Traceable Document Flows

Martin Bernauer[†], Gerti Kappel[†], Elke Michlmayr[*]
[†] Business Informatics Group (BIG),
[*] Women's Postgraduate College for Internet Technologies (WIT),
Institute for Software Technology and Interactive Systems
Vienna University of Technology, Austria
{lastname}@big.tuwien.ac.at, [*] michlmayr@wit.tuwien.ac.at

Abstract

Ad-hoc data exchange, e.g., by sending email attachments, leads to multiple copies or versions of a document at dispersed nodes in a network. However, their relationships such as is-a-version-of are lost. The relationships between a single document's existing versions together with the versions themselves form a so-called document flow, which allows to trace where a document originated and how it was distributed from node to node. The paper proposes an ontological, machine-processable framework for semantically describing and thus revealing document flows. The presented approach provides a light-weight communication infrastructure for information exchange and collaboration that is based on distributed versioning of documents and offers flexible document annotation mechanisms. After describing our infrastructure model, we present the implemented prototype and discuss a range of application scenarios for traceable document flows (TDFs).

1. Introduction

With the advent of the Web, ad-hoc data exchange between individuals has become a daily routine. Unfortunately, there are a lot of situations where documents[1] are exchanged without preserving any metadata except for the document's file name. Above all, this applies for the situation where persons want to exchange documents but cannot rely on the same information infrastructure, e.g., because they are working for different companies. Instead, they use emails with attachments, instant messaging, or filesharing applications (e.g., FTP).

Thus today's ad-hoc data exchange has the drawbacks of disregarding metadata describing *document flows* between individuals and losing metadata describing documents themselves. The following examples show the difficulties one may encounter. First, it is hardly possible to determine where a document in one's file system originated. Usually answers to this question are found after human inquiries using one's email system. Second, it is even more difficult to determine who read or edited a document before. Usually this involves lengthy inquiries and requires interviewing other people. Third, metadata about the document, if not stored proprietarily as part of the document, e.g., as it is the case with documents in Sun StarOffice or Microsoft Office format, are lost during data exchange between individuals. Again, harvesting metadata is a human task carried out by determining and interviewing people who read or edited the document before.

The first contribution is to provide an *ontological, machine-processable framework* for describing document flows and documents, thereby revealing the flows and metadata describing documents (referred to as annotations). With it, users can query metadata describing document flows, can query annotations, and can determine the past and current contents of documents. By employing Semantic Web technology, i.e., RDF [24] and RDFS [23], the metadata is described non-proprietarily and can be used by any application.

The second contribution is to propose an *infrastructure model* defining how document flows can take place in parallel with the ontological framework. Basically, every document has a globally unique identifier which is preserved when sending it across the network. Because a document may be edited by its current owner and re-distributed afterwards, the model keeps track of various versions of the document spread across the network. When re-distributing a document, its document flow and associated metadata are distributed along with it.

* This research has been partly funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

1 Throughout the paper the notion "document" refers to a file of any file type.

The infrastructure model for document flows is a peer-to-peer (P2P) model so that a shared, central information infrastructure is not necessary for document exchange. It would also be very unlikely that for every document exchange, which may involve several individuals, a single server can be negotiated or at least it would be a disruptive overhead. A P2P model has the advantage that involved users are loosely coupled and keep their autonomy. Naturally, the disadvantage is the lack of central control and a more complex model.

The presented approach can be beneficial in various applications out of which we describe four scenarios in Section 4. The first scenario shows how TDFs can be used to trace documents exchanged via email, e.g., for the use by a program chair who sends papers and review forms to program committee members. The second scenario describes the use of TDFs for collaborative authoring, e.g., when editing/writing a book. The third scenario shows how TDFs can be used as a light-weight infrastructure to support workflows, either to discover existing workflows (bottom-up) or to model workflows and constrain document flows (top-down). The last scenario shows how TDFs can be utilized to share papers in a scientific community, e.g., to share metadata about papers and determine domain experts.

The paper is organized as follows. Section 2 presents our model for traceable document flows. It comprises documents, versions, locations, and interactions thereon. In parallel, the ontological framework is defined. Section 3 describes the model's implementation. Section 4 presents application scenarios, exemplifying situations where users can benefit from using the presented approach. Section 5 discusses related work. Finally, Section 6 concludes the paper.

2. Infrastructure Model and Ontological Framework

This section presents the infrastructure model for TDFs, which employs versioning as known from configuration management [10]. To make instantiations of the model's concepts interoperable and thus available for other applications, we employ Semantic Web technology to express them, i.e., RDF and RDF Schema. We define an ontological framework for document flows comprising these concepts, which are used when describing instantiations. Being a framework, any other ontology can be used in addition, e.g., for annotating documents. The use of additional ontologies is influenced by the given application as is exemplified in Section 4.

2.1. Documents and Versions

A document is identified by a *document identifier* (DID) and keeps its identity across modifications. Since the model is intended to be implemented on the Web, a DID is a URI [14]. Between two subsequent modifications a document is represented by a static concrete occurrence having certain contents. In the presented model a document exists both as an abstract concept represented by its DID (referred to as *document*) and as set of concrete occurrences (referred to as *versions*).

A version is identified by a *version identifier* (VID) which is a URI. A modification to a version in the presented model does not necessarily result in a new version, i.e., a version's content may change over time. All versions of a document together represent its history. A version's VID is globally unique, i.e., unique across documents so that it suffices to identify and retrieve the version.

The underlying version model is organized as a two-level *acyclic version graph* (cf. [10]) that is composed of branches, each consisting of a sequence of versions. This model is employed in several version management systems such as WebDAV [15, 16] and CVS [2]. While versions constitute the nodes in the graph, three relationships between versions are distinguished, namely successor, offspring (starting a new branch), and predecessor. Each version has at most one successor and possibly several offsprings in different branches. Successor and offspring versions together are also referred to as following versions. Because a version can be merged into a version of another branch, a version can have several predecessors.

Additionally to the used concepts known from version models, we introduce the concepts of a current version and a frozen version. A *current version* is a version that has no succeeding version. Multiple current versions may exist per document in different branches. To provide for a consistent version graph, versions that are non-current can be *frozen* to prevent them from further being modified, avoiding that they become inconsistent with succeeding versions. Whether non-current versions are frozen is determined by configuration for each document separately (see below). By not freezing a version a very unconstrained exchange of versions can take place, because a succeeding version does not prevent the version from being modified.

Documents and versions are semantically described by several RDF properties. At minimum a document as well as a version must be described by properties df:title[2] and df:description, which are derived from the respective properties defined by the Dublin Core Element Set [17]. Each document is additionally described by property df:freezeNonCurrentVersions, which defines whether its non-current versions are frozen. Each version is additionally described by its document (property df:ofDocument), following versions (properties df:successorVersion and

2 prefix df is used for the document flow namespace `http://www.big.tuwien.ac.at/research/documentflows`

df:offspringVersion), and preceding versions (property df:precedingVersion). Of course, other ontologies can be used in addition to describe documents and versions, i.e., to describe a user's annotations.

2.2. Locations

Since a document as an abstract concept cannot be allocated itself, versions are allocated at *locations*. Each version is allocated at a single location, while one location can host multiple versions of the same document. Note that the concept of location is new to version models and thus makes the presented version model more expressive. Locations form the basis for modelling the distribution aspect of documents.

The meaning that is associated with a location can be very diverse. It is assumed, however, that a location is in a close relationship to a natural or juristic person such as a version's creator and owner. This relationship may also be "weaker", e.g., to persons who process a version or are simply interested in its contents. Usually, locations have autonomy in how to handle allocated versions, e.g., in how to store, to version, and to control access. Thus a modified relationship to a person, such as modified ownership, is likely to go in parallel with a modified location.

A location is represented by a document peer, short *peer*, that serves as a repository for versions, communicates with other peers, and performs interactions requested by users on versions. Peers may differ in the interactions they support, however, they must minimally support the interactions described in Section 2.3. A peer is identified by a *peer identifier* (PID) which is a URI.

A peer is semantically described by properties df:name and df:description. Moreover, property df:serves has a peer as its domain and class df:ServedEntity as its range. The latter is the root of a disjunct classification hierarchy with subclasses df:Person, df:JuristicalPerson, df:Group, and df:Role, which can be further specialized according to specific applications. The classification reflects addressees to whom document delivery in real life is targeted: persons, juristical persons such as companies or clubs, groups of persons such as departments of a company, and roles as in workflows. If a peer does not have the property, the entity it serves is unknown as it is the case with post office boxes.

To maintain a document's version graph, the dependencies between multiple versions, which are likely stored at different locations, have to be maintained. Following a P2P model, the version graph is maintained distributed. Because of the requirement of low coupling in a Web context, only references to directly preceding and following versions are maintained with each version stored at a peer. This decision is also influenced by the assumption that navigation and communication from a version to its directly preceding and following versions is more likely to be possible (e.g., by a firewall configuration) and will occur more often than therefrom to other versions, e.g., the initial one.

Having introduced locations, the rationale behind employing versioning can be refined which is different from the rationale in version management. While in the latter capturing modifications and identifying configurations is of primary concern, i.e., the *evolution aspect* of documents, the presented model employs versions to distinguish between and keep track of versions stored at different locations in a network, i.e., the *distribution aspect* of documents. Nevertheless, because the presented model is more expressive than known version models, it captures the evolution aspect as well.

A peer uses an RDF triple store to store semantic descriptions of occurrences of the concepts presented so far – documents, versions, version graphs, and peers – and of interactions (cf. Section 2.3). It can be queried about locally stored occurrences of these concepts, e.g., remote versions can be discovered by querying a local version to determine its preceding versions, offspring versions, and successor version. Aside of querying for other versions, semantic descriptions with application semantics of local versions can be queried as well.

The presented approach can well be combined with related approaches to provide enhanced functionality. First, for the replication of semantic descriptions which are currently only available locally, e.g., Edutella's replication service [20] can be used. Second, for answering queries that search for an arbitrary version (i.e., not by navigating, starting from a known version), existing discovery mechanisms, e.g., provided by Gnutella [4] or JXTA [18] can be used. Third, for the resolution of a DID to a VID, e.g., of the initial version or current versions, existing mechanisms for resolving location-independent identifiers, e.g., URNs, can be used (cf. [25] for an overview). Fourth, for querying semantic descriptions of a set of peers, related approaches such as Edutella [20] can be used.

2.3. Interactions on Documents and Versions

Because the presented model for TDFs is based on a version model, the interactions that are provided by peers mostly have counterparts in version management systems such as WebDAV and CVS, however, there are major differences. First, the presented model is richer by dealing with locations, and second, there is no central control. Thus the semantics of interactions with counterparts differ to cope with locations and decentralization. Moreover, interactions without counterparts are introduced.

Depending on a version's environment it is in a certain state. When being allocated at a peer, the version is *checked-in*. Using appropriate interactions, a following version can

be retrieved from a peer. The retrieved version has status *checked-out*. A checked-in version is *online* when the peer it is allocated at can communicate with other peers. Otherwise, or when the version is checked-out, it is *offline*. A checked-out version can be checked in using interaction `checkin`.

On checked-in version v_i basically three basic interactions can be performed. First, interaction `read` retrieves the version's content. Second, interaction `checkout` retrieves successor version v_{i+1} or offspring version $v_{i.j.1}$ when used with option `successor` or `offspring`, respectively. Third, checked-in version v_k that resides in another branch than v_i can be merged into v_i using interaction `merge`. Thereby, v_i becomes the successor of v_k. The merged versions may be allocated at different peers. Merging the versions' contents can be done manually or automatically by approaches such as [13].

When a version is checked out two files are retrieved from the peer its predecessor is allocated at. The first file contains the contents of the version and is thus called the *data file*. The second file comprises semantic descriptions concerning the version and the version graph (i.e., at least a single statement that uses the df:precedingVersion property). The file is thus called the *metadata file*. Semantic descriptions that are available for the checked-out version's predecessor are taken over to the metadata file of the checked-out version. Which data is taken over depends on (a) what the user performing the checkout is allowed to read from the data available for the checked-out version's preceding version, and (b) which data the user performing the checkout is interested in. For a checkin at a later date, both the data and the metadata file are necessary. The descriptions that were taken over are marked as such using appropriate statements.

Note that different to WebDAV and CVS the `checkout` interaction in our model retrieves a following version not a representation, thus creating a following version at checkout time instead of the time when the modified representation is checked-in at a later date. It is essential to check out a version, which has its own VID assigned to it, so that it can be recognized as a version by humans and machines, making it possible, e.g., to send it to other people via email and most important to make statements about it using an appropriate language, e.g., RDF. Moreover, if non-current versions are frozen, a checked-out succeeding version cannot become inconsistent with its predecessor. Whereby consistency means that the succeeding version cannot lack modifications that have been performed on its predecessor.

Aside of the interactions presented so far, which allow to construct version graphs, two interactions allow to modify version graphs. First, interaction `delete` removes a version from the version graph. Second, interaction `reallocate` stores a version at another peer which may involve modifying its VID (if the VID is a location dependent identifier). A user who requests to perform an interaction on a version which has been deleted or reallocated is notified of the modification. Both interactions have to be used with caution, since external references to the version become "broken". Semantic descriptions are updated upon deletion or reallocation, however, replications of that descriptions created by other applications become inconsistent.

A peer does not only store semantic descriptions of documents and versions it hosts, but also semantic descriptions of interactions it has performed. These descriptions comprise data about the user who performed an interaction, the date/time it was performed, and possibly a description in natural text. Additional application specific descriptions can be stored as well, e.g., if the interaction corresponds to some state change in a workflow description. While the effects of interactions `checkin`, `checkout`, and `merge` also manifest in the version graph, this is not entirely the case for `read`, `delete`, and `reallocate`. Descriptions of `read`s can be used, e.g., for recommendations, while descriptions of `delete`s and `reallocate`s can be used to inform users of deleted versions or to navigate to reallocated ones.

3. Implementation

We have implemented a prototype using JXTA as our P2P platform. Regarding identifiers, PIDs are URNs in the jxta namespace, DIDs and VIDs are UUIDs expressed as URNs. Thereby the prototype is independent of physical network addresses, i.e., peers and versions can be physically moved on a network without affecting their identifiers. The resolution of URNs to network addresses is provided by JXTA. Using the prototype, the user can navigate to preceding and following versions starting from any peer and any version, can retrieve more information including annotations about the version, and can read the version's contents. Because, among others, peers and versions are published using so called JXTA advertisements, they can be dynamically discovered using JXTA's Peer Discovery Protocol as well.

For a different user interface to peers and a seamless integration with standard email clients, we have implemented an SMTP-Filter and an IMAP/POP3-Agent, as shown in Figure 1. They are transparent to users and operate in a non-intrusive way. When a user wants to re-distribute a version to another user, she/he simply creates an email and attaches the version's data file to it. The SMTP-Filter checks out a following version on behalf of the user, thereby receiving the data- and metadata file from the peer. Subsequently it either sends the version to the recipient by email (alternative a in the figure) or checks it in directly at the

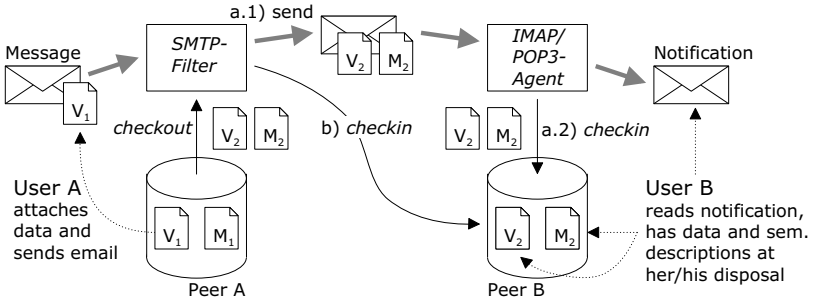

Figure 1. Sending and Receiving Traceable and Queryable Email

destination peer (alternative b). In the former case, the recipient's IMAP/POP3-Agent detects the received email and checks the version in. In either case the recipient is notified by email. Other user interface to peers, such as via HTTP and Web Services, are reasonable, but have not been implemented yet.

4. Application Scenarios

We present four exemplary application scenarios for TDFs. Namely, these are traceable email, collaborative authoring, workflows, and information exchange. In the second and the third scenario, it can safely be assumed that the contents of versions are not private and that people are willing to let others know where their documents originated, i.e., it is agreed that user specific data is available for others. In the first and the fourth scenario, however, privacy concerns may arise. There are several possibilities which can be used alone or in combination to resolve them: First, user data can be anonymized while still exploiting its value. This is possible by aggregating the metadata and providing statistical data interpretation, e.g., for features like "Users who read this document also read...". Second, one can implement authentification and authorization to specify and reveal what users are allowed to read. We do not go into details about privacy issues which is outside the focus of this paper.

4.1. Traceable and Queryable Email

The presented model for TDFs provides a non-proprietary possibility for storing and querying (also remote) annotations about documents that were received or sent as email attachments. Moreover, users can easily determine not only from whom a document was received, but also where it originally originated and who else was in possession of the document before. In addition, users can determine to which persons a document they have sent is redistributed further on.

An exemplary use of traceable email is the management of a paper review process. After submission, e.g., via a web application, which created a document and an initial version, the program chair checks out a following version of each paper and sends it (i.e., the data file) and the review form (i.e., the metadata file) to program committee members. Note that the review form is an annotation to the version, expressed according to an ontology for reviews. Committee members possibly forward received papers to additional (sub-) reviewers. After each reviewer has filled out his/her review form, he/she sends it back to the super-reviewer or to the chair where it is checked. This approach is very flexible and allows the chair to query for the status of each review at any time. Also an aggregated report summarizing all reviews of a paper can be easily created by querying the annotations.

Comparing traceable email with related work [11], the latter focusses on defining machine-processable email contents, while we focus on revealing a document's flow that is established via email.

4.2. Collaborative Authoring

As usual for version systems, our model can be employed for collaboratively editing documents. Naturally, the functionality of TDFs can be compared to CVS in this scenario. The main difference is that there is no need for a shared central document repository. In addition, the annotation mechanism is superior compared to the log facility of CVS for two reasons. First, it uses an open format that can be parsed with standard RDF tools. Second, it can be adapted to users' needs by employing additional ontologies.

For example, consider the complex collaborative task of writing an edited book. For each book chapter, there are multiple authors working together. Hence it is necessary to create a document for each chapter. For all of these documents, df:FreezeNonCurrentVersions is set to true. Thus only the author in possession of a current version owns the "edit token". She/he can edit the draft of the chapter and hand it on to a co-author afterwards. It is also possible

that authors work in parallel and merge their changes afterwards. At any moment, the editor can determine the status of each chapter. In addition, the authors can have a look at the current versions of the other chapters, e.g. to align their terminology or their references.

4.3. Workflows

In environments where a workflow management system (WfMS) is not available and/or where it is impossible to deploy one (e.g., when a shared, central WfMS cannot be negotiated) or where it is unreasonable to deploy a WfMS (e.g. too costly or time-consuming), TDFs are a light-weight alternative infrastructure for workflow management. These environments are faced more and more often with the increase of workflows crossing organizational boundaries.

Employed top-down, TDFs enables one for the definition of workflows in the form of document flows at the schema level. This can be done by using an appropriate ontology to describe document flows, which define where and in which order a document has to be processed. For example, user A (every user has its own peer) can instruct user B to proofread a document by handing it over to him, followed by user C who has to add a reference in a library catalog, and user D who has to give her permission for its publication (e.g., stored as signed annotation). At any time, users are able to query for subsequent steps and peers may restrict the document flow to a valid one, i.e., one that accords to its schema.

TDFs can also be used in a bottom-up way. Since the metadata describing document flows describes how versions were distributed across a network, it can be analyzed ex post to discover document flows that took place regularly. This way, workflow analysis can be carried out by investigating document flows.

4.4. Information Exchange

This scenario is comparable to well-known P2P filesharing applications like Kazaa [6]. The focus is on exchanging *static* documents, meaning that all versions of a document have the same content. Thus annotations are not longer specific for a single version of a document but for all of them. Here, the revelation of document flows, the possibility to annotate versions, and the possibility to query this metadata distinguishes TDFs from other filesharing applications. Of course users are not forced to annotate their documents, but decide by themselves how much time they spend for annotation.

Moreover, using TDFs previously unknown information can be discovered by exploiting document flow metadata. First, it is possible to find out *all* existing versions of a given document. The total number can then be used to rank documents. The outcome of a query determining all peers where a document's versions are stored at may form a list of potential contact addresses (e.g., persons). Most value can be gained from these features if the shared documents are related to a single domain.

An example of this application scenario is that of scientists working on the same subject. Typically, they all use a folder in their local file system to store and manage their collection of scientific papers related to their research field. If they use TDFs to share their folder and annotations with others, every user can benefit from observing which documents other users have read. E.g., one can determine researchers working in the same field, determine domain experts for a field, or simply read annotations of other users.

5. Related Work

Traceable document flows are new to P2P applications, which provide for distributed computing, file sharing, and online collaboration [19]. Nevertheless, techniques of existing approaches can be employed for TDFs, e.g., for version discovery using a central index as in [1], flooded requests as in [4], or distributed index structures using super-peers as in [3]. Approaches for directed routing using distributed hash tables (DHTs), which assign an identifier to a version based on a hash of its content and name and store it at peers with similar identifier (cf. [8, 19] for an overview), cannot be used for discovery because versions in our model cannot be allocated freely. The closest related P2P system is Storm [12], which makes documents identifiable by globally unique identifiers as we do, however, they provide for linking between documents and versioning of documents while we provide for document flows (using versioning) and annotations.

Concerning distributed concurrent versioning systems, the most prominent existing systems are CVSup [7] and DCVS [5]. However, both of them are based on the client/server-paradigm and provide repository replication and thus can not be compared to TDFs which are based a P2P model and provide distributed equitable repositories.

There are also approaches from office information systems in the 1980s that are related. They deal with form management, e.g., [22] presents among others forms that can flow through an organization. It differs from TDFs with respect to flowing artifacts (only proprietary forms can flow), the data model (versioning is not employed), functionality (annotations are not available), and the architecture (using central nodes).

There is a model for encoding semantic information in P2P networks, namely the SWAP metadata model [9] that takes a similar approach of annotating information with metadata about its origin. Unlike our approach which does not change the format of documents, all information is additionally converted to RDF representations. The SWAP

model assumes that data is not physically replicated between peers but rather queries are used for information exchange. Since document distribution is an important enabler for TDFs, this a a major difference. An interesting approach is the query routing algorithm REMINDIN' [21] designed for the SWAP platform. It uses observation of other peers' queries and answers to determine their domain knowledge and to decide who is the right peer to answer a certain query. An adaption of this algorithm for TDFs would be useful for the information exchange scenario which we presented in Section 4.4.

6. Conclusion

In this paper, we presented an infrastructure model and an ontological framework for semantically describing document flows. As we showed in Section 4, this light-weight communication infrastructure for information exchange and collaboration can be used in a variety of application scenarios. In addition, we demonstrated that semantic descriptions considerably improve collaboration.

References

[1] Napster Homepage. `http://napster.com`, 2001.

[2] Concurrent Versions System (CVS) Homepage. `http://cvshome.org`, 2003.

[3] FastTrack Homepage. `http://www.fasttrack.nu`, 2003.

[4] Gnutella Homepage. `http://rfc-gnutella.sourceforge.net`, 2003.

[5] Distributed Concurrent Versions System (DCVS) Homepage. `http://www.elegosoft.com/dcvs/`, 2004.

[6] Kazaa Homepage. `http://www.kazaa.com/`, 2004.

[7] The CVS-Optimized General-Purpose Network File Distribution System (CVSup) Homepage. `http://www.cvsup.org/`, 2004.

[8] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up Data in P2P Systems. *Communications of the ACM (CACM)*, 46(2), 2003.

[9] J. Broekstra, M. Ehrig, et al. A Metadata Model for Semantics-Based Peer-to-Peer Systems. In *Workshop on Semantics in Peer-to-Peer and Grid Computing at the WWW Conference*, 2003.

[10] R. Conradi and B. Westfechtel. Version Models and Software Configuration Management. *ACM Computing Surveys*, 30(2), 1998.

[11] O. Etzioni, A. Y. Halevy, et al. Semantic Email: Adding Lightweight Data Manipulation Capabilities to the Email Habitat. In *Workshop on Web and Databases (WebDB)*, 2003.

[12] B. Fallenstein, T. J. Lukka, et al. Storm: Using P2P to Make the Desktop Part of the Web. In *Hypertext and Hypermedia Conference*, 2004.

[13] R. La Fontaine. Merging XML Files: a New Approach Providing Intelligent Merge of XML Data Sets. In *XML Europe Conference*, 2002.

[14] IETF. Uniform Resource Identifiers (URI): Generic Syntax. `http://ietf.org/rfc/rfc2396.txt`, 1998.

[15] IETF. HTTP Extensions for Distributed Authoring – WebDAV. `http://www.ietf.org/rfc/rfc2518.txt`, 1999.

[16] IETF. Versioning Extensions to WebDAV. `http://www.ietf.org/rfc/rfc3253.txt`, 2002.

[17] Dublin Core Metadata Initiative. Dublin Core Metadata Element Set, Version 1.1: Reference Description. `http://dublincore.org/documents/dces/`, 2003.

[18] Sun Microsystems. JXTA. `http://jxta.org`, 2003.

[19] D. S. Milojicic, V. Kalogeraki, et al. Peer-to-Peer Computing. Tech. Rep. HPL-2002-57, HP Labs, 2002.

[20] W. Nejdl, B. Wolf, et al. EDUTELLA: a P2P Networking Infrastructure based on RDF. In *World Wide Web Conference*, 2002.

[21] C. Tempich, S. Staab, et al. REMINIDIN': Semantic Query Routing in P2P Networks based on Social Metaphors. In *World Wide Web Conference*, 2004.

[22] D. Tsichritzis. Form Management. *Communications of the ACM (CACM)*, 25(7), 1982.

[23] W3C. RDF Vocabulary Description Language 1.0: RDF Schema. `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`, 2004.

[24] W3C. Resource Description Framework (RDF): Concepts and Abstract Syntax. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`, 2004.

[25] T. Werf-Davelaar. Identification, Location and Versioning of Web-Resources. `http://www.kb.nl/coop/donor/rapporten/URI.html`, 1999.