. permanently connected hosts).

Current peer-to-peer file sharing systems do not offer a natural way of linking files or data items to one another. In addition, the HTTP addressing scheme, which is location-based, is well suited to documents statically served by permanently connected hosts, whereas our context assumes replicated documents and a potentially high churn in peers. We see these characteristics both as an obstacle and an opportunity: we cannot assume that peers will be permanently connected or will always provide the same content, but the high replication of documents means that the more popular documents will often, if not always, be available in the network.

5.2.2 Requirements for a Graph Data Model in a P2P File-Sharing Network.

Based on these observations, we identify the following requirements for linking documents in U-P2P :

- The main requirement for linking documents is a *naming scheme* and an associated dereferencing protocol;
- The naming scheme should follow the generic syntax of a URI [5] scheme;
- This naming scheme should not assume a single static location for a document;
- The dereferencing protocol should take document replication into account, for example multiple copies of a given document should have the same URI.

5.3 URIs and Endpoints

5.3.1 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*[1], 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`.

5.3.2 Endpoint Attributes

In order to make use of this URI in our formal 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 9 *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.

Our model extended to include Endpoint attributes is illustrated in Figure 5.1.

[1]based on the assumption 1 introduced in Section 4.3.4

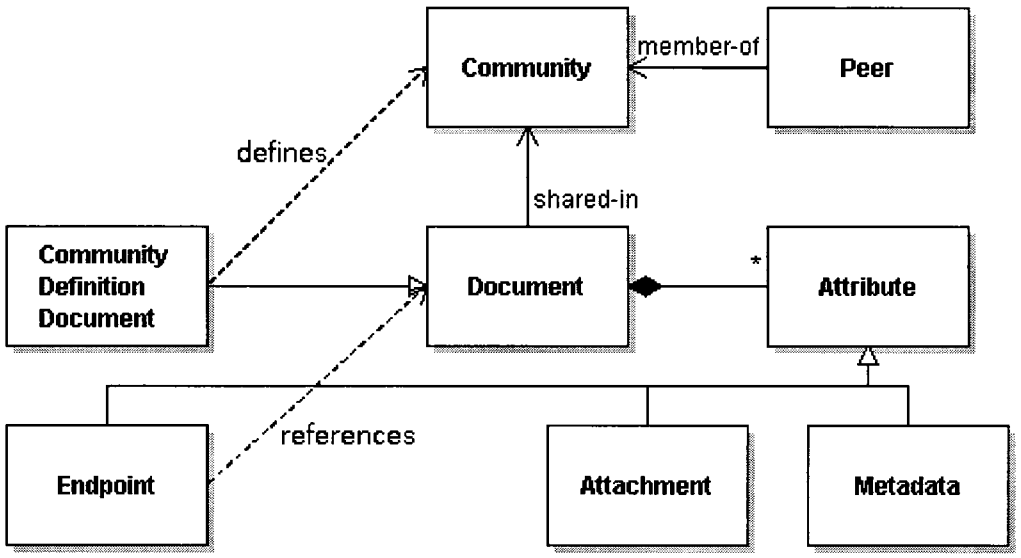


Figure 5.1: Endpoints in the U-P2P Meta-Model.

This model, or *meta-model* (see discussion in Section 4.3.5), now has a cycle, from the concept of a document, to the concept of an attribute, to the sub-concept of an endpoint attribute, back to the concept of a document via the "references" relationship. This cycle allows connections between documents and induces the graph data model.

5.3.3 Dereferencing Protocol

Dereferencing a URI, i.e. obtaining a document from its URI, can be done using only the basic P2P File-Sharing operations *search*, *download*, *publish*, *remove*[2] defined in Section 4.4.

The procedure is given by the pseudocode Algorithm 4.

The principle of this dereferencing mechanism is that the community where the document is shared is identified in the URI. In order to retrieve that document, the peer must be a member of the community, or become a member by dowloading the

[2]We note that in practice only search and download are used.

Algorithm 4 URI dereferencing algorithm

```
The peer performing the dereferencing is denoted by p.
function DEREFERENCE(up2p:id1/id2):
    if p not member-of id1 then
        {(p_i, id1)}_{i=1...m} ← Community.SEARCH(communityId=id1)
        j ← select in [1...m]                                  ▷ select a peer
        Community.DOWNLOAD(p_j , id1);
    end if
    {(p_i, id2)}_{i=1...w} ← id1.SEARCH(documentId=id2);
    k ← select in [1...w]                                      ▷ select a peer
    id1.DOWNLOAD(p_k , id2);
    return doc_id2
end function
```

community definition document from an existing member (see Section 4.5.3).

The relevant community must then be searched for the appropriate document. This search-based dereferencing mechanism may imply a high cost, but the searches are limited to a community, which reduces the search space as compared to the global network.

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.

5.3.4 Comparison with RDF

We compare here the graph data model induced by our documents and links, to the RDF model, which is the W3C standard for the Semantic Web, and is presented in Chapter 2.

RDF Over U-P2P

The purpose of RDF is to annotate resources (identified by URIs), relating them to simple annotations (literals) or to other resources. The latter type of annotations are called RDF links, and induce a graph of resources.

In U-P2P, the documents we share can be identified by URIs, and we can thus annotate them in RDF. In addition, these RDF statements could be serialized as XML documents which could themselves be shared in U-P2P, in a community with the appropriate XML schema.

We apply this approach to an example RDF statement of Section 2.1.3, the RDF triple (http://ex.org/doc/123; http://ex.org/creator; http://ex.org/person/John).

Here, we assume the document identified by "http://ex.org/doc/123" could be shared in the community "exampleDoc" and has the unique name "123" within this community. The person referenced by the URI John could also be described by a Virtual business card in the community "vcard" and have the unique name "John" in this community. Finally, the property "http://ex.org/creator" would also require a document to uniquely describe this property, and this document could be shared in the community "property", with the unique name "creator".

The RDF triple with the appropriate UP2P URIs would then be: (up2p:exampledoc/123 ; up2p:property/creator ; up2p:vcard/John). This RDF triple could be serialized as an XML document (shown in Table 5.1) and shared in a U-P2P community of RDF statements.

In this approach, the triple itself is shared in U-P2P, in addition to the resources linked by this triple, which are also found in the network. The triple will thus have a unique name in the community of RDF triples, and could be annotated by other RDF statements.

```
<?xml version="1.0"?>
<rdf:RDF>
<rdf:Description rdf:about="up2p:exampledoc/123">
<up2p:property/creator>up2p:vcard/John</up2p:property/creator>
</rdf:Description>
</rdf:RDF>
```

Table 5.1: Example RDF statement serialized in XML with U-P2P URIs.

However, the standard approach to annotating statements is to reify these statements, i.e. to make a resource out of the statement and give it a URI, using a different syntax. In U-P2P, this step is not really necessary, as serializing the statement to an XML document is sufficient: any XML document shared in U-P2P is automatically given an identifier, and is thus a resource that can be annotated just like any other document. Here, the identifier of the above document could be "statement001", for example[3].

Assuming the above statement is shared in community "RDFstatements", its reified form (serialized in XML) would be as shown in Table 5.2.

```
<?xml version="1.0"?>
<rdf:RDF>
<rdf:Statement rdf:about="up2p:RDFstatements/statement001">
<rdf:Subject>up2p:exampledoc/123</rdf:Subject>
<rdf:Property>up2p:property/creator</rdf:Property>
<rdf:Object>up2p:vcard/John</rdf:Object>
</rdf:Statement>
</rdf:RDF>
```

Table 5.2: Example reified RDF statement.

The syntax for reified statements has the advantage of using a simple, fixed schema: only three attributes rdf:Subject, rdf:Property and rdf:Object. In U-P2P,

[3]We remind the reader that in U-P2P, this identifier would normally be produced by a hash function.

we can thus create a community with this schema to share any RDF statements. In this schema, all three attributes are endpoint attributes pointing to other U-P2P documents.

The identifier of the statement itself (up2p:RDFstatements/statement001) is ambiguous here: according to RDF conventions it identifies the abstract resource constituted by the triple (up2p:exampledoc/123 ; up2p:property/creator ; up2p:vcard/John), which can be either serialized as shown in Table 5.1, or else reified then serialized, as in Table 5.2.

In U-P2P, we would obtain two different XML documents, which would have different identifiers.

General U-P2P Graph Data Model

As we have shown above, the simple fact of having a URI scheme for U-P2P documents makes it possible to use RDF above U-P2P.

However, in the RDF approach, all the annotations of the resources are stored in separate RDF documents, and these RDF documents describe a graph that can be viewed and queried without requiring any access to the resources themselves. The resources and the annotations, while both could be shared in U-P2P, are completely independent.

In our case of interest, the resources have some structure, and metadata attributes that can be queried; but in the standard RDF approach, these attributes should be represented as RDF annotations: the attribute name would be a property, and the attribute value would be a literal stored in the object of the RDF statement.

This comparison suggests a direct mapping of RDF constructs to the general U-P2P model, rather than the usage of RDF on top of U-P2P. U-P2P attributes map to RDF properties; these RDF properties take literal values in the case of U-P2P metadata attributes, and are links (statements where the Object is a URI) in the

case of U-P2P endpoints. U-P2P attachments would not normally be represented in RDF: rather the attachments would be resources to be named and annotated.

To go farther, the U-P2P meta-model can be seen as an equivalent of the RDF class formalism: U-P2P communities are similar to RDF classes, and the Community community corresponds to the class `rdf:class`.

In fact, the U-P2P approach is more similar to the Freebase approach, where each tuple of attributes constitutes a resource in itself. The graph is constituted by the resources themselves, which are tuples including links.

Our graph data model, as induced by resources and their endpoint attributes can be represented by the UML object diagram shown in Figure 5.2. The two documents "document 1" and "document 2" are related by an edge materialized by the attribute "Endpoint1".

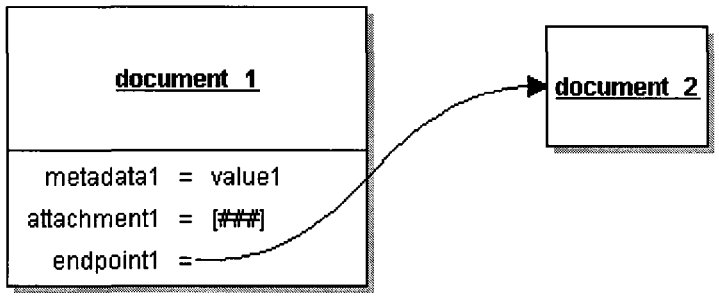


Figure 5.2: Graph of data induced by endpoint attributes.

5.3.5 Example

In our running example about movies and actors, we could connect actors to the movies that they play in by referencing movies in a "filmography" attribute of *Actors* documents. In other words, *filmography* would then be a multi-valued endpoint attribute in the schema of the *Actors* community. The community schema would then be : $(name, bio, photo, filmography^{+})$, the + indicating that the attribute is

multi-valued. The document representing Liv Tyler could then be : *('Liv Tyler', 'Liv Tyler is the daughter of... ', [lt.jpg],* `up2p:Cinema/twotowers`*)*

An example graph of documents can be represented by the UML object diagram shown in Figure 5.3. We use the meta-model view discussed in Section 4.3.5: here, we see the communities *Actor* and *Film* as classes, and the documents shared in those communities as objects of those classes.

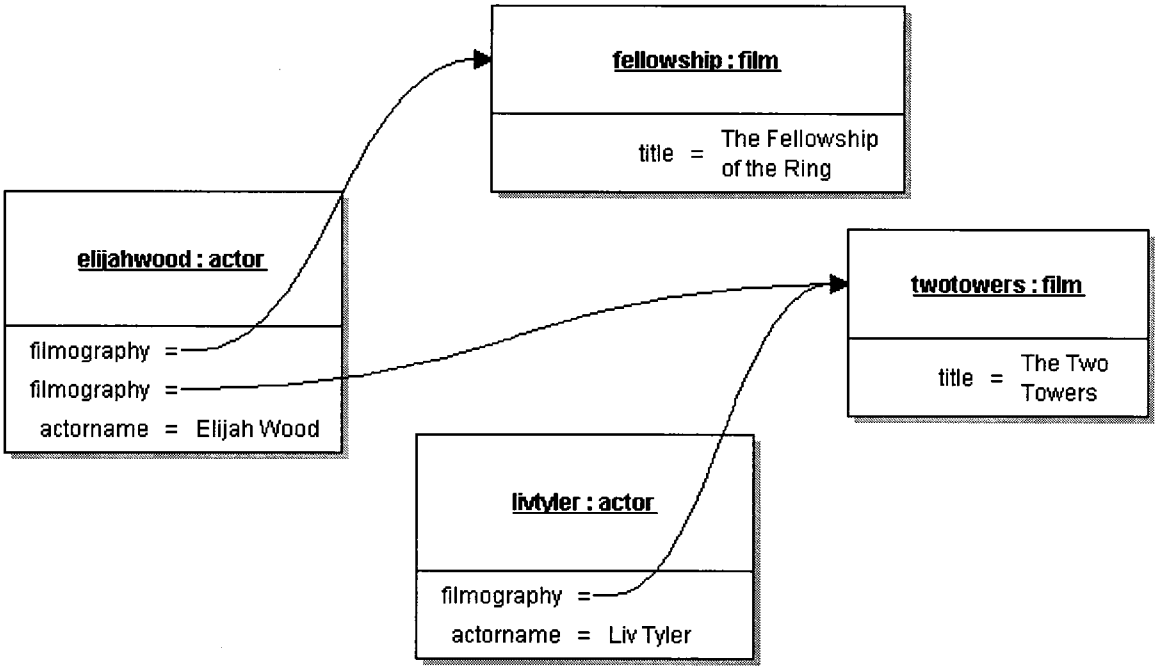


Figure 5.3: An Example graph of related U-P2P documents.

5.4 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.

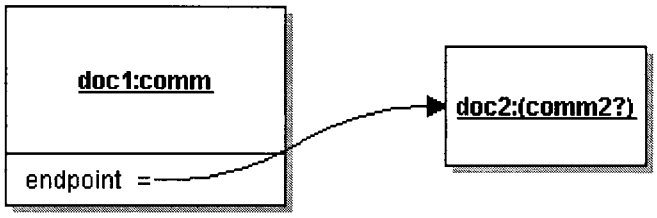


Figure 5.4: Graph pattern represented by the predicate $edge(doc_1, comm, endpoint, doc_2)$

The simplest patterns 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.

5.4.1 Edge Queries

Edge Patterns

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

Definition 10 *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*.*

This predicate represents a simple graph pattern, illustrated in Figure 5.4.

Edge Pattern Queries

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.

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. This definition is thus asymetric, reflecting the asymetric representation of an edge in our graph data model, which is represented as an attribute in one of the two nodes that the edge relates.

We present here simple edge pattern queries in a declarative form. The lower-case names denote constants, and the upper-case X and Y denote variables.

Closed formulas (i.e. formulas with no free variables) define boolean queries:

$$edge(doc_1, comm, endpoint, doc_2) \tag{5.1}$$

$$\exists Y_1, \exists Y_2 edge(Y_1, comm, endpoint, Y_2) \tag{5.2}$$

The intuitive meaning of the above queries can be respectively interpreted as : "does doc_1 in community *comm* reference doc_2 through the attribute *endpoint*?", and "does any document of community *comm* reference another document through the attribute *endpoint* ?"

Open formulas, i.e. formulas with one or several free variables, define queries where the answers of the query are bindings for the free variables of the query.

The "open" queries that can be defined using the predicate *edge* are the following:

- A query with two free variables:

$$\{X, Y \mid edge(X, comm, endpoint, Y)\} \tag{5.3}$$

  This query can be interpreted as: "find all documents in community *comm* referencing another document through the attribute *endpoint*, and in each case return both the referencing and the referenced documents".

- Queries with one free variable, and constants:

$$\{X \mid edge(doc_1, comm, endpoint, X)\} \tag{5.4}$$

$$\{X \mid edge(X, comm, endpoint, doc_2)\} \tag{5.5}$$

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*"

- Finally, queries with one free variable, and one bound variable:

$$\{X \mid \exists Y, edge(Y, comm, endpoint, X)\} \tag{5.6}$$

$$\{X \mid \exists Y, edge(X, comm, endpoint, Y)\} \tag{5.7}$$

These queries can respectively be interpreted as "find all documents referenced by some document of community *comm* through the attribute *endpoint.*", and "find all documents in community *comm* referencing another document through attribute *endpoint*".

Edge Pattern Query Answering

Queries with a single free variable return documents, and thus extend the functionality of simple SEARCH queries, which is a fundamental part of file-sharing. 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.

We first discuss the case of queries 5.4 and 5.5, which do not have bound variables.

In Algorithms 5 and 6, we detail the query answering procedures for queries 5.4 and 5.5, using the basic P2P File-Sharing operations *search*, *download*, and *lookup*.

We assume that *doc* is given as a URI.

Algorithm 5 Algorithm for answering query 5.4.

```
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 6 Algorithm for answering query 5.5.

```
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
```

From these two base algorithms we can derive algorithms to answer each of the other queries listed in 5.4.1, by the following variations:

- In the case of query 5.7, Algorithm 6 can be used, with a wildcard condition in the final search function call, as follows (we only reproduce the relevant line, and use the asterisk to denote a wildcard):

$$\text{results} \leftarrow comm.\text{SEARCH}(endpoint = *)$$

- For query 5.6, we reformulate the query as follows:

$$\{X \mid \exists Y, edge(Y, comm, endpoint, X)\} = \{X \mid \exists Y, \exists Z, \quad edge(Y, comm, endpoint, X) \wedge \quad edge(Y, comm, endpoint, Z)\}$$

We now obtain a complex query, which we will address in the following section. We give a brief informal view of how this query can be answered: We first observe that in the predicate $edge(Y, comm, endpoint, Z)$, taken first, Z is existentially quantified, and Y is the target of the subquery, which makes this subquery of the same form as query 5.7. Intuitively, the solution we use here is to collect through this subquery all "candidate" documents for Y, then to use each of these candidates as a constant in the other subquery, which then becomes a query of the form 5.5.

- The boolean answer to query 5.2 can be derived from the answers to query 5.7: if the set of answers to 5.7 is empty, then the answer to 5.2 is false, if the set is not empty, then it is true[4].

- The boolean answer to query 5.1 can be derived from the query 5.5, applied to doc_1: if the answer set contains doc_2, then the answer to query 5.1 is true, and it is false otherwise.

- Query 5.3, which returns pairs of documents, can be answered by rewriting the query as for query 5.6. As for query 5.6, the subquery $edge(Y, comm, endpoint, Z)$ must be executed first; and for each document Y_i answer of this subquery, the subquery $edge(Y_i, comm, endpoint, X)$ must be executed as query 5.5, with Y_i (now seen as a constant term) in place of *doc*. For all the answers X_{ij} of this subquery, the pair (X_{ij}, Y_i) is an answer to the original query 5.3, i.e. a binding for X and Y.

5.4.2 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.

We first present the general declarative form of this class of queries, then provide an algorithm for answering a query of this class.

[4]Note that query 5.6 could be used in the exact same way.

Preliminaries

We will define the class of *path pattern* graph queries based on the elementary *edge* queries defined above.

As we have shown in Section 5.4.1, simple "edge" pattern queries can be reduced to two main forms (namely queries 5.4 and 5.5).

Intuitively, the query is a pattern that matches directed edges between two documents, and for each matching edge, query 5.4 returns the document at the *start* of the edge, whereas query 5.5 returns the document at the *end* of the edge.

We would like a single generic way of expressing both types of queries. For this purpose 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:

$$edge_l(X, c, e, Y) =_{def} edge(X, c, e, Y) \tag{5.8}$$

$$edge_r(X, c, e, Y) =_{def} edge(Y, c, e, X) \tag{5.9}$$

We can now rewrite our queries 5.4 and 5.5 as follows:

$$\{X \mid edge(doc, comm, endpoint, X)\} = \{X \mid edge_r(X, comm, endpoint, doc)\} \tag{5.10}$$

$$\{X \mid edge(X, comm, endpoint, doc)\} = \{X \mid edge_l(X, comm, endpoint, doc)\} \tag{5.11}$$

In order to answer queries 5.10 and 5.11 we now introduce an additional convenience function that takes the parameter *dir* as an argument, and redirects to the previously defined query answering Algorithms 5 and 6.

Algorithm 7 Path Pattern Query Answering Algorithm

```
function AnswerQuery0(dir, comm, endpoint, doc)
    if dir = l then
        return AnswerQuery1(comm, endpoint, doc)
    else if dir = r then
        return AnswerQuery2(doc,comm, endpoint)
    end if
end function
```

Path Pattern Graph Queries

Definition 11 *The class of* path pattern queries *is defined by the following general declarative form :*

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

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 5.10 and 5.11);*
- $(comm_0,\ldots,comm_n)$ *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*. A simple extension of this definition would allow this input

document *doc* to be the result of an additional sub-query similar to a basic SEARCH.

In order to include this extension, we define the class of *extended path pattern queries*, based on the previous definition 11:

Definition 12 Extended path pattern queries *are queries of the general declarative form defined in definition 11, where the formula* $edge_{b_n}(Y_n, comm_n, endpoint_n, doc)$ *is replaced by a conjunction of predicates of the following form:*

$$(Y_n = (y_1, \ldots, y_n)) \wedge R_{comm_n}(y_1, \ldots, y_n) \wedge expr(x_{m_1}, \ldots, x_{m_k})$$

where $expr(x_{m_1}, \ldots, x_{m_k})$ *is a boolean search condition on the metadata of document* Y_n.

Example Based on the example of Section 5.3.5, involving Movies and Actors, we give the declarative form of two example path pattern graph queries.

Example Query 5.12 : Films that an actor who has played in the film titled "The Two Towers" has played in:

$$\begin{aligned}\{X \mid \exists Y_0, \exists Y_1, \quad & edge_r(X, actor, filmography, Y_0) \\ \wedge \quad & edge_l(Y_0, actor, filmography, Y_1) \\ \wedge \quad & Y_1 = (y_1, \ldots y_n) \\ \wedge \quad & R_{film}(y_1, \ldots y_n) \\ \wedge \quad & y_{i_{title}} = \text{“}TheTwoTowers\text{”}\}\end{aligned}$$

(i_{title} denotes the position of the attribute "title" in R_{film})

The graph pattern represented by this query is represented in Figure 5.5.

Example Query 5.13: Actors who have at most three degrees of separation from

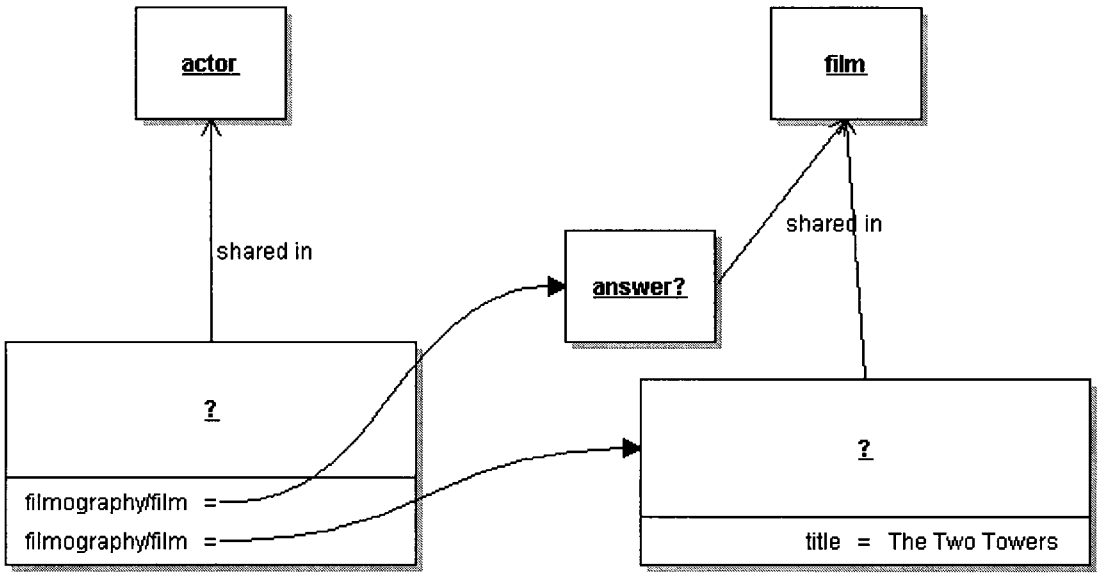


Figure 5.5: Graph pattern represented by example query 5.12.

Kevin Bacon[5]:

$$\begin{aligned}\{X \mid \exists Y_0, \ldots \exists Y_4, \quad & edge_l(X, actor, filmography, Y_0) \\ \wedge \quad & edge_r(Y_0, actor, filmography, Y_1) \\ \wedge \quad & edge_l(Y_1, actor, filmography, Y_2) \\ \wedge \quad & edge_r(Y_2, actor, filmography, Y_3) \\ \wedge \quad & edge_l(Y_3, actor, filmography, Y_4) \\ \wedge \quad & edge_r(Y_4, actor, filmography, \text{up2p:movies/kbacon})\end{aligned}$$

(We assume the documentId of the document describing Kevin Bacon is up2p:movies/kbacon)

[5] "degrees of separation" as in the popular "Kevin Bacon game" : An actor is one step (or degree of separation) away from each actor that they have co-starred in a movie with, two steps away from each actor who has co-starred with their own co-stars, etc.

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 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 answer to this subquery gives constants that are input to the subquery associated with the index $n - 1$. The new query is now shortened by one term, and this process can be repeated recursively until it is only a simple edge query: The answers to this last subquery give the final answers to the query.

Each of the subqueries associated with the index values $i = 1 \ldots n$ is answered using Algorithm 7.

The full query answering algorithm is listed in Algorithm 8.

Algorithm 8 Path Pattern Graph Query Answering Algorithm

function ANSWERPATHQUERY($[b_0, \ldots, b_n]$, $[comm_0, \ldots, comm_n]$, $[endpoint_0, \ldots, endpoint_n]$, *doc*)
  $Y_{list} \leftarrow$ ANSWERQUERY0(b_n, $comm_n$, $endpoint_n$, *doc*)
  if $n = 0$ **then**
    return Y_{list}
  else
    for all $Y \in Y_{list}$ **do**
  $answer_{list} \leftarrow answer_{list} \cup$ ANSWERPATHQUERY($[b_0, \ldots, b_{n-1}]$, $[comm_0, \ldots, comm_{n-1}]$, $[endpoint_0, \ldots, endpoint_{n-1}]$, Y)
    end for
    return $answer_{list}$
  end if
end function

5.4.3 Discussion

The class of graph pattern queries presented here is quite restricted. As we have discussed in Section 5.3.4, our graph data model is comparable to RDF. If we used exactly RDF, we would attempt to adapt SPARQL exactly as a query language over our data. However, for our more general data model, of documents related by links, a new language was required.

We believe that a language of comparable expressiveness to SPARQL could be defined, and the constructs presented here are a limited subset of such a language.

The main limitations of our language, as compared with SPARQL (or its hypothetical equivalent for our data model), are the following:

- edge "labels" cannot be used as query variables;
- we do not support queries with multiple free variables;
- queries on documents must be made in the context of specific communities;
- the graph patterns supported by our query languages are limited (only paths).

We believe that most of these limitations are not inherent to the data model. Support for graph queries in U-P2P is a proof-of-concept feature : extending the language (and supporting query execution) requires query planning and optimization, which is a problem out of the scope of this thesis.

We may observe, however, that queries with a single free variable, where the results are documents, are consistent with the basic paradigm of file-sharing, where users find, then download documents, as opposed to the expressive queries required by full-fledged database systems or the fine-grained approach of SPARQL.

5.5 Query Reuse

In this thesis, we have enhanced the functionalities of traditional file-sharing, by using a general approach where useful concepts specific to File-Sharing (and to our proposed enhancements), such as communities or endpoints, are described in documents and shared along with other documents.

Here, we apply the same approach to sharing *queries*. Reusing queries allows non-technical users to benefit from complex queries created by advanced users, better capable of expressing their information needs.

This feature is similar to the principle of sharing *pipes* on the social web site Yahoo Pipes[6].

We first discuss the requirements for making queries reusable. Then, following the approach of Section 4.5, we define a *Community of Queries* to describe queries in structured documents, and finally we add the concept of a query to our model of P2P file-sharing.

5.5.1 Requirements

If we consider the examples of queries given in Section 5.4.2, we can note that the information needs expressed by these queries are very specific, and are probably of little utility to most users of a system. On the other hand, a slightly abstract version of these queries could be of much higher utility: for example, instead of providing a query finding actors who have co-starred in a film with Elijah Wood, users could be more interested in a generic query, that, given an actor, will return the co-stars of this actor.

In addition, considering the longer-term goals of 'piping' and recombining queries to build more complex queries, it is a desirable feature that the standard output of a

[6]http://pipes.yahoo.com/, also see Section 3.3.2 for a review.

query be reusable as input to another query.

For these reasons, we will consider the class of *Path Pattern Queries* defined in Definition 11, where the input document *doc* is not part of the query definition, but a parameter to be provided for any query execution. However, we see it as useful to indicate in the query definition what type of input, in terms of community, the query expects. In our above example, the abstract query returning co-stars of an actor should be given a document of the community C_{Actor} as input.

5.5.2 Query Definition Documents

We introduce the Community of Queries, C_{Query}, modeling the concept of a Path Pattern Query. The schema of the community must include an encoding of the declarative form of the query, and we add some additional attributes to facilitate the discovery and reuse of these queries.

We thus define the schema R_{Query} with the following attributes:

- name;
- description;
- a reference to the community that this query applies to;
- a representation of the declarative query.

For our prototype implementation U-P2P, where documents are structured in XML, we give the tree schema of C_{Query}, illustrated in Figure 5.6.

5.5.3 Extended Model of P2P File-Sharing

We add the concept of a query to our model of P2P File-Sharing, and show how queries are modeled by Query Definition Documents. The extended model is illustrated in Figure 5.7.

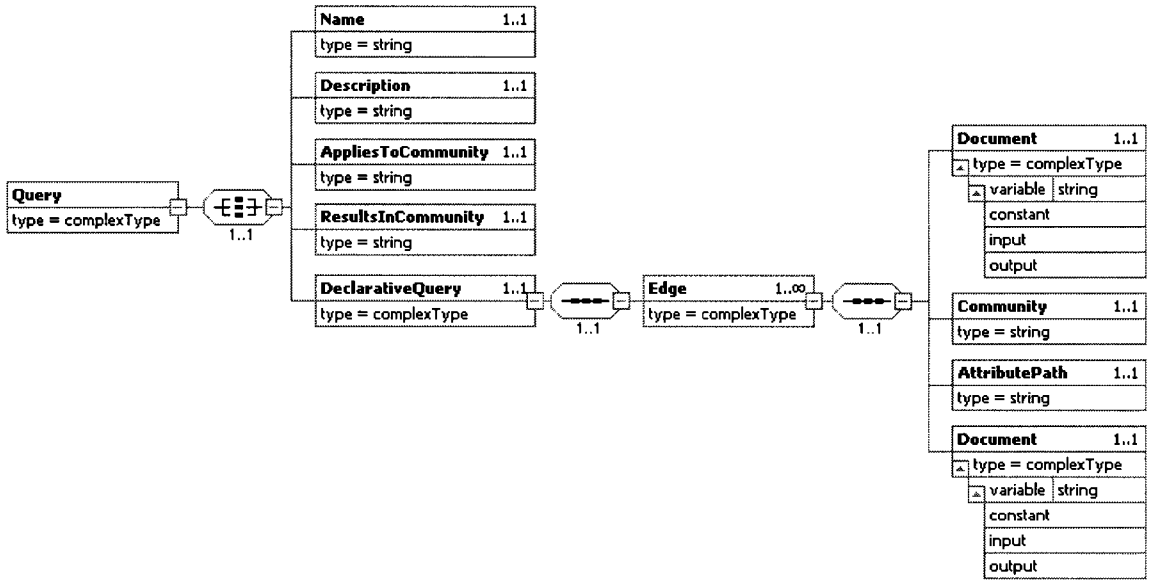


Figure 5.6: XML Schema for Query Definition Documents

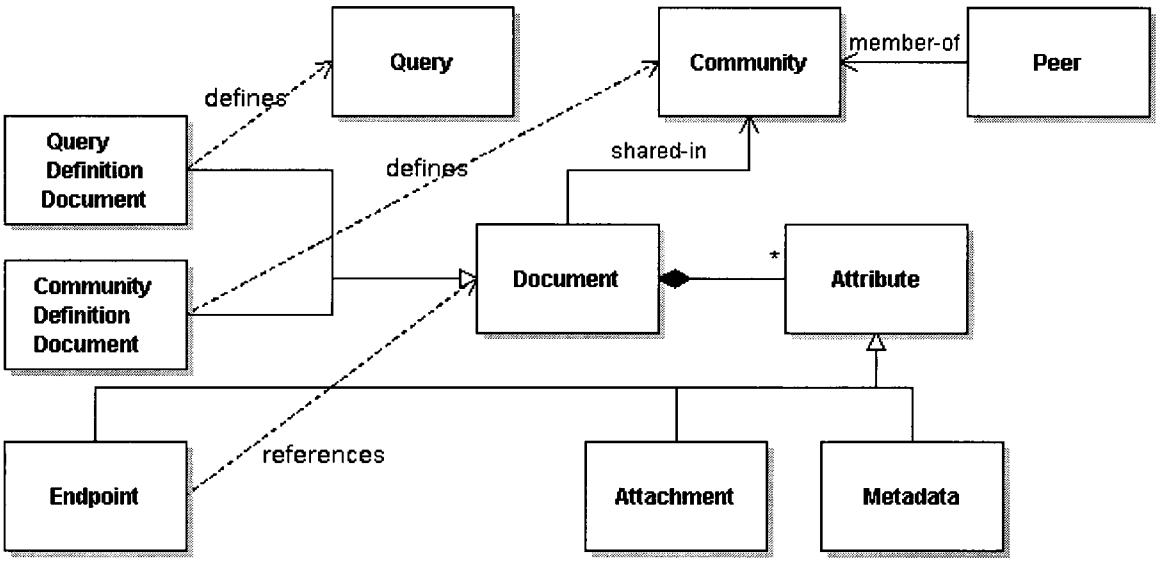


Figure 5.7: Extended model of P2P file-sharing, including queries.

Chapter 6

Extensible Query Processing Based on a Tuple-Space Design

In this chapter we describe the design of our prototype P2P file-sharing system U-P2P, and show how the different functionalities described in the previous chapters are implemented.

In Section 6.1 we present our initial requirements and general tuple-space based design.

In Section 6.2 we describe how the basic P2P file-sharing operations are supported by this design.

In Section 6.3 we show how path pattern graph queries (as defined in Section 5.4) can be processed by additional agents interacting with the tuple-space.

6.1 General Tuple Space Design

6.1.1 Pre-existing P2P File-Sharing Prototype

As we have mentioned before, this work builds on a previous version of U-P2P, our prototype file-sharing system, that supports the multiple community model presented in Chapter 4. This previous version of U-P2P was presented in [3], [48] and [49], and

we briefly present here its design.

The high-level design of U-P2P comprises four main components implementing the main functionalities of the P2P file-sharing client:

- The local Repository stores documents of the communities that the peer is member of, and responds to searches and retrieval queries (documents or document attribute values).

- A Network Adapter propagates the queries to other peers reachable through community protocols, and also handles queries originating from other peers (to be evaluated against the local repository). The Network Adapter also performs Downloads (retrieves documents stored by other peers).

- A user interface collects input from the user and submits requests to the Repository and Network Adapter. When the user performs searches, the User Interface collects and stores a temporary collection of Responses, which are presented to the user.

- A central "Web Adapter" implements the logic of P2P file-sharing, delegating specific tasks to the other components.

All these components are strongly coupled and feed requests into one another, accepting requests from the user interface or the network. Our current work aims to use the file-sharing operations (publish, remove, search, download) as "building blocks" to implement extended functionalities, such as expressive queries.

For this purpose, our design aims to remove the central logic component, and decouple the three other functional components : network adapter, repository, and user interface. These components should provide atomic "services" that can be invoked asynchronously, and in arbitrary sequence, via a coordination component.

6.1.2 Tuple Space Design

Requirements

The primary requirement for our design is to reuse and decouple the existing components of U-P2P, and to introduce a flexible coordination component in order to support new complex processing. In addition, our longer-term goal of managing *knowledge*, envisions peers forming their knowledge base by downloading documents expressing intensional and extensional knowledge, then performing ad-hoc reasoning based on whatever knowledge happens to be in their local repository, and reachable in their network neighbourhood, at a given time.

The Linda Tuple Space Model

A computation model well suited for such ad-hoc processing is the "Blackboard Model", where a central shared memory space is shared between several agents, which execute often rule-based behavior, triggered by elements appearing in the shared space.

The main architecture implementing this model is a tuple space, first proposed by Gelernter and Carriero, who designed the Linda coordination language [50]. We briefly present the concept of a Linda tuple space.

In a Linda tuple space, the data items found in the space are *tuples*, that agents can input and output by three basic operations: *in, out, and read.* The operations *in* and *read* consist in matching tuples to a template: *in* removes the matching tuples and returns them, whereas *read* returns a copy of the tuples, leaving a copy in the tuple space.

A tuple is formed of any number of fields, which may be *formal* or *literal.* A formal field is a typed field which has no value but which matches any field of that type, and a literal field is a typed field with a precise value, and it only matches another field

with the same type and value.

In and *read* operations use these field matching rules, to match tuples: a template matches another tuple if all their fields match pair-wise (i.e. the i^{th} field of the template matches the i^{th} field of the other tuple). In a tuple matching operation, only the template may have formal fields.

A Tuple Space Based Design

The principle of a tuple space to support reasoning for the Semantic Web has been suggested by several authors [51, 52].

We have thus based our design on a tuple-space architecture, where the three major components of our application — user interface, local database, and network adapter — interact to perform these basic operations, and where additional agents can seamlessly be integrated to perform additional processing.

Each of these three components inputs and outputs tuples to the tuple-space via a *proxy agent*, which manages a set of templates specific to the component. Each time that a tuple in the tuple-space (encoding an atomic processing request) matches a template of the agent, the tuple is retrieved and processed by the agent. Responses, if any, are output back into the tuple-space and may trigger new processing by other agents.

The architecture of U-P2P is illustrated in Figure 6.1.

6.1.3 Agents

The proxy agents that interact with the tuple space are designed to support rule-based processing, based on a set of templates. The back-end component behind each proxy agent implements a number of simple functionalities, and the processing of each function is triggered by tuples appearing in the tuple-space.

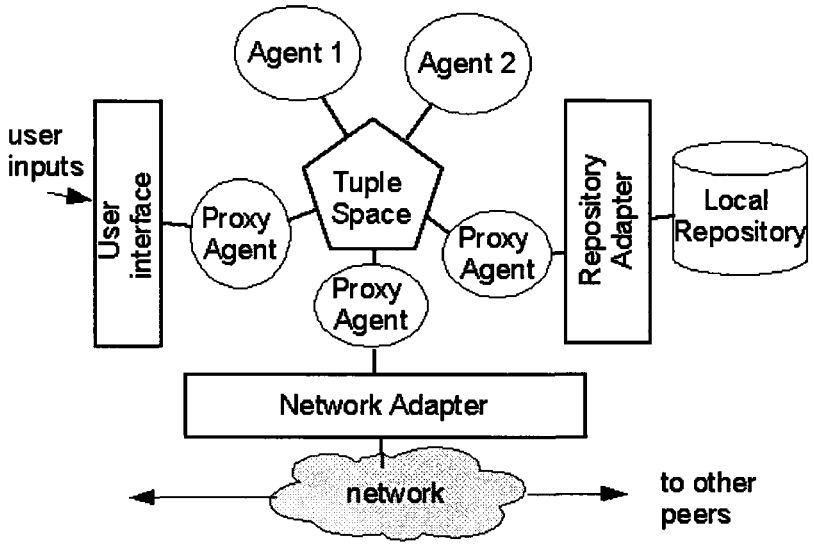


Figure 6.1: Architecture of U-P2P

Each proxy agent must thus have a set of templates, and react to any tuple matching these templates.

The proxy agents of U-P2P have a generic design, illustrated in Figure 6.2

Each template handler is a separate process, concurrently listening for new matching tuples.

This design ensures that the agents' interaction with the tuple-space (essentially listening for requests matching the different templates) is separate and asynchronous with respect to the actual processing of these queries, with a request queue that serves as a buffer for incoming requests. The queue ensures that requests are processed in the order in which they arrive. If the agent had a single process, several requests could arrive during the time necessary for a single request to be processed, and the agent would then arbitrarily select one of the new requests in the tuple space to process.

Proxy Agents that process tuples do not remove these tuples from the tuple-space, but they store a copy of each processed query until it disappears from the tuple-space. An additional agent, the "Cleaning Agent", monitors all the tuples in the tuple-space and removes them after a short lifetime in the tuple-space. The lifetime

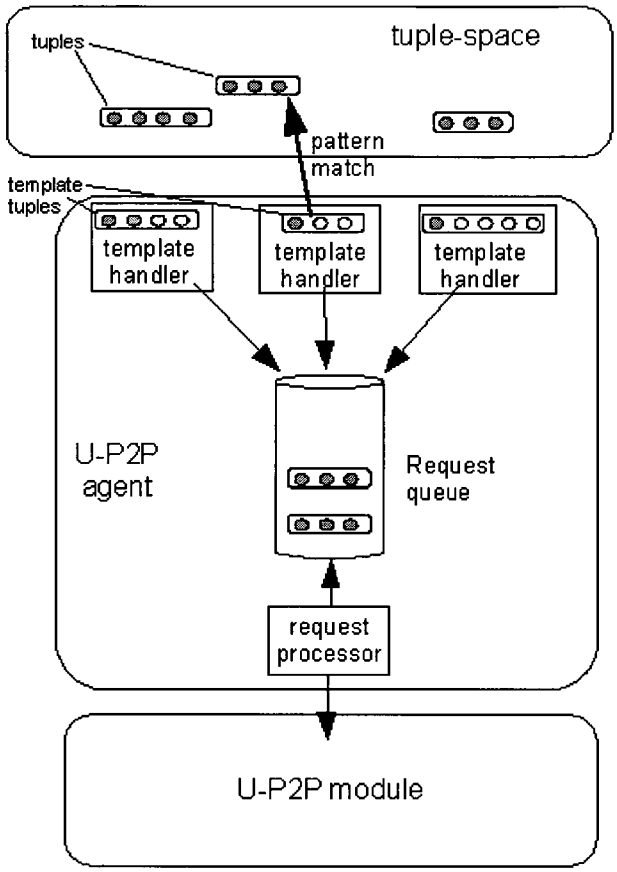


Figure 6.2: Generic Architecture of U-P2P Agents

is an adjustable parameter of the application; it must be long enough for all agents to read the tuple at least once, and short enough for tuples to be removed before another identical Tuple is likely to appear. This situation is unlikely to occur with only the basic Agents present, but in dynamic extensions where new agents may be created, this type of "collision" must be avoided.

6.2 Simple File-Sharing Functionality

This section describes how the basic File-Sharing functionality is implemented in our tuple space architecture.

6.2.1 Tuples

The fundamental P2P File-sharing operations are implemented as tuples exchanged by the different Proxy Agents. A tuple encoding a request has a *verb* in the first field, identifying the specific request, and appropriate parameters in the following fields. Searches have a special query identifier field that is used to track and route responses appropriately.

The tuple representation of the requests and responses, as well as the agents which may output each type of tuple are listed in table 6.1.

Operation	Tuple Representation	Output by
Publish	('Publish', *communityId*, *Document*)	user interface
Remove	('Remove', *communityId*, *documentId*)	user interface
Search Request	('Search', *communityId*, *expr*, *queryId*)	user interface, network adapter
Search Response	('SearchResponse', *communityId*, *documentId*, *peerId*, *Document**, *queryId*)	repository, network adapter
Download	('Retrieve', *peerId*, *communityId*, *documentId*)	user interface, network adapter
Download Response	('RetrieveResponse', *communityId*, *documentId*, *Document*)	repository, network adapter

Table 6.1: Fundamental File-Sharing operations in a Tuple Space architecture.

6.2.2 Processing

Each of these Request and Response tuples will trigger certain processing from the different Proxy Agents.

The following tables summarize the processing of the different request tuples by the Local Repository (Table 6.3), Network Adapter (Table 6.4), and User Interface

(Table 6.2). (N/A indicates that the Proxy Agent does not use a template matching the tuple.)

Publish	N/A
Remove	N/A
Search Request	N/A
Search Response	If the agent is currently listening for Responses with this *queryId*, then the temporary collection of Search Responses identified by the *queryId* is updated with the new Response
Download Request	N/A
Download Response	If the Agent is listening for this Download Response, then the Document is extracted from the tuple and rendered to the user in the User interface.

Table 6.2: Processing triggered by the different tuples in the User Interface.

6.3 Graph Query Processing

We now show how graph queries can be processed in this tuple space based design.

As shown by Algorithm 8 in the previous chapter, complex graph queries can be decomposed into simpler subqueries for processing by P2P file-sharing primitives.

At the implementation level, queries are processed by agents automatically generated from the specification of a query. The principle of these agents is that they are capable of decomposing complex queries into the basic building-blocks of U-P2P, which they can then invoke by outputting tuples (as defined in Table 6.1) to the tuple-space. The agents are rule-based, in order to handle multiple asynchronous responses to each sub-query that they output to the tuple-space.

We first show how the behavior of an agent can be specified by declarative rules,

Publish	Document stored in the local repository of the community identified by *communityId*
Remove	Document identified by *documentId* removed from local repository of community identified by *communityId*. If the document is not present, nothing happens.
Search Request	The query *expr* is evaluated against the local repository of community *communityId*; for each matching document *d*, a SearchResponse tuple is output.
Search Response	N/A
Download Request	If the *peerId* is that of the local peer, then the Agent creates a Download Response tuple with the document identified by *documentId* from the local repository of community *communityId*. The Download Response tuple is output to the tuple-space.
Download Response	If the Download Response was not output by the Repository Agent itself, then the Document is extracted from the Response and stored in the local repository of the community *communityId*.

Table 6.3: Processing triggered by the different tuples in the Local Repository.

Publish	N/A
Remove	N/A
Search Request	If the agent is currently listening for Responses to the same Request (identified by *queryId*), then the Request is ignored. If not, the Request is sent to other peers through the network, according to the network protocol of the community *communityId*.
Search Response	If the agent is currently listening for Responses with this *queryId*, then the Response is extracted and sent to the peer that originated the Request *queryId*.
Download Request	If the *peerId* is *not* that of the local peer, then the network adapter connects to the peer identified by this *peerId*, and retrieves the document. It then places the document in a Download Response tuple and outputs the tuple to the tuple-space.
Download Response	If the Download Response was not output by the Network Adapter Agent itself, then the Document is extracted from the Response and sent to the peer that originated the query.

Table 6.4: Processing triggered by the different tuples in the Network Adapter.

then give the rule-based definitions of agents implementing the query answering Algorithms 5 and 6. Finally, we show how a set of agents can be generated to cooperatively process a complex path pattern query.

6.3.1 A Rule-Based Agent Definition Language

The behavior of an Agent can be specified as a set of forward-chaining rules based on the Linda coordination language [50]. A list of rules specifies the behavior of an agent, and we briefly present an abstract language to specify such a list of rules.

Each rule is formed by a *Head*, which is a template matching tuples that will activate the rule, and a *Tail*, which is a sequence of tuple-instructions (tuple inputs and outputs) to be executed when the rule fires. Each tuple-instruction is a basic Linda operation specifier (in / out / read) followed by a tuple definition.

These tuples only contain String fields, and may include variables: input instructions (in or read) are defined by templates, and the fields of the tuples that match these templates are bound to variables that can be reused in subsequent instructions.

Our variable binding process is based on the concept of formal fields, to which we add a variable name instead of a value. In order to reuse a variable, we use what we call "variable" fields, which will appear at runtime to be literal fields, but whose values are filled in dynamically from stored variables. For example, a formal field bound to variable x will allow us to read a value and store it in the variable x, whereas a field declared as a *variable* field will be filled at runtime with the current value for x.

A single agent may have multiple rules sharing the same set of variables, and this avoids the need of loop constructs or condition statements, since the former can be replaced by a short rule which can be triggered repeatedly, and a conditional fork can usually be replaced by two rules triggered by the two alternative inputs of the condition.

In order for the execution of an agent to have a meaningful effect in the P2P file-sharing system, the tuples output by agents should match the templates of the permanent U-P2P modules, as listed in Table 6.1.

Finally, we note that we have added a functionality to lookup metadata values from search responses: The user interface temporarily stores search responses and answers XPath queries over the metadata values of the documents returned in the search[1].

This abstract "Agent Definition Language" can be represented by a schema. Figure 6.3 shows a diagram representing this schema.

[1]We remind the reader that search responses contain the metadata attributes of the documents matching the search, but not the attachments.

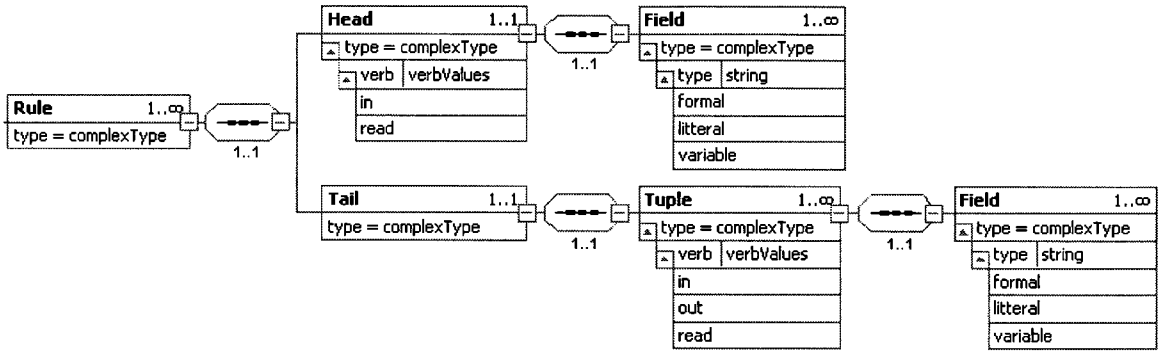


Figure 6.3: Schematic representation of the Agent Definition Language.

6.3.2 Elementary Query Processing Agents

Using the above agent definition language, we now give the definition of agents to process the query answering Algorithms 5 and 6.

We use the following notations:

- the Linda operation for each instruction is listed before each tuple, each tuple is limited by brackets, and the fields separated by semi-colons;
- ?*name* denotes a formal field where the value in the read tuple is stored in the variable *name*;
- $*name* denotes a variable field, i.e. a literal field where the value is populated at runtime from the variable *name*;
- fields with simply a value are literal fields containing the value;
- the symbol '+' denotes string concatenation.

The query answering algorithm for queries of the type of Query 5.4, defined in Algorithm 5, is implemented by the agent defined in Table 6.5.

The trigger of the rule is the verb "Subquery", with the input parameters of the algorithm, i.e *doc*, *comm*, and *endpoint*, and a query identifier which allows the answers of a query to be tracked. The second field of the tuple serves as a selector

Rule	:
Head	:
read	(Subquery; right; ?*doc*; ?*comm*; ?*endpoint*; ?*queryId*)
Tail	:
out	(Search; $*doc*; $*queryId*)
in	(SearchResponse; ?*docComm*; ?*docId*; ?*peerid*; ?*document*; $*queryId*)
out	(LookupSR; $*docComm*; $*docId*; $*endpoint*)
in	(LookupSR; $*docComm*; $*docId*; $*endpoint*; ?*URI*2)
out	(SubqueryAnswer; $*URI*2; $*queryId*)

Table 6.5: Definition of Agent implementing Algorithm 5.

to filter requests of the form of query 5.4, as opposed to queries of the form 5.5, for which the value of the second field is 'left', after the parameter of the corresponding $edge_{dir}$ predicate (see Section 5.4.2).

The first instruction in the Tail of the Rule is a Search Request based on the input URI, which basically allows the URI to be dereferenced: the metadata of the document will be available for local lookups. Note that if the document *doc* is not available in the network, the agent will not complete the processing of the rule, and no answers will be found to the subquery.

The following lines define a lookup of the value of the attribute *endpoint* in the document *doc*, and the output of the final response.

A similar definition can be given for an agent to process Query 5.5, defined in Algorithm 6. This agent definition is given in Table 6.6.

The processing of the query is triggered by the head of the first rule, which again contains the input parameters and a query identifier.

The Tail consists of an instruction making use of the built-in string concatenation function, in order to obtain the search condition from the input *endpoint* and document URI *doc*. Finally, the search is output.

*Rule*1	:
Head	:
read	(Subquery; left; ?*doc*; ?*comm*; ?*endpoint*; ?*queryId*)
Tail :	
out	(Search; $*comm*; $*endpoint* + '=' + $*doc*; $*queryId*)
*Rule*2	:
Head	:
in	(SearchResponse; $*comm*; ?*docId*; ?*peerid*; ?*document*; $*queryId*)
Tail	:
out	(SubqueryAnswer; up2p: + $*comm* + '/' + $*docId*; $*queryId*)

Table 6.6: Definition of Agent implementing Algorithm 6.

The purpose of the second rule is to convert each search response into a URI, which is the expected format of the search responses. The built-in *Concatenate* function is used again, to obtain a well-formed URI from the *communityId* and *documentId*.

This second rule will be triggered multiple times (for each response to the final "search" in the first rule) and thus function as a "loop" construct. Note that the rule is not active until its template is defined, which requires the query identifier and input variable *comm* input through the first rule.

6.3.3 Complex query processing

We now show how complex Graph Pattern Queries can be answered by agents.

Our approach is to decompose a path query of length n into n subqueries corresponding to the n edges, and generate an agent to coordinate the invocation of subqueries, relying on the agents defined in the previous section to execute these subqueries.

This coordination involves feeding the results of one subquery into the next, in the appropriate order, and can be managed by using distinct query identifiers for the

subqueries.

The coordinating agent for a Query of length n has $n+2$ rules, which are specified in Table 6.7.

Rule	0 :	
Head :	*read*	(PathQuery, ?*doc*, *id*0)
Tail :	*out*	(SubQuery, dir_n, \$*doc*, $comm_n$, $endpoint_n$, id_1)
Rule	1 :	
Head :	*in*	(SubQueryAnswer, ?*doc*, *id*1)
Tail :	*out*	(SubQuery, dir_{n-1}, \$*doc*, $comm_{n-1}$, $endpoint_{n-1}$, id_2)
	...	
Rule	i :	
Head :	*in*	(SubQueryAnswer, ?*doc*, id_i)
Tail :	*out*	(SubQuery, dir_{n-i}, \$*doc*, $comm_{n-i}$, $endpoint_{n-i}$, id_{i+1})
	...	
Rule	n :	
Head :	*in*	(SubQueryAnswer, ?*doc*, id_n)
Tail :	*out*	(SubQuery, dir_0, \$*doc*, $comm_0$, $endpoint_0$, id_{n+1})
Rule	$n+1$:	
Head :	*in*	(SubQueryAnswer, ?*doc*, id_{n+1})
Tail :	*out*	(PathQueryAnswer, \$*doc*, id_0)

Table 6.7: Rules of Query Coordinating Agent

Note: id_1 to id_{n+1} must be unique identifiers generated on creation of the agent.

The process of executing a query is thus as follows:

1. A query coordination agent must be created and activated
2. An initial Request, the tuple (PathQuery, *doc*, id_0) is output to the tuple space
3. Final answers to the query will match the template (PathQueryAnswer, ?*doc*, id_0).

6.4 Cost and Scalability Considerations

6.4.1 Analysis of the Cost of a Graph Query

We consider the cost of evaluating a path pattern query (as defined in Section 5.4.2) to be characterized by the following aspects:

- The network load (number of atomic search messages output, and queryHit messages received, during the execution of a graph query)
- The time delay to obtain results of a query : we evaluate here the time taken to obtain the first answers, rather than the average time for an answer to be returned, or the time for all answers to be obtained. The reason for this is that in a file-sharing network, complete answers to queries are not normally an expectation: a few relevant answers are usually what the user expects.

Example Query

In order to illustrate the query answering process, we take an example path pattern query of length 3 :

$$\begin{aligned} \{X | \exists Y_0, Y_1, \quad & edge_l(X, c_0, e_0, Y_0) \\ \wedge \quad & edge_l(Y_0, c_1, e_1, Y_1) \\ \wedge \quad & edge_r(Y_1, c_2, e_2, doc) \end{aligned}$$

The path pattern represented by this query is illustrated in Figure 6.4. X is the target of the query.

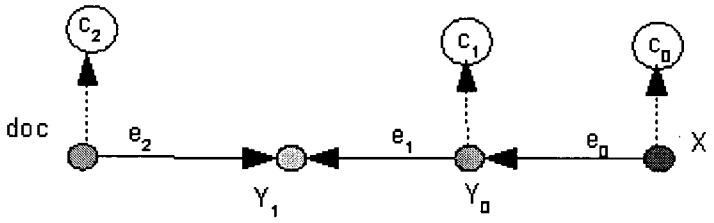


Figure 6.4: Path pattern represented by our example query.

Query Search tree

Answering this query requires searching the graph of data: the documents which will answer the subqueries, and ultimately the query itself, form a subgraph where the nodes are related by paths matching the query pattern, or a fraction of that path pattern.

An example search graph is illustrated in Figure 6.5.

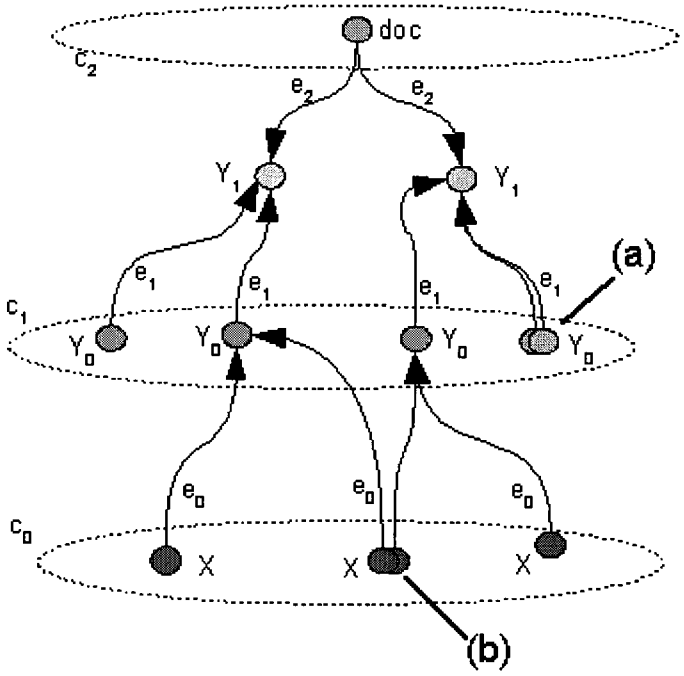


Figure 6.5: Search graph of our example query.

The nodes in this graph represent documents, and the dashed ellipses indicate that a group of documents are shared in a given community (which is also reflected

by colourings of the nodes).

The labels (a) and (b) indicate two special characteristics of this graph, which we detail here.

- The node marked by (a) illustrates the situation where a particular document is replicated in the network, and is returned multiple times as an answer to a given search subquery.
- The node marked by (b) illustrates the situation where a document occurs as part of two separate search paths.

In these two cases, multiple copies of the same document would be returned during the search process, and for practical considerations (particularly cost) we can represent this search graph by a tree, with a node for each occurence of a document. In this view, the duplicated nodes shown by labels (a) and (b) are each fully separated into two nodes.

In the following discussion we will thus consider this graph to be a tree.

Query Processing

The query is processed from the root of the tree, (*doc*), and the different branches of the tree are explored in parallel. For each branch the processing is identical, as each branch follows the same pattern.

We develop the different query processing steps on our example, and reference the relevant query processing Algorithms 4, 5 and 6 detailed in Chapter 5.

We assume that the peer is a member of all the relevant communities[2].

In our example case, the first step of the query is a dereference query, for document *doc*, followed by a lookup of its endpoint attribute e_2 (Algorithm 5). The next step

[2]In the contrary situation, finding and downloading the communities entails $O(n)$ searches and downloads (the query is related to at most n different communities), which as we will see further is negligible compared to the number of searches and responses for documents

of the query is a search in community c_1 for candidate documents Y_0 (Algorithm 6). Third, a search in community c_0 provides the final results X of the query (Algorithm 6).

Network Load

We now evaluate the network load involved in processing a graph query.

The query answering process can be seen as the parallel exploration of the branches of its search tree. For each node that is not a final answer to the query, (all nodes except those at depth n), one of Algorithms 6 or 5 is executed, to obtain children of that node. These two algorithms each involve a single search message output to the network.

The number of output searches is thus equivalent (in the "big O" sense) to the number of nodes in the subtree of depth n of the search tree, rooted at the search tree's root. The number of query answer messages, in our view of the search tree, is equivalent (again in the "big O" sense) to the total number of nodes in the graph: for answers found locally, no search reply message was output to the network.

The total network load is thus equivalent, in a "big O" sense, to the number of nodes in the search tree.

The number of nodes in a tree can be simply characterized by two quantities: the depth of the tree, and the branching factor of the nodes.

The number of nodes in a tree is equivalent (in the "big O") sense, to the branching factor[3] of the tree to the power of the tree depth.

The branching factor of a given node in the search tree represents the number of answers to the *edge* subquery that was output at that node. In general, the number

[3]The braching factor of a tree is often considered to be the maximal branching factor of its nodes; we consider here an "effective branching factor", without entering the technical details of its definition, as an average branching factor over the tree. Technical details would be meaningless here since we do not attempt to formally characterize this branching factor.

of answers returned for a given search will depend both on the parameters of the search itself, and on the structure of the graph of data. This branching factor is not a quantifiable parameter of a query, and for the purpose of evaluating the cost of a query, we will consider it to be an arbitrary constant b.

The depth of a search tree is equal to the length n of the path pattern in the query.

The network load associated with a path pattern query of length n is thus : $O(b^n)$, i.e. exponential in n.

Time to Receive First Answers

With respect to time, as the exploration of the different branches in the tree is done in parallel and independently, the time until the first answer will be the shortest time to explore a branch of length n, i.e. the minimum of this quantity over all branches.

We assume that the time to process an elementary search in a given node is negligible compared to the network delay of obtaining a response from a different peer.

The time to process a particular branch is then the sum of the network delays to obtain the results of the subqueries, the shortest possible delay being the negligible time of obtaining the result locally.

The average delay to receive a query answer is related to the physical network topology, and the distribution of documents (location and replication) in this physical network.

If answers are returned with an average delay of d, the time to process a particular branch of the search tree will be $O(n \times d)$.

The minimum over all branches, i.e. the time to receive the first answers in the graph, is dominated by the time to process any given branch. We can thus also estimate it to be $O(n \times d)$, or linear in the length n of the query.

We note that a query returning many answers will have a search tree with a high branching factor, and a higher likelyhood of obtaining many answers locally or with small delays. Hence the parameters given for the network load will also influence the time to obtain answers.

Conclusions

We can thus conclude that the main quantifiable factor influencing the cost of graph query execution is the length n of the path pattern query : the network load (number of search messages output) is exponential in n, and the time to obtain answers is linear in n.

Secondary factors that influence these costs are the following:

- the structure of the graph of data, such as the degree of the nodes : this factor influences mostly the network load, and to a minor degree, the time to receive answers.
- the distribution (location and replication) of the documents answering the different subqueries : this factor influences mostly the time to receive the first answers. A high level of replication also increases the network load by providing many redundant answers to queries.
- the topology of the physical network influences the time to receive answers.

The influence of these factors is not quantifiable in general.

6.4.2 Scalability

The functionalities offered by our "linked data" approach to file-sharing - management of multiple communities, graph queries - are built on top of a fundamental layer where the basic operations of file-sharing, namely publish, delete, search, download,

are processed using a traditional file-sharing protocol, which is the Gnutella protocol in our current implementation U-P2P.

The major concern in terms of scalability for file-sharing applications, is the network load[4] represented by searches and search responses. With respect to the network, the scalability of our solution directly depends on the cost and scalability of the underlying protocol.

We thus take the scalability of the underlying protocol as a baseline, and discuss here the scalability of our solution with respect to this baseline.

The central difference between our model and a traditional file-sharing system, is the multiplicity of communities. In our model, all our considered operations are community-specific, and thus searches only reach the peers that are members of the considered community: these peers only represent a subset of the total network of peers[5].

For each search output to the network, a community-specific search will thus generate less network traffic than a search in a traditional file-sharing network. This would imply that our file-sharing model scales better than the baseline.

With the introduction of complex graph queries, the network load generated by the processing of such queries may offset the benefits of multiple communities. As discussed in the previous section, the number of search and search reply messages generated by the execution of a graph query of length n is exponential in n.

This suggests that the cost of processing graph queries could negatively impact the scalability of our solution.

However, we could assume that a user's information needs are better served by expressive queries such as graph queries, and that a given information need may require fewer complex queries than it would require simple searches.

[4]see for example the study [20] for a discussion.

[5]with the exception of the Community community, to which all peers belong

This discussion shows that the scalability of our solution cannot easily be evaluated by a theoretical study. As comparable systems do not currently exist, we cannot conjecture on the proportion of peers in the network belonging to each community, the topology of the graph of data, the usage of complex queries by the users, or on other criteria which determine the scalability of our solution.

A proper scalability evaluation would require a large-scale deployment and extensive testing.

Chapter 7

Case Study

This chapter presents a case study where we set up a collection of interlinked documents in a U-P2P network, and execute some graph queries over this data.

7.1 Data : Freebase Films and Actors

The data we used in this case study is a modified dataset from Freebase[1].

Freebase data is made available to the public in the form of data "dumps", which are "tsv" files (tab separated values) each containing the data of a Freebase table. Each row in the file corresponds to a "topic" (an entity modeled in the table), with the fields separated by <tab>characters, and within each field, multiple values separated by commas, if applicable.

We collected the file `film.tsv`, listing several thousand films, and the file `actor.tsv` listing several thousand actors.

Films are described by the following attributes : `name, id, initial release date, directed by, produced by, written by, cinematography, edited by, music, language, rating, estimated budget, imdb id, country, starring, runtime, locations, film collections, soundtrack, featured`

[1] http://www.freebase.com, see also [15] and Section 3.2.1 for an overview

`film locations, genre, film series, story by, sequel, prequel, subjects, personal appearances, dubbing performances, film format, costume design by, other crew, trailers, distributors, other film companies, production companies, tagline, release date, netflix id, film festivals, nytimes id, featured song.`

Most of these attributes are optional, and may be multi-valued; in fact, most of the films described in the data lack a large part of these attributes.

Actors are described by the following attributes : `name, id, film, dubbing performances, imdb entry, netflix id, nytimes id.`

For actors, these attributes are also mostly optional and may have multiple values.

We note that in both cases the list of attributes does not correspond exactly to the attributes visible in the browsing interface of Freebase.

As in our examples of Chapter 5, we focus on the relationship between actors and the films that they have played in.

This relationship is expressed in the data by the attribute "films" in the table "Actors", and the attribute "starring" of films. Although Actors and Films have a unique identifier (the second attribute, in both cases), the attribute "films" does not reference films by their `id`, but rather lists keys that are repeated in the "starring" attribute of the relevant films.

To compare with our example of Section 5.3.5, we illustrate in Table 7.1 the link between the freebase entry for actress Liv Tyler, and the entry for the film "The Two Towers". We only show the relevant values in the attributes "actor/film" and "film/starring".

The relationship between the actress and the film is explicited by the repeated identifier "/guid/9202a8c04000641f8000000007235959". Other values in the "films" attribute for actress Liv Tyler relates the actress to other films she has played in, and the other identifiers in the film entry relate the film to other actors who have played

name	*films*
Liv Tyler	**/guid/9202a8c04000641f8000000007235959**, /guid/9202a8c04000641f800000000052ee7d, [...]

name	*starring*
The Lord of the Rings: The Two Towers	**/guid/9202a8c04000641f8000000007235959**, /guid/9202a8c04000641f80000000070c1184, /guid/9202a8c04000641f800000000acc1580, [...]

Table 7.1: Freebase entries for actress Liv Tyler and film "The Two Towers".

in that film.

From this data we collected a sample of approximately 600 films and 400 actors, which we converted to U-P2P documents. In order to have a well-connected graph, we focused on American films, and selected films that were released after the year 2000.

We first created an XML schema for films, and parsed the list of films, creating an XML document for each film. Then we created an XML schema for Actors, and parsed the list of actors.

We replaced the repeated keys of the "Actor/films" and "Film/starring" attributes by a one-way reference in the Actor/film attribute, in the form of a `up2p:` URI. The attribute "Actor/films" is thus a multi-valued endpoint attribute, as in our example of Section 5.3.5.

7.2 Experiments

7.2.1 Setup

For the purpose of these experiments, we installed our prototype file-sharing application U-P2P on three networked computers. We created a community of Films, and a community of Actors, that we published to two peers each, so that one peer was

member of both communities, and the two others were each member of one community[2].

We then spread out the Actor and Film documents in each community, with some documents only on one peer, others replicated on both peers of a community.

Our deployment of documents was thus somewhat similar to the example of Section 4.5.4, except that the Peer 2 was a member of all three communities, i.e. Community, Actor, and Movie (which we call here Film after the term used in the Freebase data).

Mirroring the example of Section 4.5.4, we illustrate our experimental setup in Figure 7.1. We have also added edges showing the graph structure of the data. Note that this diagram is incomplete, showing only a small fraction of the documents present in our experimental P2P network, and also missing some edges of the graph, which would overload the diagram.

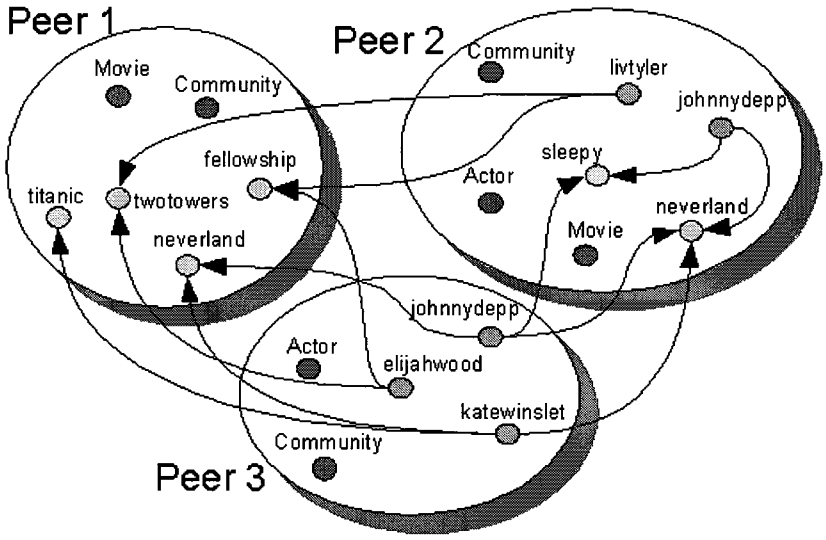


Figure 7.1: Sample of a graph of data, deployed in three peers.

[2]Note that all peers were also members of the Community community, which made them respectively members of three, two and two communities.

7.2.2 Queries

An interesting relationship between actors, briefly discussed in the context of the example queries of section 5.4.2, is the concept of "degrees of separation" : actors who have played in the same film can be seen as being only one "degree" apart.

In our graph of data, such relationships are represented by patterns of two edges.

We chose several actors at random from our collection, and queried the data repository to find out who was related to them by one, two, or three degrees of separation. As our graph of data turned out to be well connected, a radius of three degrees of separation from a given actor returned approximately half of the overall data set.

As an example, we detail the figures obtained for one and two degrees of separation from the actress Charlize Theron, who was modeled in our data collection.

The document describing Charlize Theron listed six films that she had played in, that were also in our sample. The subsequent query for actors who had also played in these films, and were present as documents in the file-sharing network yielded seven actors, which can be said to be related to C. Theron by one degree of separation. A query for films that these actors had played in resulted in 35 films, and finally a total of 52 actors were found that had played in these films, thus constituting the group of actors related to Charlize Theron by (less than) two degrees of separation.

The first few results to this query took approximately five to ten seconds, and the last took three minutes.

Note that if we wanted to select actors related by exactly two degrees of separation, as opposed to "at most two degrees", our model of graph queries would not be sufficiently expressive to specify this, as the films retrieved in the two steps of the query would need to be different.

In order to try queries with an odd number of edges, we also added an edge to

the "degree of separation" queries: this allows for example to discover the films that actors related to another actor have played in.

With these different types of queries, we can observe the evolution of the network load and query time for query lengths ranging from two to six edges.

Our main findings are presented in the following section.

7.2.3 Results and Analysis

With respect to the network load, i.e. the total number of search messages and responses that are output to the network while the query is being processed, the numbers increase strongly with the length of the query. This tendency is visible in the graph of Figure 7.2. The squares indicate the overall traffic generated by the query, which can be related to the number of final answers obtained, which is shown by the stars. The smooth line is an interpolation of the squares, meant to show the exponential increase of the network traffic with respect to the query length.

This degree of complexity could be expected, from the complexity analysis presented in ther previous chapter[3]. We may notice that with respect to the exponential increase in cost, the queries of even length are rather below the lines, whereas the queries of length 3 and 5 are rather above; as we have mentioned previously these even queries of our example are conceptually all very similar, representing degrees of separation between actors, whereas the odd queries return films that these actors may play in. Due to the structure of the graph, the query answering algorithm ends up being slightly different. This difference is apparent for small values of n, and disappears from a complexity analysis where n is assumed to be high.

The duration of query processing for different query lengths is shown in Figure 7.3. This graph was obtained by recording the specific times where the final answers

[3]Note that this observation does not claim to conclusively prove the exponential nature of the phenomenon

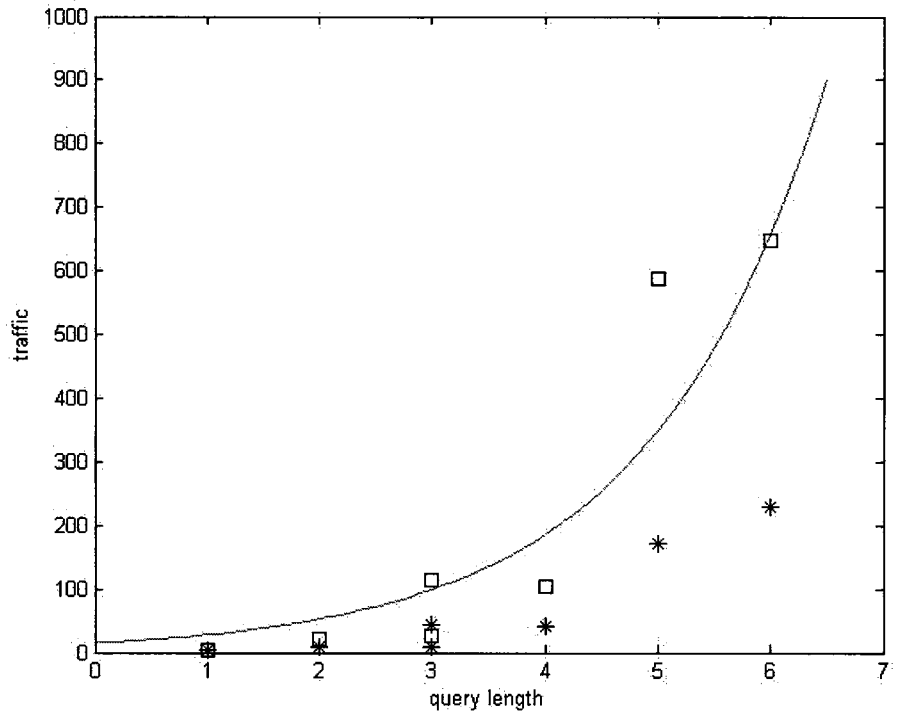


Figure 7.2: Number of messages exchanged during the execution of a query of length n.

to queries appeared in the tuples-space of the peer processing the query, either from the local repository or from a remote peer.

The horizontal axis represents time on a logarithmic scale, where the unit is the millisecond, and the length of queries goes up on the Y axis. To give the reader an idea of the timescale, the quickest answers at the bottom left took around half a second, the two isolated points in the top center correspond to times of five and six seconds, and the very last answers to the right of the graph took between eleven and twelve minutes.

We observe that simple two-edge queries can be processed very quickly, partly due to locally available answers, whereas it appears clearly here that the average query answering time grows exponentially with the length of the query.

To compare this phenomenon with our previous cost analysis (Section 6.4), we

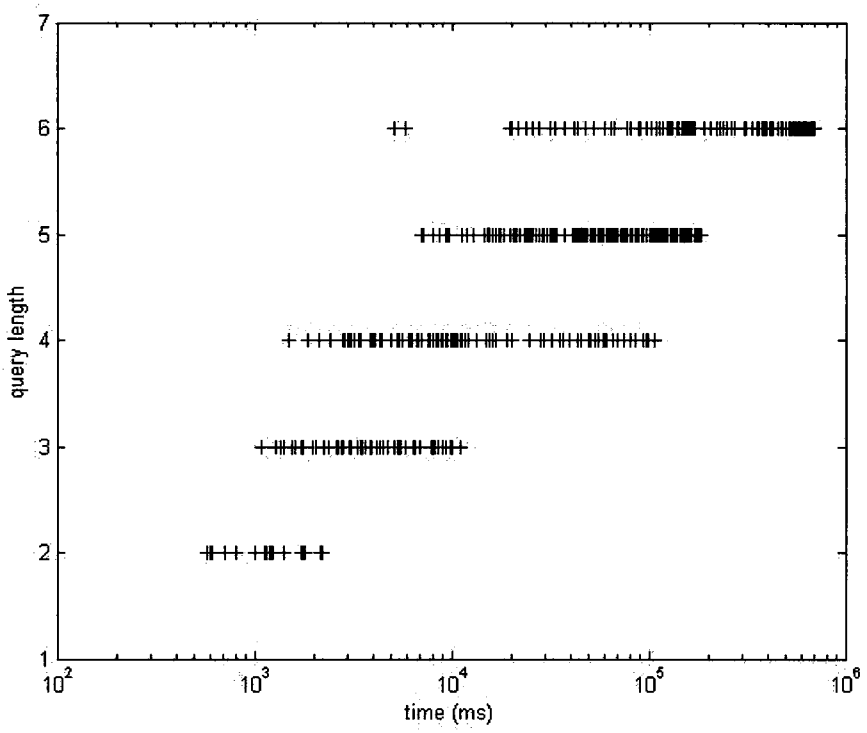


Figure 7.3: Distribution of query processing time for different lengths of graph queries.

can see several reasons for this exponential processing time. Mainly, our hypothesis that the local processing time of queries is very small compared to network delays is not true in this case.

As the number of queries grows exponentially in the network, the view of a parallel processing no longer holds, as each peer becomes overloaded by the exponential number of requests.

In our opinion, this costly processing is not a major obstacle. For one thing, despite the long delay in obtaining the last results to a query, the first results have always appeared in a few seconds. Once the queries start arriving, the user may start analyzing these results and deciding where to go next.

In the context of file-sharing, queries are usually seen as a "best-effort" feature, rather than having strict soundness and completeness requirements.

Furthermore, we believe that the setting of this experimentation may not be typical of a file-sharing network deployment.

The network that was used for this experimentation is a local, high-bandwidth network in a research lab, and only contains three peers. We also used a fairly large amount of data in only two communities, thus requiring relatively heavy processing with little physical distribution: in our vision, a typical deployment of U-P2P would rather cover a broader range of domains, to a shallower level, and perhaps generate more network traffic with less heavy local query processing.

Finally, we note that our tuple space based design, while being well suited for ad-hoc query processing, is not a model of high efficiency. As the interactions between components go through the tuple space, real-time constraints over this tuple space tend to produce a bottleneck effect.

In general, we consider that algorithmic and technical improvements to query processing are an important part of U-P2P, and of other data management systems, but are very much orthogonal to the discussion developed in this thesis.

Chapter 8

Conclusion

8.1 Solution to the Research Problem

The core of our research problem was to apply the contribution model of P2P file-sharing to the collaborative construction of knowledge.

This thesis describes a complete solution to this problem:

- We have formally defined the principles of P2P file-sharing applications for structured data (also known as "schema-based file-sharing systems"), in Sections 4.3 and 4.4.

- We have formally described an extension to this model that supports multiple communities. This extension is described in Section 4.5. We note that a prototype implementing this extended model existed in the form of the prior U-P2P prototypes, previously presented in [3, 48, 49], but the model itself is a new contribution.

- We have formally described a second extension to this model, that introduces the concept of "endpoint attributes", defining links between documents of a file-sharing network and inducing a graph data model.

  This model is described in Section 5.3.

- We have proposed a class of graph queries for the graph of interlinked documents, and algorithms to answer these queries. Our graph query language and its associated query answering algorithms are defined in Section 5.4.

The specific requirements of decentralized control, file-sharing contribution model, and suitability for knowledge construction are addressed as follows:

- The basic file-sharing operations, formally defined in Section 4.4 define an interface to the community such that a peer has full and exclusive control over its local storage.

- The general contribution model of file-sharing can also be described by the operations defined in Section 4.4. Our solution exclusively relies on these operations for the contribution of documents, but also of communities, links in the graph, and queries.

- With its data model directly comparable to RDF, and supporting graph queries, our solution clearly supports the construction of knowledge, by allowing to annotate and describe interrelated concepts.

8.2 Summary of Contributions

Our main contributions are the following:

- We have formally defined the principles of P2P file-sharing applications for structured data. Our model defines the fundamental concept of a community, models the transient relationship of a peer participating in a file-sharing community, and establishes a framework of simple data management operations : publish, remove, search, download, and lookup.

- We have provided formal foundations for file-sharing in a network with multiple communities, and shown how multiple communities define a metamodel for modeling concepts and instances of these concepts.

- We have defined a URI scheme suitable for documents shared in a multiple-community network, and a dereferencing mechanism for this URI scheme.

- We have presented a graph data model based on this URI scheme, a class of graph queries for this data model, and algorithms to answer such graph queries in the file-sharing network. In addition, the file-sharing principle is applied to these queries by sharing the queries in the network, thus supporting the reuse of queries between peers.

- We have presented an extensible design for a file-sharing application based on a tuple space, demonstrating the use of this computation model for processing complex tasks with asynchronous characteristics, such as graph queries over a file-sharing network.

Derived Publications The main contributions of this thesis have been presented in the following peer-reviewed publications :

- Davoust, A., Esfandiari, B.: Towards semantically enhanced peer-to-peer file-sharing. In Meersman, R., Tari, Z., Herrero, P., eds.: OTM Workshop on Semantic Extensions to Middleware. Volume 5333 of Lecture Notes in Computer Science., Springer (2008) 937–946

- Davoust, A., Esfandiari, B.: Towards semantically enhanced peer-to-peer file-sharing. Journal of Software **4**, Special Issue on Semantic Extensions to Middleware. (2009) (to appear).

- Davoust, A., Esfandiari, B.: Linking and Navigating Data in a P2P Data-Sharing Network. (demo) Proceedings of the Linked Data on the Web Workshop, Madrid, Spain, April 21, 2009, CEUR Workshop Proceedings. (2009) (to appear)

- Davoust, A., Esfandiari, B.: Peer-to-peer sharing and linking of social media based on a formal model of file-sharing. Submitted for publication in Journal of Distributed and Parallel Databases, Special Issue on Data Management in Social Media. Available as: Technical Report SCE-09-04, Department of Systems and Computer Engineering, Carleton University (2009)

- Davoust, A., Esfandiari, B.: Collaborative Building, Sharing and Handling of Graphs of Documents Using P2P File-Sharing. OTM Workshop on Semantic Extensions to Middleware. (2009)

8.3 Limitations and Future Work

8.3.1 Data Emergence in the File Sharing Network

An important aspect of our research problem was to retain certain interesting properties of file-sharing networks, in particular the "darwinian" emergence of popular or useful data.

Our model allows end-users to contribute data at all levels, i.e. users can publish and download documents, communities, linked documents, or queries. We consider this contribution model to be a fundamental component of the file-sharing paradigm, and conjecture that in itself it implies the emergence of popular documents, communities, relevant links and useful queries.

However, validating this conjecture would require a large-scale deployment of our application, and extensive testing, which was not feasible in the context of this thesis.

8.3.2 Expressiveness of Queries

The class of graph queries presented in this thesis has a limited expressiveness, as discussed in Section 5.4.3. While we believe this language to be useful for exploring the graph of data, it is more of a proof-of-concept proposal than an optimal language.

An interesting direction for future research would be to extend our query processing approach to support more complex graph patterns.

8.3.3 Semantics

Within the broader context of Semantic Web oriented research, the scope of our research problem can be related to the Semantic Web layer cake presented in the introduction. This thesis specifically addresses the layer of "linked data", represented in the cake by the W3C standard Resource Description Framework.

Despite some discussion on the semantics of certain data items, such as community definition documents, formal knowledge representation and semantic reasoning are beyond the scope of the model that we have presented.

The most likely direction for future work in the ongoing project U-P2P is another step up in the Semantic Web layer cake, to explore the possibility of collaborative construction of ontologies in a P2P file-sharing network.

List of References

[1] T. Berners-Lee, J. A. Hendler, and O. Lassila, "The semantic web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.

[2] T. O'Reilly, "What is web 2.0." http://oreilly.com/web2/archive/what-is-web-20.html, 2005.

[3] A. Mukherjee, B. Esfandiari, and N. Arthorne, "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*, (Washington, DC, USA), pp. 701–705, IEEE Computer Society, 2002.

[4] "The RDF suite of specifications," 2004.

[5] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform resource identifier (URI): generic syntax," RFC 3986, Internet Engineering Task Force, jan 2005.

[6] T. Berners-Lee, T. Bray, D. Connolly, P. Cotton, R. Fielding, M. Jeckle, C. Lilley, N. Mendelsohn, D. Orchard, N. Walsh, and S. Williams, "Architecture of the world wide web, volume one," December 2004.

[7] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," tech. rep., W3C, 2006.

[8] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of sparql," in *International Semantic Web Conference* (I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo, eds.), vol. 4273 of *Lecture Notes in Computer Science*, pp. 30–43, Springer, 2006.

[9] J. Giles, "Internet encyclopaedias go head to head," *Nature*, vol. 438, pp. 900–901, December 2005.

[10] S. Auer and J. Lehmann, "What have Innsbruck and Leipzig in common? extracting semantics from wiki content," in *ESWC* (E. Simperl, J. Diederich,

and G. Schreiber, eds.), vol. 275 of *CEUR Workshop Proceedings*, pp. 503–517, CEUR-WS.org, 2007.

[11] H. Halpin, "Is there anything worth finding on the semantic web?," in *18th International World Wide Web Conference (WWW2009)*, April 2009.

[12] A. Akbik and J. Bross, "Wanderlust: Extracting semantic relations from natural language text using dependency grammar patterns," in *Proceedings of the Semantic Search 2009 Workshop, SemSearch09 co-located with the 18th World Wide Web conference, Madrid, Spain, April 20th, 2009*, 2009.

[13] M. Buffa, F. Gandon, G. Ereteo, P. Sander, and C. Faron, "Sweetwiki: A semantic wiki," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 1, pp. 84–97, 2008.

[14] D. B. Martin Hepp and K. Siorpaes, "Ontowiki: Community-driven ontology engineering and ontology usage based on wikis," in *Proceedings of the 2005 International Symposium on Wikis (WikiSym 2005)*, OCT 2005.

[15] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, (New York, NY, USA), pp. 1247–1250, ACM, 2008.

[16] R. Cook, "Re:[data-modeling] "public person" and privacy." Email on Freebase Data Modeling mailing list, june 2008.

[17] P. Mika, "Ontologies are us: A unified model of social networks and semantics," in *International Semantic Web Conference*, vol. 3729 of *Lecture Notes in Computer Science*, pp. 522–536, International Semantic Web Conference 2005, Springer, November 2005.

[18] L. Zhang, X. Wu, and Y. Yu, "Emergent semantics from folksonomies: A quantitative study," *Journal on Data Semantics VI*, 2006.

[19] "Gnutella: The gnutella protocol specification 0.6." http://rfc-gnutella. sourceforge.net, 2002.

[20] Q. Lv, S. Ratnasamy, and S. Shenker, "Can heterogeneity make gnutella scalable?," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, (London, UK), pp. 94–103, Springer-Verlag, 2002.

[21] P. Haase, B. Schnizler, J. Broekstra, M. Ehrig, F. van Harmelen, M. Menken, P. Mika, M. Plechawski, P. Pyszlak, R. Siebes, S. Staab, and C. Tempich, "Bibster–a semantics-based bibliographic peer-to-peer system," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 2, no. 1, pp. 99 – 103, 2004.

[22] D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati, "Logical foundations of peer-to-peer data integration," in *PODS* (A. Deutsch, ed.), pp. 241–251, ACM, 2004.

[23] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov, "The piazza peer data management system," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, pp. 787–798, July 2004.

[24] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov, "Schema mediation in peer data management systems," in *ICDE* (U. Dayal, K. Ramamritham, and T. M. Vijayaraman, eds.), pp. 505–, IEEE Computer Society, 2003.

[25] I. Tatarinov and A. Y. Halevy, "Efficient query reformulation in peer-data management systems," in *SIGMOD Conference* (G. Weikum, A. C. König, and S. Deßloch, eds.), pp. 539–550, ACM, 2004.

[26] P. Adjiman, F. Goasdoué, and M.-C. Rousset, "SomeRDFS in the semantic web," *Journal on Data Semantics*, vol. 8, 2007.

[27] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pp. 46–66, July 2000.

[28] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "Opendht: a public dht service and its uses," in *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, (New York, NY, USA), pp. 73–84, ACM, 2005.

[29] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *Networking, IEEE/ACM Transactions on*, vol. 11, no. 1, pp. 17–32, 2003.

[30] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "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, October 2001.

[31] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," *Lecture Notes in Computer Science*, vol. 2218, pp. 329–??, 2001.

[32] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 41–53, 2004.

[33] R. Dingledine, M. J. Freedman, and D. Molnar, "The free haven project: Distributed anonymous storage service," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability* (H. Federrath, ed.), Springer-Verlag, LNCS 2009, July 2000.

[34] A. D. R. Marc Waldman and L. F. Cranor, "Publius: A robust, tamper-evident, censorship-resistant, web publishing system," in *Proc. 9th USENIX Security Symposium*, pp. 59–72, August 2000.

[35] R. J. Anderson, "The eternity service," in *In Proceedings of Pragocrypt*, pp. 242–252, 1996.

[36] T. Vander Wal, "Folksonomy coinage and definition." http://www.vanderwal.net/folksonomy.html, 2007.

[37] I. Ohmukai, M. Hamasaki, and H. Takeda, "A proposal of community-based folksonomy with rdf metadata," in *Proceedings of the Workshop on End User Semantic Web Interaction, held in conjunction with the International Semantic Web Conference 2005 (ISWC'05)*, (Galway, Ireland), Springer, November 2005.

[38] V. Tanasescu and O. Streibel, "Extreme tagging: Emergent semantics through the tagging of tags," in *ESOE* (L. Chen, P. Cudré-Mauroux, P. Haase, A. Hotho, and E. Ong, eds.), vol. 292 of *CEUR Workshop Proceedings*, pp. 84–94, CEUR-WS.org, 2007.

[39] S. Bindelli, C. Criscione, C. A. Curino, M. L. Drago, D. Eynard, and G. Orsi, "Improving search and navigation by combining ontologies and social tags," in *OTM '08: Proceedings of the OTM Confederated International Workshops and*

Posters on On the Move to Meaningful Internet Systems, (Berlin, Heidelberg), pp. 76–85, Springer-Verlag, 2008.

[40] M. Cai and M. Frank, "Rdfpeers: A scalable distributed rdf repository based on a structured peer-to-peer network," in *WWW '04: Proceedings of the 13th international conference on World Wide Web*, (New York, NY, USA), pp. 650–657, ACM, 2004.

[41] P. Cudré-Mauroux, S. Agarwal, and K. Aberer, "GridVine: an Infrastructure for Peer Information Management," *IEEE Internet Computing*, vol. 11, no. 5, pp. 864–875, 2007.

[42] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica, "Complex queries in dht-based peer-to-peer networks," in *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, (London, UK), pp. 242–259, Springer-Verlag, 2002.

[43] P. A. Boncz and C. Treijtel, "AmbientDB: Relational query processing in a P2P network," in *DBISP2P* (K. Aberer, V. Kalogeraki, and M. Koubarakis, eds.), vol. 2944 of *Lecture Notes in Computer Science*, pp. 153–168, Springer, 2003.

[44] T. Berners-lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets, "Tabulator: Exploring and analyzing linked data on the semantic web," in *In Procedings of the 3rd International Semantic Web User Interaction Workshop (SWUI06*, p. 06, 2006.

[45] M. Jarrar and M. D. Dikaiakos, "MashQL: a query-by-diagram topping SPARQL," in *ONISW '08: Proceeding of the 2nd international workshop on Ontologies and nformation systems for the semantic web*, (New York, NY, USA), pp. 89–96, ACM, 2008.

[46] R. Rivest, "The MD5 Message-Digest algorithm," RFC 1321, Internet Engineering Task Force, apr 1992.

[47] T. Berners-Lee, "Linked data." World Wide Web design issues, July 2006.

[48] N. Arthorne, B. Esfandiari, and A. Mukherjee, "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.

[49] N. Arthorne and B. Esfandiari, "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, (New York, NY, USA), p. 14, ACM, 2006.

[50] D. Gelernter and N. Carriero, "Coordination languages and their significance," *Commun. ACM*, vol. 35, no. 2, pp. 97–107, 1992.

[51] C. Pedrinaci, T. Smithers, and A. Bernaras, "Opportunistic reasoning for the semantic web: Adapting reasoning to the environment," in *New Forms of Reasoning for the Semantic Web* (R. Piskac, F. van Harmelen, and N. Zhong, eds.), vol. 291 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2007.

[52] R. Tolksdorf, E. Bontas, and L. Nixon, "Towards a tuplespace-based middleware for the semantic web," in *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pp. 338–344, Sept. 2005.