P2P-DIET: Ad-hoc and Continuous Queries in Peer-to-Peer Networks using Mobile Agents*

Stratos Idreos and Manolis Koubarakis

Intelligent Systems Laboratory
Dept. of Electronic and Computer Engineering
Technical University of Crete
GR73100 Chania, Crete, Greece
{sidraios, manolis}@intelligence.tuc.gr

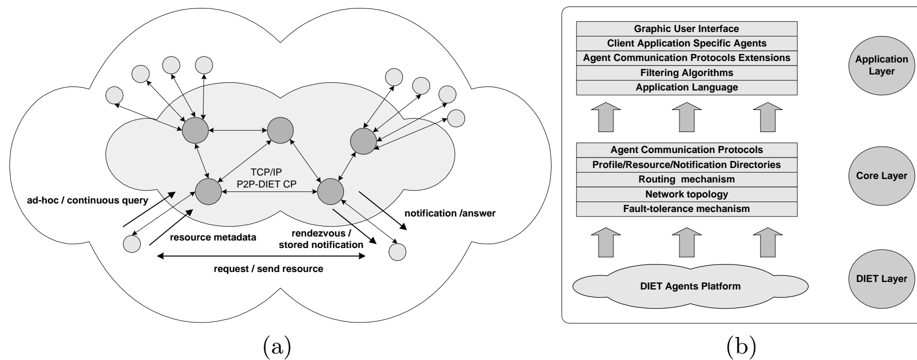
Abstract. This paper presents P2P-DIET, a resource sharing system that unifies ad-hoc and continuous query processing in super-peer networks using mobile agents. P2P-DIET offers a simple data model for the description of network resources based on attributes with values of type text. It also utilizes very efficient query processing algorithms based on indexing of resource metadata and queries. The capability of location-independent addressing is supported, which enables P2P-DIET clients to connect from anywhere in the network and use dynamic IP addresses. The features of stored notifications and rendezvous guarantee that all important information is delivered to interested clients even if they have been disconnected for some time. P2P-DIET has been developed on top of the Open Source mobile agent system DIET Agents and is currently been demonstrated as a file sharing application.

1 Introduction

In peer-to-peer (P2P) systems a very large number of autonomous computing nodes (the *peers*) pool together their resources and rely on each other for *data* and *services*. P2P systems are application level *virtual* or *overlay networks* that have emerged as a natural way to share data and resources. Popular P2P *data sharing* systems such as Napster, Gnutella, Freenet, KaZaA, Morpheus and others have made this model of interaction popular.

The main application scenario considered in recent P2P data sharing systems is that of *ad-hoc querying*: a user poses a query (e.g., "I want music by Moby") and the system returns a list of pointers to matching files owned by various peers in the network. Then, the user can go ahead and download files of interest. The complementary scenario of *selective information dissemination (SDI)* or *selective information push* [8] has so far been considered by few P2P systems [1, 10]. In an SDI scenario, a user posts a *continuous query* to the system to receive notifications whenever certain *resources* of interest appear in the system (e.g.,

* This work was carried out as part of the DIET project (IST-1999-10088), within the UIE initiative of the IST Programme of the European Commission.



(a) (b)

Fig. 1. The architecture and the layered view of P2P-DIET

when a song of Moby becomes available). SDI can be as useful as ad-hoc querying in many target applications of P2P networks ranging from file sharing, to more advanced applications such as alert systems for digital libraries, e-commerce networks etc.

At the Intelligent Systems Laboratory of the Technical University of Crete, we have recently concentrated on the problem of SDI in P2P networks in the context of project DIET[1]. Our work, summarized in [9], has culminated in the implementation of P2P-DIET, a service that unifies ad-hoc and continuous query processing in P2P networks with super-peers. Conceptually, P2P-DIET is a direct descendant of DIAS, a *d*istributed *i*nformation *a*lert *s*ystem for digital libraries, that was presented in [10] but was never implemented. P2P-DIET combines ad-hoc querying as found in other super-peer networks [2] and SDI as proposed in DIAS. P2P-DIET goes beyond DIAS in offering many new features discussed above: client migration, dynamic IP addresses, stored notifications and rendezvous, simple fault-tolerance mechanisms, message authentication and encryption. P2P-DIET has been implemented on top of the open source DIET Agents Platform[2][3] and it is currently available at `http://www.intelligence.tuc.gr/p2pdiet`. This paper concentrates on the architecture, functionality and agents of P2P-DIET.

A high-level view of the P2P-DIET architecture is shown in Figure 1 (a). There are two kinds of nodes: *super-peers* and *clients*. All super-peers are equal and have the same responsibilities, thus the super-peer subnetwork is a *pure* P2P network (it can be an arbitrary undirected graph). Each super-peer serves a fraction of the clients and keeps *indices* on the resources of those clients.

Clients can run on user computers. Resources (e.g., files in a file-sharing application) are kept at client nodes, although it is possible in special cases to store resources at super-peer nodes. Clients are equal to each other only in terms of download. Clients download resources directly from the resource owner client.

[1] `http://www.dfki.de/diet`
[2] `http://diet-agents.sourceforge.net/`

A client is connected to the network through a single super-peer node, which is the *access point* of the client. It is not necessary for a client to be connected to the same access point continuously since *client migration* is supported in P2P-DIET. Clients can connect, disconnect or even leave from the system silently at any time. To enable a higher degree of decentralization and dynamicity, we also allow clients to use *dynamic IP addresses*. Thus, a client is identified by an identifier and public key (created when a client bootstraps) and not by its IP-address. Super-peers the keep the client's identification information and resource metadata for a period of time when a client disconnects. In this way, the super-peer is able to answer queries matching those resource metadata even if the owner client is not on-line. Finally, P2P-DIET provides message authentication and message encryption using PGP technology. For details on the network protocols and implementation see [6].

The rest of the paper is organized as follows. Section 2 presents the metadata model and query language used for describing and querying resources in the current implementation of P2P-DIET. Section 3 discusses the protocols for processing queries, answers and notifications. Section 4 discusses other interesting functionalities of P2P-DIET. Section 5 discusses the implementation of P2P-DIET using mobile agents. Finally, Section 6 presents our conclusions.

2 Data models and query languages

In [10] we have presented the data model $\mathcal{AWPS}$, and its languages for specifying queries and *textual* resource metadata in SDI systems such as P2P-DIET. $\mathcal{AWPS}$ is based on the concept of *attributes* with values of type *text*. The query language of $\mathcal{AWPS}$ offers *Boolean* and *proximity operators* on attribute values as in the Boolean model of Information Retrieval (IR) [5]. It also allows textual *similarity* queries interpreted as in the vector space model of IR [11].

The current implementation of P2P-DIET supports only *conjunctive* queries in $\mathcal{AWPS}$. The following examples of such queries demonstrate the features of $\mathcal{AWPS}$ and its use in an SDI application for a digital library:

$$AUTHOR \sqsupseteq \textit{Smith} \ \wedge \ TITLE \sqsupseteq (\textit{peer-to-peer} \vee \\ (\textit{selective} \prec_{[0,0]} \textit{dissemination} \prec_{[0,3]} information))$$

$$AUTHOR \sqsupseteq \textit{Smith} \wedge \\ ABSTRACT \sim_{0.8} \textit{``Peer-to-peer architectures have been...''}$$

The data model $\mathcal{AWPS}$ is attractive for the representation of textual metadata since it offers linguistically motivated concepts such as *word* and traditional IR operators. Additionally, its query language is more expressive than the ones used in earlier SDI systems such as SIFT [11] where documents are free text and queries are conjunctions of keywords. On the other hand, $\mathcal{AWPS}$ can only model resource metadata that has a *flat* structure, thus it cannot support hierarchical documents as in the XML-based models of [4]. But notice that IR-inspired constructs such as proximity and similarity *cannot* be expressed in the query languages of [4] and are also *missing* from W3C standard XML query languages

XQuery/XPath. The recent W3C working draft[3] is expected to pave the way for the introduction of such features in XQuery/XPath. Thus our work on $\mathcal{AWPS}$ can be seen as a first step in the introduction of IR features in XML-based frameworks for SDI.

3 Routing and Query Processing

P2P-DIET targets content sharing applications such as digital libraries [10], networks of learning repositories [12] and so on. Assuming that these applications are supported by P2P-DIET, there will be a stakeholder (e.g., a content provider such as Akamai) with an interest in building and maintaining the super-peer subnetwork. Thus super-peer subnetworks in P2P-DIET are expected to be more stable than typical pure P2P networks such as Gnutella. As a result, we have chosen to use routing algorithms appropriate for such networks.

P2P-DIET implements routing of queries (ad-hoc or continuous) by utilizing *minimum weight spanning trees* for the super-peer subnetwork, a *poset* data structure encoding continuous query subsumption as originally suggested in [1], and *data* and *query indexing* at each super-peer node. Answers and notifications are unicasted through the shortest path that connects two super-peers.

3.1 Ad-hoc querying

P2P-DIET supports the typical *ad-hoc query scenario*. A client A can post a query q to its access point AP. AP broadcasts q to all super-peers through its minimum weight spanning tree. *Answers* are produced for all matching network resources and are returned to the access point AP that originated the query through the shortest path that connects the super peer that generated the answer with AP (unicasting). Finally, AP passes the answers to A for further processing. Answers are produced for all matching resources regardless of whether owning resource clients are on-line or not, since super-peers do not erase resource metadata when clients disconnect (see Section 4).

Each super-peer can be understood to store a relation

$$resource(ID, A_1, A_2, \ldots, A_n)$$

where ID is a resource identifier and $A_1, A_2, \ldots, A_n$ are the attributes known to the super-peer network. In our implementation, relation *resource* is implemented by keeping an *inverted file index* for each attribute A_i. The index maps every word w in the vocabulary of A_i to the set of resource IDs that contain word w in their attribute A_i. Query evaluation at each super-peer is then implemented efficiently by utilizing these indices in the standard way.

[3] `http://www.w3.org/TR/xmlquery-full-text-use-cases`

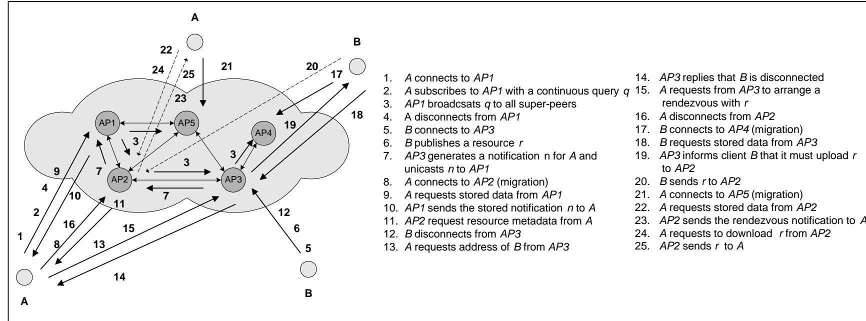


Fig. 2. A stored notification and rendezvous example

3.2 Continuous queries

SDI scenarios are also supported. Clients may *subscribe* to their access point with a *continuous query* expressing their information needs. Super-peers then *forward* posted queries to other super-peers. In this way, matching a query with metadata of a published resource takes place at a super-peer that is *as close as possible* to the origin of the resource.

Whenever a resource is published, P2P-DIET makes sure that all clients with continuous queries matching this resource's metadata are notified. Notifications are generated at the access point where the resource was published, and travel to the access point of every client that has posted a continuous query matching this notification following the *reverse path* that was set by the propagation of the query.

We expect P2P-DIET networks to *scale* to very large numbers of clients, published resources and continuous queries. To achieve this, we utilize the following data structures at each super-peer:

- A *partially ordered set* (called the *continuous query poset*) that keeps track of the subsumption relations among the continuous queries posted to the super-peer by its clients or forwarded by other super-peers. This poset is inspired by SIENA [1]. We can also have it in P2P-DIET because the relation of subsumption in $\mathcal{AWPS}$ is reflexive, anti-symmetric and transitive i.e., a *(weak) partial order*. Like in SIENA, P2P-DIET utilizes the continuous query poset to to minimize network traffic: in each super-peer no continuous query that is less general than one that has already been processed is actually forwarded.
- A sophisticated *index* over the continuous queries managed by the super-peer. This index is used to solve the *filtering problem*: Given a database of continuous queries db and a notification n, find all queries $q \in db$ that match n and forward n to the neighbouring super-peers or clients that have posted q.

4 Stored notifications and rendezvous at super-peers

Clients may not be online all the time, thus we can not guarantee that a client with a specific continuous query will be available at the time that matching resources are added to the network and relevant notifications are generated. Motivated by our target applications (e.g., digital libraries or networks of learning repositories), we do not want to ignore such situations and allow the loss of relevant notifications.

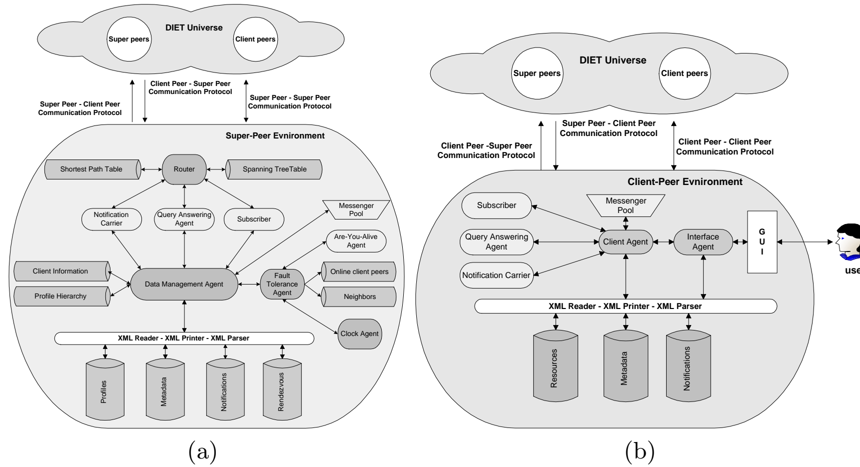
Assume that a client A is off-line when a notification n matching its continuous query is generated and arrives to its access point AP. AP checks if A is on the active client list. If this is true then n is forwarded to A, otherwise n is stored in the *stored notifications directory* of AP. Notification n is delivered to A by AP next time A connects to the network.

A client may request a resource at the time that it receives a notification n, or later on using a saved notification n on his local *notifications directory.* Consider the case when a client A requests a resource r, but the resource owner client B is not on-line. A requests the address of B from *AP2* (the access point of B). A may request a *rendezvous* with resource r from *AP2* with a message that contains the identifiers of A and B, the address of AP and the path of r. When B reconnects, *AP2* informs B that it must upload r to AP as a *rendezvous file* for A. Then, B uploads r. AP checks if A is on-line and if it is, AP forwards r to A or else r is stored in the *rendezvous directory* of AP and when A reconnects, it receives a rendezvous notification from AP.

The features of stored notifications and rendezvous take place even if clients *migrate* to different access points. For example, let us assume that A has migrated to *AP3*. The client agent understands that, and requests from AP any rendezvous or notifications. A updates the variable *previous access point* with the address of *AP3*. AP deletes A form its client list, removes all resource metadata of A from the local resource metadata database and removes the continuous queries of A from the poset. Finally, A sends to *AP3* its resource metadata and continuous queries. A complete example is shown in Figure 2.

5 Agents of P2P-DIET

The implementation of P2P-DIET makes a rather simple use of the DIET Agents concepts *environment*, *world* and *infohabitant.* Each super-peer and each client occupy a different world, and each such world consists of a single *environment.* All the worlds together form the *P2P-DIET universe.* However, the P2P-DIET implementation makes heavy use of all the capabilities of lightweight mobile agents offered by the platform to implement the various P2P protocols. Such capabilities are agent creation, cloning and destruction, agent migration, local and remote communication between agents etc.



(a) (b)

Fig. 3. A Super-Peer Environment and a Client Peer Environment

5.1 The super-peer environment

A world in a super-peer node consists of a single *super-peer environment*, where 10 different types of agents live. A super-peer environment is shown in Figure 3 (a).

The *data management agent* is the agent with the greatest number of responsibilities in the P2P-DIET universe. This agent manages the local database of peer meta-data, resource meta-data, continuous queries and their indices. Moreover, it arranges rendezvous and stores notifications and rendezvous files. The data management agent can create notification carriers and messengers (these agents will carry out the tasks discussed below).

The *router* is responsible for the correct flow of messages in the network. It holds the shortest path table and the spanning tree table of the local super-peer. Mobile agents travel around the network using information from the local Router on each super-peer environment where they arrive. The *make routing paths scheduler* is a very lightweight agent that decides when it is the right time for the router to update its routing paths when the network is in an unstable condition.

A *subscriber* is a mobile agent that is responsible for subscribing continuous queries of clients to super-peers. To subscribe a continuous query q of client C, a subscriber S starts from the environment of C. Then, it migrates to all super-peers of the network to subscribe q and to find any resource meta-data published earlier that match q. S will start from the super-peer environment of the access point A of C and it will reach all super-peers through the minimum-weight spanning tree of A. Whenever a subscriber finds any resource metadata matching its continuous query in a super-peer environment B, it clones itself. The clone returns to the environment of C to deliver the notification by travelling towards super-peer A through the shortest path that connects B and A. The

original subscriber will continue in order to visit the rest of the super-peers of the network. A subscriber agent destroys itself when it returns to the client-peer environment with a notification. A subscriber can also destroy itself away from its starting client environment when it is on a remote super-peer environment with no notifications to deliver and no more super-peers to visit.

A *notification carrier* is a mobile agent that is responsible for delivering notifications. A notification carrier may start from a super-peer environment SP and travel along the shortest path to the environment of super-peer AP and from there migrates to the environment of client C if C is online. Note, that it is not possible for the notification carrier to travel directly to the client environment for two reasons. First, the super-peer SP does not know the IP address of C. Second, the notification must arrive to environment AP because more than one clients may have continuous queries that match the notification. A notification carrier destroys itself after it has delivered the notification to the client or when it arrives to the access point of the client and the client is not online.

A *query answering agent* is a mobile agent that answers queries. The query-answering agent that finds the answers to query q of client C starts from the environment of C. Then, it migrates to all super-peers of the network to search for answers to q. It will start from the super-peer environment of the access point A of C and it will reach all super-peers through the minimum-weight spanning tree of A. Each time, it finds any resource metadata matching q in a super-peer environment B, it clones itself. The clone returns to the environment of client C to deliver the answer by travelling towards super-peer A through the shortest path that connects B and A. The original query-answering agent will continue in order to visit the rest of the super-peers of the network to search for answers to q. A query-answering agent destroys itself when it returns to the client-peer environment with an answer. A special case that a query-answering agent will destroy itself away from its starting client-peer environment, is when it is on a remote super-peer environment with no answers and no more super-peers to visit.

A *messenger* is a mobile agent that implements remote communication between agents in different worlds. A messenger is a very lightweight agent that will migrate to a remote environment and deliver a message to a target agent. We need Messengers to support simple jobs i.e, just send a message to a remote agent. For example, consider the case that a client agent sends a `connect` or `disconnect` message to its access point. In this way, we do not create a new type of agent for each simple job that is carried out by our system. Messengers can be used to support new simple features that require remote communication. Each environment has a messenger pool. When a messenger arrives at an environment, it delivers the message and stays in the pool, if there is space. In this way, when an agent wants to send a remote message, it assigns the message to a messenger from the pool unless the pool is empty, in which case a new messenger will be created.

Subscribers, notification carriers, query-answering agents and messengers use information from the local router on each super-peer environment that they

arrive in order to find the address of their next destination. They use shortest paths and minimum weight spanning trees to travel in the pure peer-to-peer network of super-peers. In this way, they may ask two types of questions to a router:

- I want to migrate to all super-peers and I started from super-peer X. The answer of the local router to this question are the IP addresses of remote the super-peers that are children in the minimum weight spanning tree of the local super-peer.
- I want to migrate to super-peer X. The answer of the local router to this question is the IP address of the remote super-peer that is its neighbor and is in the shortest path from the local super-peer to the destination super-peer X.

The *fault-tolerance agent* is responsible for periodically checking the clients agents, that are supposed to be alive and are served by this super-peer. The agents of the neighbor super-peers are checked too, to guarantee connectivity. The fault-tolerance agent can create are-you-alive agents. A useful heuristic in P2P-DIET, is that a fault-tolerance agent does not check a node x (client or super-peer) if x has sent any kind of message during the last period of checks (there is no need to ensure that x is alive in this case). An *are-you-alive agent* is a mobile agent that is sent by the fault-tolerance agent to a remote client-peer environment or super-peer environment to check whether the local agents are alive or not. An are-you-alive agent will return to its original environment with the answer. In each super-peer environment there is an are-you-alive agent pool where agents wait for the local fault-tolerance agent to assign them a remote environment to check. All are-you-alive agents return to their original environment to inform the local fault-tolerance agent on the status of the remote environment that they checked and then they stay in the local are-you-alive agent pool. The *clock agent* is the scheduler for the fault-tolerance agent. It decides when it is the right time to send messages or to check for replies.

5.2 The client-peer environment

The world in the client-peer nodes has a *client-peer environment*. The *client agent* is the agent that connects the client-peer environment with the rest of the P2P-DIET universe. It communicates, through mobile agents, with the super-peer agent that is the access point or any other remote client agents. The client agent sends the following data to the remote super-peer agent of the access point: the continuous query of the client, the metadata of the resources, the queries, the requests for rendezvous etc. The client agent can create Subscribers, query-answering agents and messengers. Figure 3 (b), shows all the agents in the client-peer environment. Additionally, an *interface agent* is responsible for forwarding the demands of the user to the client agent and messages from the client agent to the user. A *messenger*, *query-answering agent*, *notification carrier* and *subscriber* may inhabit a client-peer environment and are exactly the same as the agents that inhabit the super-peer environments.

6 Conclusions

We have presented the design of P2P-DIET, a resource sharing system that unifies ad-hoc and continuous query processing in P2P networks with super-peers. P2P-DIET has been implemented using the mobile agent system DIET Agents and has demonstrated the use of mobile agent systems for the implementation of P2P applications. Currently we are working on implementing the query and SDI functionality of P2P-DIET on top of a distributed hash table like Chord [7] and compare this with our current implementation. We are also working on more expressive resource description and query languages e.g., such as the ones based on RDF and currently used in EDUTELLA [12].

References

1. A. Carzaniga and D. S. Rosenblum and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM TCS*, 19(3):332–383, August 2001.
2. B. Yang and H. Garcia-Molina. Designing a super-peer network. In *Proc. of ICDE 2003*, March 5–8 2003.
3. C. Hoile and F. Wang and E. Bonsma and P. Marrow. Core specification and experiments in diet: a decentralised ecosystem-inspired mobile agent system. In *Proc. of AAMAS 2002*, pages 623–630, July 15–19 2002.
4. C.-Y. Chan, P. Felber, M. Garofalakis, and R. Rastogi. Efficient Filtering of XML Documents with XPath Expressions. In *Proc. of ICDE 2002*, pages 235–244, February 2002.
5. C.-C. K. Chang, H. Garcia-Molina, and A. Paepcke. Predicate Rewriting for Translating Boolean Queries in a Heterogeneous Information System. *ACM TIS*, 17(1):1–39, 1999.
6. S. Idreos and M. Koubarakis. P2P-DIET: A Query and Notification Service Based on Mobile Agents for Rapid Implementation of P2P Applications. Technical Report TR-TUC-ISL-2003-01, Intelligent Systems Laboratory, Dept. of Electronic and Computer Engineering, Technical University of Crete, June 2003.
7. I. Stoica and R. Morris and D. Karger and M. F. Kaashoek and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of ACM SIGCOMM 2001*, San Diego, California, August 2001.
8. M. J. Franklin and S. B. Zdonik. "Data In Your Face": Push Technology in Perspective. In *Proc. ACM SIGMOD 1998*, pages 516–519, 1998.
9. M. Koubarakis and C. Tryfonopoulos and S. Idreos and Y. Drougas. Selective Information Dissemination in P2P Networks: Problems and Solutions. *ACM SIGMOD Record, Special issue on Peer-to-Peer Data Management, K. Aberer (editor)*, 32(3), September 2003.
10. M. Koubarakis and T. Koutris and P. Raftopoulou and C. Tryfonopoulos. Information Alert in Distributed Digital Libraries: The Models, Languages and Architecture of DIAS. In *Proc. of ECDL 2002*, volume 2458 of *Lecture Notes in Computer Science*, pages 527–542, September 2002.
11. T.W. Yan and H. Garcia-Molina. The SIFT information dissemination system. *ACM TDS*, 24(4):529–565, 1999.
12. W. Nejdl and B. Wolf and Changtao Qu and S. Decker and M. Sintek and A. Naeve and M. Nilsson and M. Palmer and T. Risch. Edutella: A P2P Networking Infrastructure Based on RDF. In *Proc. of WWW-2002*. ACM Press, 2002.