Integrated Project

Priority 2.4.7

Semantic based knowledge systems

The Social Semantic Desktop

NEPOMUK

Distributed Search System - Basic Infrastructure

Deliverable D4.1

Version 1.1
28.12.2006
Dissemination level: PU

Nature	Prototype
Due date	31.12.2006
Lead contractor	EPFL
Start date of project	01.01.2006
Duration	36 months

Authors

Vasilios Darlagiannis, Ecole Polytechnique Fédérale de Lausanne (EPFL)
Roman Schmidt, Ecole Polytechnique Fédérale de Lausanne (EPFL)
Renault John, Ecole Polytechnique Fédérale de Lausanne (EPFL)
Ekaterini Ioannou, Forschungszentrum L3S

Mentors

Leo Sauermann, Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI)
Mikhail Kotelnikov, CogniumSystems
Stéphane Lauriere, Edge-IT S.A.R.L
Dr. Thomas Roth-Berghofer, TU Kaiserslautern

Project Co-ordinator

Dr. Ansgar Bernardi
German Research Center for Artificial Intelligence (DFKI) GmbH
Erwin-Schroedinger-Strasse (Building 57)
D 67663 Kaiserslautern
Germany
Email: bernardi@dfki.uni-kl.de, phone: +49 631 205 3582, fax: +49 631 205 4910

Partners

DEUTSCHES FORSCHUNGSZENTRUM F. KUENSTLICHE INTELLIGENZ GMBH
IBM IRELAND PRODUCT DISTRIBUTION LIMITED
SAP AG
HEWLETT PACKARD GALWAY LTD
THALES S.A.
PRC GROUP - THE MANAGEMENT HOUSE S.A.
EDGE-IT S.A.R.L
COGNIUM SYSTEMS S.A.
NATIONAL UNIVERSITY OF IRELAND, GALWAY
ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE
FORSCHUNGSZENTRUM INFORMATIK AN DER UNIVERSITAET KARLSRUHE
UNIVERSITAET HANNOVER
INSTITUTE OF COMMUNICATION AND COMPUTER SYSTEMS
KUNGLIGA TEKNISKA HOEGSKOLAN
UNIVERSITA DELLA SVIZZERA ITALIANA
IRION MANAGEMENT CONSULTING GMBH

Versions

Version	Date	Reason
0.10	10.10.2006	First draft
0.15	24.10.2006	Related work input
0.20	24.10.2006	Requirements input
0.25	21.11.2006	P-Grid concept
0.30	23.11.2006	DI component
0.35	25.11.2006	Social metadata interactions
0.40	27.11.2006	Further related work input
0.45	29.11.2006	Introduction and executive summary
0.50	01.12.2006	Refinement of related work
0.55	06.12.2006	Refinement of requirements
0.60	07.12.2006	Evaluation
0.65	08.12.2006	Conclusions
0.70	20.12.2006	Mentors' feedback
1.00	22.12.2006	Final version
1.10	28.12.2006	Version for saubmission (M. Junker)

Explanations of abbreviations on front page

Nature
R: Report
P: Prototype
R/P: Report and Prototype
O: Other

Dissemination level
PU: Public
PP: Restricted to other FP6 participants
RE: Restricted to specified group
CO: Confidential, only for NEPOMUK partners

Executive summary

The goal of this document is to describe the basic infrastructure of the Distributed Search System, which is realized by the Distributed Index (DI) component. It focuses on the functionality available in the first Nepomuk prototype, which will be later extended by additional operations to cover the mandatory functional requirements.

More particularly, the requirement of finding information in remote desktops raised the need of a component to perform distributed search inside a group of Nepomuk users. DI is the identified component in the Nepomuk architecture for performing this task. The implemented component is based on P-Grid, a highly scalable structured overlay network. Nevertheless, a number of state-of-the-art approaches have been investigated and evaluated in their ability to meet the functional and non-functional requirements of the Nepomuk case studies.

Therefore, Section 2 provides the identified functional and non-functional requirements. This set of requirements is the result of analyzing the case studies and gathering information based on a detailed questionnaire distributed to the responsible partners. Moreover, commonly deployed P2P applications have been evaluated to extract additional requirements for the targeted system. However, meeting the complete set of non-functional requirements is very challenging, since many conflicts arise and trade-offs are in place.

Section 3 evaluates the state-of-the-art solutions for distributed search. More specifically, a number of P2P overlay networks have been designed aiming in providing a number of features such as scalability, fault-tolerance, load-balancing, etc. Some well-known approaches are Chord, Pastry, CAN, Gnutella, Edutella, RDFGrowth and Freenet, which are shortly described. However, P-Grid has certain advantages over them, i.e., its hybrid architecture combining unstructured and structured overlays, preserving key-order, which is essential for complex search operations such as range queries and similarity queries as well as great adaptability to several environment conditions, i.e., churn rate. Therefore, it has been selected for implementing Nepomuk's distributed search task.

Section 4 describes the API (defined in WSDL) and the provided functionality of the DI component. Moreover, it discusses the interactions with the other Nepomuk components. Afterwards, Section 4.3 provides important information on the core concepts, the architecture and the implementation of P-Grid. A preliminary evaluation is given too, based on some locally performed experiments. Section 4.5 investigates deeper the nature and the context of the exchanged social metadata. Storage, ontologies and instances are discussed in further detail, as they have been identified by the personal workspace model developed in WP2000.

Summarizing, the complete followed procedure of identifying the requirements, evaluating the existing solutions, defining the interactions and developing the first prototype for distributed search are given in this document. The preliminary results show promising performance for the Nepomuk case studies.

Table of contents

1 Introduction

The social dimension of the Nepomuk project (The Social Semantic Desktop) introduces the requirement for rich user interactions in several aspects. Therefore, it is mandatory to provide the ability of searching for users and remote resources. The goal of this document is to describe the basic infrastructure of the Distributed Search System, which is realized by the Distributed Index (DI) component. It aims at identifying the critical requirements and their trade-offs, evaluating the existing systems and their appropriateness for Nepomuk, describing the details of the DI component and providing the concepts, the architecture and the implementation details of P-Grid.

In order to motivate for the technical content of this work, we examine some basic facts of the distributed systems. Typically, distributed system architectures follow two basic paradigms: (i) Client/Server (C/S), or (ii) Peer-to-Peer (P2P). In the C/S paradigm, participants are assigned to unambiguous roles of either service consumers (clients) or service providers (servers). In the P2P paradigm, the participants have potentially equivalent responsibilities and they may concurrently act both as service consumers and providers.

The limited physical capabilities of any server poses restrictions on the number of clients that can be served, making servers critical bottlenecks as the number of Internet users increases considerably. Moreover, the centralized nature of the C/S paradigm makes it an easier target for distributed Denial of Service (DoS) attacks, as opposed to the P2P alternative. Also, authorities may apply certain censorship policies to the provided services of centralized systems. In contrast, such an attempt in P2P approaches is significantly more difficult. Further, today's typical users are equipped with very powerful machines compared to the past. Such end-point devices are even capable of providing demanding services to a certain extend. Exploiting their integration in service provisioning can decrease significantly the service cost. Harnessing the power of such devices has been the focus of a large part of the distributed systems research community [Ora01].

Following a centralized approach has many disadvantages, especially for supporting large communities, such as the Mandriva Linux club, one of Nepomuk's case studies. A decentralized approach offers better opportunities in meeting the raised requirements of this complex system.

However, distributed systems adopting the P2P paradigm pose significantly more complex communication needs than their C/S counterparts. Many deployment difficulties, e.g., the presence of end-point devices with heterogeneous network, storage and computing capabilities make the development of P2P systems more challenging. More importantly, fundamental difficulties are introduced by three factors: (i) asynchronous concurrent events, (ii) limited local knowledge and (iii) arbitrary failures. In addition, multi-party services raise the need for flexible communication patterns among the participating entities. In order to address the aforementioned issues, P2P systems develop dedicated virtual networks, the so-called overlay networks, on top of the physical telecommunication networks [LCP$^+$04] [CF04]. Overlay networks are a mandatory abstraction from physical networks to both flexibly fulfill functional requirements such as connectivity maintenance, indexing and routing, as well as to satisfy non-functional requirements, such as scalability, fault-tolerance and load-balancing.

However, designing such overlay networks is a challenging task. There is a set of requirements that has to be met in designing overlay networks, which may be effortlessly deployed on top of the physical network while making use of available resources optimally. During the remaining of this document, it will become clear which these requirements are, in which aspects existing work fails to meet them, and which the proposed solution is, together with a preliminary evaluation. The provided functionality will offer the ability of finding available users or resources to the core Nepomuk components.

Accounting for the aforementioned challenges, P-Grid has been selected as a solid infrastructure over which to develop the Nepomuk system. The wide range of requirements coming from several independent case studies raise the need for adaptability to different situations. P-Grid can perform efficiently in stable environments where peers do not behave very dynamically (high churn rate) by reaching the theoretical optimal bounds of lookup operations. In addition, P-Grid can adapt to much more dynamic environments by utilizing the low maintenance cost of the existing unstructured substrate to query for the available resources, thus revealing the need to maintain the structured topology and the distributed index. The existence of heterogeneity can be well exploited with the introduction of super-peers. This adaptability process is transparent to the other Nepomuk components and the developed applications.

Addressing the needs for communication in Nepomuk and bringing together the social aspect with the technical aspects of semantic desktops, this work will provide rich functionality to users as well as the other components of the Nepomuk architecture. For example, the identified personas, e.g. Dirk will be able to perform rich queries on the Nepomuk network. He will be able to set ranges for the attributes of interest or consider rich metadata and perform such operations on a large scale network. In the following section, an overview of Nepomuk use cases and several persona scenarios describe in detail which is the required functionality that is later addressed with the provided architecture and implemented by the described prototype. A preliminary evaluation gives promising expectations for larger scale deployment and richer functionality in the following years.

2 Requirements

2.1 Use Cases and Case Study Scenarios

In the Nepomuk project, four case studies are under investigation for applying the resulting tools and mechanisms: (i) Bioscience Case Study (WP8000), (ii) Professional Business Services Case Study (WP9000), (iii) Organizational Knowledge Management Case Study (WP10000) and (iv) Mandriva Linux Community Case Study.

The objectives of the first case study are the adoption, application and validation of the Nepomuk system for biomedical research. The case study takes place in the environment of Institut Pasteur.

The objectives of the second case study are the adoption, application and validation of the Nepomuk system in the work context of knowledge workers in an international group of consulting companies, as it is facilitated and supported by TMI/ICCS.

The third case study focuses on the particular demands that result from the software research and development process at SAP and generally in large organizations. This process is characterized by high complexity of dependencies between the stages of this process and the high amount of required communication and coordination efforts. To deal with this complexity, the participants in this process rely on tools that help them to accomplish their individual tasks as part of the whole.

The objectives of the fourth case study are the adoption, application and validation of the Nepomuk system in the context on an active open-source online community. It will equip the on-line community of Mandriva club members with a tool of a new generation for sharing knowledge related to the open-source Mandriva-Linux project.

The requirements identified by these case studies shall be considered in the integrated P2P infrastructure. However, it is mostly the requirements of the forth case study that raise the most challenging requirements as they deal with a large open user community. The focus is both on functional requirements (common core operations and services that should be provided to the applications) as well as non-functional requirements. Apparently, it is the latter type of requirements that raise challenging issues in meeting them and addressing their (frequently) conflicting nature.

The following subsections will present our procedure to identify functional and non-functional requirements from the aforementioned case studies. A questionnaire covering the technical aspects and scenarios describing typical activities in the context of the case studies are described first before we present extracted requirements. Additionally, we will present requirements discussed in the literature of distributed systems to complete the set of requirements concerning the Distributed Search and Storage component of Nepomuk.

2.1.1 Questionnaire

The questionnaire filled-out by all case study partners and some technical partners, using the DI component directly from their components, covers five different aspects of distributed information systems: (i) the data model, (ii) manipulation and querying, (iii) data management, (iv) data access and (v) network environment.

Data Model

First of all, it is important to know what kind of data users of a distributed information system want to share and find. Therefore, the questionnaire started with questions regarding the data structure of the inserted data, e.g., unstructured, attribute-value pairs, XML, RDF/S, binary data, etc. Additionally to the type of shared data, it is important to know how the data should be indexed and later retrievable by other users, e.g., documents can found by a unique identifier, by their filename, or by additional meta-data added by the user before inserting.

The answers showed that users currently share more or less any kind of data from unstructured statements which should be annotated in the future as supported by Nepomuk, over XML-based documents to binary files such as Microsoft Office files, PDFs and any other kind of binary files. Further, users want to be able to find any kind of information in the shared documents, independent of their file type, requiring a full-text index over all shared documents. On the other hand, a unique and persistent identifier for shared objects seems to be essential for the Bioscience case study.

Manipulation and Querying

This section contains questions regarding which query language is desired by users to search for data in the distributed index and to probably update data already maintained there. The spectrum of possible query languages ranges from simple keyword based queries to full-fledged query languages such as SQL, additionally providing data manipulation operations. If it is not already part of the query language, the need for additional result operations, e.g., ranking operations such as top-k, can be requested. Finally, users are asked if they need query reformulation capabilities, e.g., to reformulate a query over heterogeneous but semantically related schemas/ontologies.

Users intend to issue mainly simple keyword-based queries but also more complex queries should be supported by the DI component, such as predicative search, conjunctive queries up to SQL (SPARQL) queries. As probabilistic data will likely also be stored in the DI component, inference on this data should be supported as well. Top-k queries are of interest for most of the Case Studies. We further identified the need for query reformulation and similarity search, e.g., users want to be able to find all cars with a certain name even if they are annotated under the concept of vehicle.

Data Management

This part of the questionnaire deals with the problems of data management, i.e., how and how much data is inserted into the system, how long it is expected to remain there and how often data modifications are foreseen. We were concerned about the number of data items inserted into the system as well as the physical size of each of them. The number of data items is especially interesting for the indexing part of DI whereas the size of data items is important for the storage and replication of those items in view of future extensions of DI to support data storage. Together with expected insert and update frequencies, users were asked about their consistency requirements for those data modification operations.

The gathered responses indicate that all case studies will first insert a large number of data at system startup and then systematically add data during the lifetime of the network. Though the insert operation will be most frequent one, it should be possible to update and even delete data. Instead of physically removing data from the system, it may remain in the system marked as deprecated. Operations will be executed at the beginning at lower frequencies, e.g., a query per minute and an update per hour, but might increase fast with growing community size. The DI component will have to deal with up to millions of entries for files of varying sizes, from small statements to large attachments. Modifications of data should be visible after a reasonable delay whereby shorter delays are tolerable for data shared between local collaborators. Those changes should probably be visible even immediately. The last strong requirement concerns the data itself, as it should be replicated and kept available in the system independent of users' behavior, i.e., independent if users are online or offline.

Data Access

Next, we tried to identify user requirements regarding access restrictions for queries, updates, and deletions on a user or group level demanding a distributed or centralized user/group management. Along with access restrictions, we investigated the need for data encryption of index information and/or data itself in the future. This is different from the first part as structured overlays usually spread the index information in the network enabling participating peers to see and modify the local part of the index if it is not encrypted.

The received answers made clear that access restrictions are essential in the case studies, either based on users/groups or role-based. The experience of users showed that if access rights can be given at a fine-grained level, the acceptance to share information is usually higher. For small working groups, access restrictions should be defined and maintained distributed whereas a centralized solution would be preferred on a company or community scale, e.g., using LDAP. The Bioscience case study further stressed the need for data encryption, both of index information and data itself, whereas encryption is less important for others.

Network Environment

The last part of the questionnaire investigated the expected network environment the DI has to run in each of the Case Studies. This includes on the one hand the number of peers participating and on the other hand their properties, e.g., if mainly servers running all the time will form the network or mainly laptops from users joining and leaving the network frequently. Independent of which type of computers participate in the network, they might have limiting factors such as low bandwidth connections, located behind a firewall or connecting to the Internet using a dynamic IP address. At last, the expected user behavior was of interest for us, especially how many users will insert and query the system and with which frequency, as most users in P2P systems are free-riders and data is usually only provided by a small fraction of peers.

The Distributed Search and Storage component will initially be formed by around 100 peers including long running servers and laptops from users leaving and joining the network frequently. Users working at home or while traveling would still like to access data of their collaborators and are likely to be behind a firewall, using a dynamic IP address and may even have a slow Internet connection. In the future, larger networks consisting of around or even more than 100.000 peers are expected also increasing the query frequency from initially only one query per minute. Depending on the performed operation, users are willing to wait longer for global-scale operation, e.g., a global search on all documents shared by a community, whereas local operations performed in a local network with collaborators should be considerable faster.

2.1.2 P2P Scenarios

The above described questionnaire was intended to develop a better feeling for the technical requirements our case study partners have for the DI component in Nepomuk. To help everybody to better understand their scenarios where they envision the use of our distributed index, we asked them to provide us simple P2P scenarios. Again, also technical partners provided scenarios if their components rely in some point on the DI component. In the following, a few exemplary scenarios are presented:

Data sharing in groups

After Claudia returned from a trip to Belfast, she wants to share her experience about Belfast with her colleagues and friends. She took a lot of pictures of the city, some of them she wants to make available for her friends only, and some of them publicly available. Claudia selects some pictures for her friends and assigns her friends to the list of people allowed to see those pictures. She selects other pictures and just clicks on publicly available. Her friends can now find and download her pictures whenever they want even when she is not online. Publicly available pictures may be found by users but if Claudia is not online, users cannot download them.

Searching the Social / Collaborative Network

Dirk is new in his current project and the team. He already attended one project meeting but had problems following it. However, he wrote down some terms and notes that seem to be important to that project. He wants to get relevant documents and information related to these terms by searching for them in his new collaborative network. Therefore, he issues some queries that contain keywords such as the noted terms. His computer connects to the computers of his collaborative network (co-workers that fit best to the queries), i.e. to computers of co-workers that his computer deem to be part of his project-related collaborative network, where a local-search on their public data is performed. The results are then merged on Dirk's computer and presented in an appropriate way.

Shared information space

Dirk is getting to know a new tool that is used in CID, the EU project he is now working on with partners in several European countries. The project has an information area where the partners share information and publications relevant to the project. This area includes information capturing the field of P2P systems and relevant information for CID. While Dirk is examining the information he realizes that a paper he read a few days ago is very relevant to the project and should be accessible to the other partners in the project, a publication about a new topology. He adds relevant CID keywords to certain places in the paper; "Graph Topology", "Self Organizing Network". Partners in the CID project who have chosen to get information about graph topology and self organizing network get a short notice that new information is available together with a short description of the item. Hans, who is based in Lausanne, finds the paper interesting and looks through it but thinks that the paper is also relevant in another aspect for CID and adds "Trust Computation". He also adds the paper to his private information area with the node "Distributed Computing". Since Peter, another CID colleague based in Hanover is interested in trust computation he now gets a notification of the paper. He looks at the paper and finds it very interesting, it was good that Hans added the node; otherwise Peter would have missed the information. Dirk finds the information space very interesting after this experience and feels that it helps him not only to do his work but also to get to know and work together with his colleagues in the project.

Task delegation

Claudia refines her to-do list based on the process model obtained from Dirk. She decides that the negotiation of the consortium agreement should be done by an experienced person. She queries the P2P network for competencies and availability. The P2P index tells her that Ambrosia, being experienced in external negotiations and acquisitions, is the right person to contact. Claudia delegates the task "negotiation of consortium agreement" to Ambrosia. Ambrosia's task manager receives the message from Claudia's task manager via direct connection; Ambrosia confirms that she accepts the job.

2.2 Functional Requirements

From both, the questionnaire and the provided scenarios, we were able to extract some functional requirements we will present and discuss in the following. We will first focus on the functional requirements we consider relevant for the Distributed Search and Storage component as some provided scenarios require additional P2P functionality not covered by this component per definition.

2.2.1 Search

One of the main functionality of the Distributed Search and Storage component is the ability to efficiently search for information shared by users. Independent of the type of data shared by users, the DI component should support at least keyword-based search on shared information. We consider thereby the full-text search on meta-data, annotations, statements, etc. whereas full-text search on binary documents such as PDF or Microsoft Office files are out of the scope of DI. The basic reason is the huge amount of information needed to support such operations, which is very expensive in terms of required resources for large scale distributed systems. Those should be provided by social networks issuing queries directly at target peers not requiring a global full-text-index of shared documents.

Additionally to keyword-based search, more complex search operations are required, e.g., structured queries on meta-data represented by attribute-value pairs. Meta-data provided in form of RDF and RDF Schema requires a more powerful query language such as SPARQL which should further be supported by the DI component. RDF and RDF Schema also enable query reformulation, i.e., reformulation of a query from one schema to another in case heterogeneous schemas are used within a system. This might also require to first identify related or similar schemas if such translations between schemas are not provided or approved. This requires a general search for semantic similarity on the schema level, file names, annotations, etc.

Finally, the DI component should support conjunctive queries and ranking queries results, e.g., returning only the top-k hits including a quality measurement.

2.2.2 Data manipulation

Data manipulation includes the insertion of data, resulting in additional index information, and the modification or deletion of data already maintained by the system. It is important to keep in mind that data in this context is the actual shared information itself plus the meta-data used to describe the shared information. A local modification of file therefore may result in an update for the data itself and in a separate update of the meta-data used for lookups. The DI component therefore has to provide insert, update and delete operations for shared data. Further, the DI component should be able to flag data as deleted instead of physically removing data from the system.

Data shared in the DI component should be replicated and kept available so that users are able to find and retrieve shared data independent of users' availability, i.e., users should be able to find and retrieve files even if the user providing a file is currently not online. This requires replication of the data itself shared in the system and the index information required to find the data. To avoid inconsistencies after data manipulations, the system should provide reasonable consistency, again, for the data itself and the index information.

2.2.3 Access Control and Data Encryption

For all functional requirements described above, security in terms of access restrictions and encryption have to be provided, e.g., the DI component should be able to define access rights for queries and data modifications for users and groups or based on roles of users. In this way, users are able to share their data only with collaborators, friends or users they trust, if they desire.

Encryption of data should be additionally provided as the DI component has to deal with private and confidential data not intended to be seen by users outside a group or community. This involves data encryption and the encryption of index information as it is spread out among participating peers. Further, any communication between peers has to be encrypted if confidential data is shared in such a network.

2.2.4 Additional P2P Requirements

The P2P scenarios provided by the case study partners showed the need for P2P functionalities per definition not covered by the Distributed Search and Storage component as they involve mainly direct network communications and direct access to remote desktops. We do not consider those functionalities part of DI as either no storage and search is involved or they simply cannot be resolved by the DI component, at least not with reasonable efficiency. For completeness, we give in the following a list of identified functionalities not covered by the DI service.

Direct communications are messages sent from a desktop to another desktop with the aim of user notification or to search a remote desktop. Example operations include the full-text search on shared documents at collaborators desktops in a company or simple instant messaging. Full-text search was also part of the functional requirements for DI but not on large documents such as PDF documents but only on meta-data and simple annotations, e.g., tags. Apart from documents, users could query the shared calendar or task list of colleagues to find a free slot for a meeting or the see what a colleague is currently working on. After a free time slot was found at each user involved in a meeting, another instant message could be sent to a group to add this new event to their calendars or to delegate a task from one user to another. As these operations occur between individual peers and no data has to be shared over a long time for other users, those operations are out of the scope for the DI component.

Notifications are another example involving direct communication between two peers without the need to store or search for data. To give an example, a user could be interested in changes of a document and wants to be notified if a new version is available. As second example, consider the case when a user wants to inform a community about a new paper she/he discovered and considers as interesting for certain members of the community. In all those cases, a message containing the notification would be sent directly from one user to the other.

2.3 Non-functional Requirements

In addition to the aforementioned identified functional requirements, a set of non-functional requirements is mandatory in order to provide an appropriate working P2P system. Here, we briefly discuss the most important of them.

2.3.1 Scalability and Expandability

The factors of scale have been investigated in depth in the context of traditional distributed systems. Three basic dimensions have been identified: (i) the numerical dimension, which consists of the number of users, objects and services encompassed, (ii) the geographical dimension, which consists of the distance over which the system is scattered and (iii) the administrative dimension, which consists of the number of organizations that exert control over parts of the system [Neu94]. Modern P2P systems are challenging systems that may scale enormously over all of the three identified dimensions. Peers may be distributed globally and each user may have the absolute control of each own machine. Typical sizes of P2P systems may reach several millions of users and it is envisaged to be extended to billions. The significant peer heterogeneity and the highly unpredictable peer failure rate observed in P2P systems make it hard to accurately evaluate their scalability with trivial metrics.

Scalability in the design of overlay networks may be measured both in time and space dimensions. However, there is always a trade-off. Time complexity involves the number of hops that are required to forward the messages from the source node to the destination. Space complexity is related to the size of the state that each node is required to maintain in order to keep the overlay in a functioning state. This information includes the routing table entries of the neighbor nodes as well as the indexing structures for the advertised items. Additionally, the size of the messages and the overhead of the maintenance procedure should be considered in order to evaluate the scalability of the overlay network.

Expandability expresses how a system or a component can be modified to increase its storage, communication or functional capacity at low cost. For example, expandability of the topology is the ease with which the topology graph can be expanded to larger sizes. As long as an overlay network provides linear expandability properties, there is no extra complexity imposed.

2.3.2 Availability, Reliability and Fault-tolerance

Availability is the ability of an item to be in a state to perform a required function at a given instant of time or at any instant of time within a given time interval, assuming that the external resources, if required, are provided. Reliability is the ability of an item to perform a required function under given conditions for a given time interval [ITU94].

Reliability (though closely related) is distinct from availability. An important difference between the two concepts is that reliability refers to failure-free operation during an interval, while availability refers to failure-free operation at a given instant of time, usually the time when a device or system is first accessed to provide a required function or service.

Fault-tolerance is considered as a particular mean to develop the attributes that constitute the dependability concept. Fault-tolerance is the ability of a system or component to continue normal operation despite the presence of hardware or software failures. In terms of overlay networks, it expresses the resilience of the connectivity when failures are encountered by the arbitrary departure of peers.

As it has already been mentioned, P2P systems have been studied extensively [SGG02], [KWX01], [SW04]. One main aspect among others is the observation of user behavior. An interesting result is the fact that the majority of the peers tend to stay connected with the system for a relatively short time. This results in a highly dynamic system with high join and leave rates, which hinders fault-tolerant routing as it is also discussed in [ADS02] and [HK03]. This kind of behavior imposes another requirement especially important for structured overlays. Effects like overlay partitioning are possible in case where a special overlay maintenance procedure is not in place. High maintenance costs to keep the structure in a stable state and provide reliable services are some of the consequences.

2.3.3 Security

Before proceeding with the concept of security in P2P systems, it is crucial to characterize the user behavior in such systems. Relevant peer characterizations are provided in the context of rational behavior in P2P systems from an economy view point [SP03] and in terms of dependable routing [Hol04], [HMKR04], [HSSS04]. Overlay network security is highly relevant to node misbehavior. Revisiting the node classification provided in [Hol04] four different node types may be identified: (i) Cooperative nodes, (ii) Inactive nodes, (iii)

Selfish nodes and (iv) Malicious nodes. Though the notion of inactive nodes is mostly appropriate for wireless ad hoc networks, it may be of interest to P2P networks both because P2P systems may be deployed in wireless ad hoc networks as overlay networks and because ad hoc nodes have a peer-to-peer relationship at the network level. Nevertheless, selfish and malicious peers are of prime interest for the dependable and secure operation of P2P systems, so their definition is given. Selfish nodes maximize their own gain.

Security is the ability of a system to manage, protect and distribute sensitive information [MAC04]. In the context of P2P overlay networks, security issues are basically raised by the presence of malicious peers. Additionally, selfish peers may behave in a way that could have similar results. Castro [CDG+02] and Sit [SM02] address the most important security aspects in P2P overlays. They are mainly focusing on the forwarding operation, where malicious peers can either drop the packages or forward them in wrong directions and on indexing responsibilities. Furthermore, [DGM02] focus on the Denial of Service (DoS) attack problem.

2.3.4 Persistence, Consistency and Integrity of the Indexing Structure

Advertised content's persistence, consistency and integrity is not a functionality that has to be offered by the overlay network itself. For example, in the case of distributed storage systems, an additional layer is suggested to be used on top of the overlay network, which is responsible to handle such issues. Interesting examples of distributed storage systems are the CFS [DKK+01] system, which is built on top of Chord and the PAST [RD01b] system, which is built on top of Pastry. The general principles of distributed file systems are studied in [TN97] and [BDET00].

However, providing consistent and valid indexing information about the advertised resources is a crucial requirement posed on P2P overlay networks. Index replication mechanisms and information updating mechanisms are needed.

2.3.5 Load-balance, Fairness and Heterogeneity

Load-balance in P2P systems is the extent to which the load is evenly spread across nodes. The load considered in this context consists of the effort required for the basic overlay operations, e.g., maintenance, routing, indexing, caching, etc. On the one hand, designing an overlay network that avoids hot spots increases the performance and the fault-tolerance of the overall system. Appropriate mechanisms are required to evenly distribute the common tasks among the peers (e.g., uniform distribution of advertised resources [BSS02]).

On the other hand, by taking into account the heterogeneous environment, not all of the nodes are capable of offering the same amount of resources. A fair solution should provide the necessary incentives and the weighted balance between the resource contribution and the consumed overlay services.

Overlay network design should take into account the heterogeneity in the physical capabilities of the peers and the user behavior. Designing schemes that require homogeneous components can either decrease the system capabilities to those achievable by the weakest components or faulty/inefficient operation should be expected from the least capable nodes. Moreover, the observed variation in user behavior (e.g., up-time patterns) [MTG03] should be taken into account in the design of the overlay to increase the efficiency of the systems.

2.3.6 Autonomy

P2P systems are composed of a set of independent nodes that are not centrally controlled or administered. Despite this fact many proposals adopted hierarchical solutions for efficiency reasons (e.g., KaZaa [LRW03] and eDonkey [eDo05]). Such systems imposes additional requirements on the supporting infrastructure. In the case of eDonkey particularly the servers run different software. This is an even stronger requirement than potentially assigning a super-peer role to certain capable peers. Flat approaches (either structured like Chord [SMLN$^+$03] and Pastry [RD01a], or unstructured like Gnutella [Gnu05a] and Freenet [CSWH00]) are definitely preferable over the hierarchical alternatives since they maintain peer autonomy.

2.3.7 Constraint Requirements and Trade-offs

This section presents several critical effects and interferences between pairs of conflicting requirements from the aforementioned requirement set in the context of P2P overlay networks. The degree at which the conflicting requirements may be fulfilled is discussed. In following list, two examples are shortly discussed to make this issue more comprehensive.

1. **Fault-tolerance versus heterogeneity.** In the context of P2P systems where peers represent unreliable components, fault-tolerance is achieved mostly by the use of redundancy and replication mechanisms. DHT-based approaches suggest a large number of neighbors that usually increases logarithmically with respect to size of the system. While it has been shown that such an approach provides high fault-tolerance [LBK02], it ignores practical limitations raised by peers of low physical capabilities. Moreover, it should be noted that the aforementioned study makes the assumption that peer availability follows a Poisson distribution, which does not reflect the empirically observed reality [DMS05].

2. **Heterogeneity versus load-balance.** On the one hand, designing large-scale, self-organized systems may be a great challenge in scenarios where a large number of low capability and unstable peers participate. Currently, only non-autonomous systems (e.g., KaZaA, eDonkey) seem to work efficiently in wide deployments (with the exception of Overnet [Ove05]). Following such hierarchical solutions the workload is unevenly distributed among normal peers and super-peers. On the other hand, following non-hierarchical solutions needs to go beyond the currently proposed schemes to achieve efficiently fault-tolerant and stable overlays. Adaptive mechanisms are required to provide the maximum efficiency while preserving the autonomy of the peers.

3 State-of-the-Art

A great variety of approaches have been investigated to meet the critical set of the identified (and possibly additional) requirements for the operation of the P2P overlay networks. Analyzing the design mechanisms that characterize the P2P overlay networks we can identify the following general categories:

- Overlay networks vary on the degree of structure from tightly structured networks such as Chord [SMK$^+$01] or Pastry [RD01a] to loosely structured ones such as Freenet [CSWH00] or Gnutella [Gnu05b]. Tightly structured (or simply structured) overlays continuously maintain their topology, aiming at a "perfect" structure, e.g., a hypercube or a butterfly topology. Structured topologies may require high maintenance cost especially in the presence of a high churn rate. Also, they deal uniformly with the shared objects and services provided by the system and they are unaware of their distribution, a fact that might cause a significant mismatch. Moreover, most Distributed Hash Table (DHT) based approaches (which is the most common mechanism to build structured overlay networks) cannot support easily range queries[1] as effectively as lookup queries (however, P-Grid has been designed to support range queries, as it will become more clear in the following sections). Alternative investigations include several mappings of local data structures on distributed network topologies, such as tries [FV02] or modifications of traditionally used topologies such as hypercubes [SSDN02], butterflies [MNR02] and multi-butterflies [Dat02].

  On the other hand, loosely structured (or simply unstructured) overlays do not aim to reach a predefined targeted topology, but rather they have a more "random" structure. However, it has been observed that certain connectivity policies (i.e., preferential attachment) may result in topologies described by power-law networks or networks with small-world characteristics. Unstructured topologies are typically inefficient in finding published, rare items and the embedded searching operations are in general considerably costly in terms of network overhead (most approaches use flooding or at best, selective dissemination mechanisms [LRS02]). The observed power-law topology (though it provides a graph with a small diameter[2]) distributes the communication effort unevenly and introduces potential hot spots at peers with a high degree since they become "hubs" in the resulting overlay network infrastructure. However, in scenarios where the query distribution is non-uniform (i.e., lognormal, Zipfian) unstructured networks may operate efficiently.

- Further, overlay networks may vary in the dependency of the peers on each other. Approaches such as Chord or Freenet treat all of the participants equally and they are referred as pure or flat P2P networks. On the other hand, hierarchical approaches such as Napster [Nap05] or eDonkey [eDo05] separate the common overlay related responsibilities and assign the majority (or all) of the tasks to a small subset of (usually) more powerful nodes only, e.g., for resource indexing. This subset of peers is usually termed as "servers", "super-peers" or "ultra-peers". The fault-tolerance of flat approaches is considerably higher than approaches with hierarchical structure since failures or attacks to any single peer do not have as significant consequences. However, such approaches do not deal well with the heterogeneity of the participating peers both in terms of physical capabilities and user behavior. The complexity of flat approaches is usually higher compared to the hierarchical counterparts.

[1] Range queries are queries searching not for a single item that matches a specific key but rather for a set of items, which are "close" to a description based on, e.g., metadata.

[2] Small diameter is a desirable feature for a network topology in order to reduce the maximum number of hops required to reach any destination in the overlay.

On the other hand, hierarchical solutions require a certain infrastructure and may be controlled easier by third parties than the non-hierarchical alternatives. The operational load is unequally balanced among the networked entities and high dependency exists among them.

It should be noted that several systems follow hybrid mechanisms in more than one dimensions. By doing so, hybrid designs aim to deal better with the limitations of the pure approaches.

3.1 Unstructured overlay networks

Unstructured networks may be designed as non-hierarchical and hierarchical. Gnutella 0.4 [Gnu05a] is an example of the former, which offers a non-hierarchical P2P approach with minimum maintenance cost. However, the employed flooding mechanism used for querying makes Gnutella unscalable [Rit01] and caused a system breakdown in the end of 2000 when the number of users increased considerably.

Therefore, hierarchical approaches became more popular as solutions to deployed systems. The concept of super-peers, i.e., peers with additional capabilities, is introduced in these systems. Super-peers form usually the backbone of the overlay network having normal peers placed around them. The architectures of eDonkey [eDo05] and KaZaA [LRW03] may be considered typical representatives of this design direction. The basic drawback of this approach is the requirement for the existence of super-peers (or servers), which subsequently imposes new requirements on the system. Super-peers have to operate legitimately since their actions have greater effect in overlay operations. Further, super-peer failures are more severe than normal peer failures. Also, malicious behavior of super-peers has far greater impact compared to the behavior of peers in non-hierarchical P2P systems. Additionally, there is lack of incentives for super-peers to serve the rest of peers. Super-peers may also become performance bottlenecks if there is not a sufficient number of them.

3.1.1 Gnutella and power-law networks

The distribution of the node degree in a network is a significant factor for several network properties. A number of real life complex networks, e.g., income of individuals, genera, Internet file sizes [RH02], the World Wide Web, metabolic systems, paper co-authorship, movie actors [AB02], cognitive sciences [SCKH04], etc. appear to have a power law distribution of the form $P(k) = k^{-y}$ where for the most typical cases $1 \leq y \leq 3$ [WC03].

Apparently, for one of the most well-known P2P networks, the Gnutella network, it has been shown that its nodes follow a power law distribution (though not built implicitly into its design) [ALPH01], [Kab01] [PSAS01], [NG01], [ALH02]. The underlying mechanism for this evolution is the preferential attachment mechanism, where the probability Π_i of connecting to peer i with degree c_i is $\Pi_i = c_i / \sum_j c_j$. Peers tend to connect to well-known peers with higher probability ending up in the so-called "rich get richer" phenomenon. Networks that have power law distributed node degree are called scale-free networks.

Discussion

However, power-law networks are not an adequate solution for Nepomuk. While such networks have desirable characteristics such as relatively small diameter and can effectively support heterogeneity, they are vulnerable to attacks and diseases dissemination [Ald03], [BB03]. Moreover, a node with high degree holds an important position in the network. A possible removal of the node can drastically change properties such as the diameter and might result in multiple smaller graphs (network fragmentation). Moreover, if the graph is considered as a communication network, nodes with high degree are involved in delivering a large amount of traffic, ending up to potential traffic "bottlenecks".

3.1.2 Freenet

Freenet [CSWH00] is a peer-to-peer file-sharing application that supports the publication, replication and retrieval of files with a special focus on protecting the anonymity of authors and readers of the files. It is not straightforward for a node to determine what it stores, since the files are encrypted when they are stored and sent over the network. Freenet uses an adaptive routing scheme for efficiently routing requests to those nodes where they are most probable to appear. Freenet maintains routing tables, i.e., neighboring sets that are dynamically updated, as searches and insertions of data occur. Thus, the Freenet graph evolves dynamically over time as implied by the routing tables. In order to further improve search efficiency, Freenet uses dynamic replication of files along search paths.

When a peer joins a Freenet network it needs to know some existing nodes in the network. By interacting with those neighbors it fills its initially empty routing table. When a search request arrives at a peer, it may be that the peer already stores the file and can immediately resolve the request. Otherwise it has to forward the request to a neighboring peer. A peer from the routing table that is assigned the closest identifier to the requested identifier in terms of lexicographic distance, is selected as a next peer to forward the request to. The routing strategy is based on depth first search with backtracking. Requested results are routed back along the same request resolution path and are replicated along this path to provide faster hits for further requests. Results are stored in the local peer's store. If the local store is full, instead of storing the resource, an entry in the routing table is created. If the routing table is full, again the least-recently-used (LRU) strategy is being used to eliminate the oldest entry.

Discussion

While Freenet is one of the most well known systems providing anonymity characteristics, it is not adequate for Nepomuk. It lacks the required scalability properties to effectively support case studies such as the Mandriva Linux club and the reliability/availabity needs of the advertised content. Anonymity is not a major requirement as it can be concluded from the requirements section.

3.1.3 Edutella

Often, learning object (LO) providers do not want to abandon control over their resources to a third party, not even among the members of a coalition. The same concern about abandoning control also often applies to individuals, who may not want to give away their content to any centralized repository. In order to deal with this issue, distributed environments have shown to be a feasible solution for interconnection, integration and access to large amounts of information. P2P networks are one example of the impact the distribution of information might have in the sharing of information. In such networks, peers can offer various services to the user ranging from search and delivery of content, to personalization and security services. In addition, they contribute to the solution of managing the information growth, and allow every learning resource provider to offer its information without loosing control over it.

The Edutella P2P network [NWQ$^+$02] was developed with these principles as main design requirements. Edutella is a schema-based P2P network for an open world scenario in which LO's are freely offered (at not charge) and everybody is able to join (no agreement with an existing member of the network is required). It has various service facilities implemented, like for example query or publishing/subscription. Schema-based means that peers interchange RDF meta-data (data about data) among each other but not the resources themselves, that is, they interchange information about e.g. title, description, language and authors of a resource. This information can be queried using the QEL query language [NS03] (based on Datalog). Metadata interchange and search services provide the basic infrastructure needed to retrieve information about resources and services.

Discussion

Edutella aims at metadata exchange, however, it cannot be used for the needs of Nepomuk since it does not meet many of the identified requirements. However, lessons can be learnt on how to exchange metadata over the opted overlay network, as it will get more clear in the rest of this report.

3.2 Structured overlay networks

Structured overlay networks tightly control their topology and place the indexing information for the advertised resources at specified locations. Thus, subsequent queries can be routed to these locations and lookup can become more efficient. The great majority of modern structured P2P systems use Distributed Hash Tables (DHTs) as a communication infrastructure[3]. DHTs are powerful abstractions, where indexing information is placed deterministically at the corresponding peers having GUID (N_{id}) closest to the data object's advertisement unique key (R_{id}). Apparently, as their name denotes, DHTs are distributed versions of the well-known hash table based data structures. DHTs use a (commonly pre-defined among the peers) hash function to select the way resources should be treated. In fact, using widely acceptable hash functions in distributed environments is a way to provide a communication mechanism without the need to exchanging messages.

DHTs provide a scalable way to store and retrieve data objects under given keys [HKRZ02]. Each key lookup is resolved in multiple steps, resulting in a multi-hop path to be taken in the overlay. Thus, the core operation that is provided by DHTs is the following: given a key, route efficiently the query to the final destination. However, since the topologies of DHTs are usually constructed with specific constraints to provide the desired functionality, they usually do not necessarily match the topology of the underlying network.

In principle, the following steps have to be taken to design a DHT-based structured overlay network[4].

1. Define the key space (also called "virtual address space" [CF04]) and assign the keys to the participating peers and the advertised resources. It is very important that both the peers and the resources share the same key space. The length of the identifier (m) must be large enough to make the probability of keys hashing to the same identifier negligible.

2. Define the "distance" function $D(R_{id}, N_{id})$, which may be used to map the resources to the responsible peers, so that the distance function will take its minimum value. The distance function may differ considerably both in structure and semantics based on the particular system, e.g., XOR-based [MM02], prefix-based [RD01a] and arc-based [SMLN$^+$03].

3. Define the procedure supplied to peers for populating their routing tables in order to provide highly efficient routing services. Apparently,

[3]Early approaches of distributed systems developed distributed structures based on Linear Hashing [LN97], [LNS96].

[4]These steps are important to be provided here since the designed and implemented system follows a structured overlay network approach.

this selection is based on the targeted topology, e.g., mesh, torus, ring, hypercube, etc. In the cases where peers have the freedom to select among a number of candidate neighbors, additional constraints or optimization policies may be applied, e.g., optimal mapping to the underlying network, observed reliability, etc.

4. Moreover, certain additional factors have to be considered, e.g., how to handle the dynamic peer participation and the resulting effects, how to redistribute the responsibilities for the address space, how to provide resilient and fault-tolerant services, etc.

In the following subsections, several important DHT-based representatives are shortly described.

3.2.1 Chord

Chord [SMLN$^+$03] is an important reference point in the evolution of designing DHT-based P2P overlay networks, which uses consistent hashing [KLL$^+$97] to assign keys to its peers. Consistent hashing is designed to let peers enter and leave the network with minimal interruption. This decentralized scheme tends to balance the load on the system, since each peer receives roughly the same number of keys, and there is little movement of keys when peers join and leave the system. In the steady state, each peer maintains routing state information for $O(logN)$ other peers, where N is the population of the network. Node identifiers (i.e., N_{id_i} for node i) are ordered on an identifier circle using a modulo operation (with operand 2^m), thus holding that $0 \leq N_{id_i} \leq 2^m - 1$, for each peer. In this scheme, key $R_{id} = k$ is assigned to the peer whose N_{id_i} is equal to or immediately follows k in the identifier space (assuming that there is no peer j in the system so that $k < N_{id_j} < N_{id_i}$).

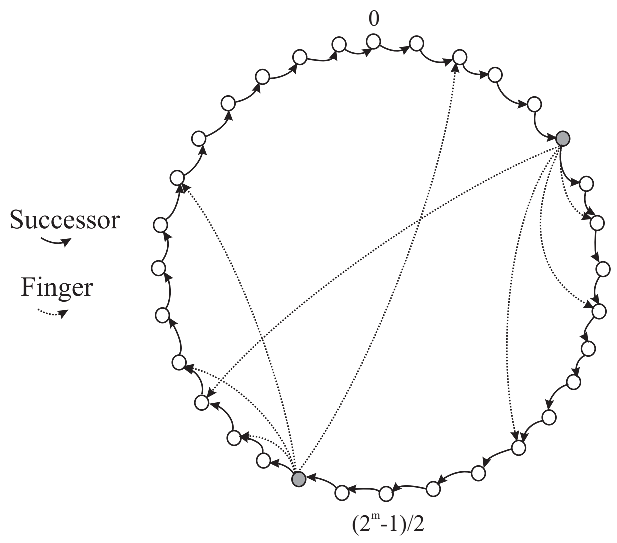


Figure 1: Chord architecture.

Chord's architecture is shown in Figure 1 forming a ring, which is enhanced with exponentially distributed shortcuts, called "fingers". Each peer is connected to its successor in the ring. Maintaining correct successors is a system requirement for its correct operation. Further, while having correct successors provides valid system operation, maintaining correct fingers (pointing to exponentially further away peers) provide efficient lookup operations. Correct fingers permit peers to halve the distance from the final destination on each routing step. Thus, on each peer join or leave action the system needs to perform $O(logN)$ topology maintenance operations, i.e., reassigning $O(logN)$ fingers and building $O(logN)$ fingers for the newly joined peer. Moreover, in an improved Chord design where fault-tolerance is targeted, additional $O(logN)$ predecessors copy their indexing information to the new peer [LNBK02].

Discussion

While Chord's architecture is an interesting approach and an evolutionary step from the previously developed architectures (e.g., Gnutella [Kab01], eDonkey [eDo05] and Freenet [CSWH00]) by addressing the lookup scalability problem very efficiently, it does not consider every requirement found in highly dynamic and heterogeneous P2P systems. Moreover, its design does not preserves the key-order, thus making inefficient the support of complex queries as it is needed in the Nepomuk project.

3.2.2 CAN

The Content Addressable Network (CAN) [RFH+01] is a distributed decentralized P2P infrastructure. The architectural design is a virtual multi-dimensional Cartesian coordinate space on a multi-torus (D-dimensional coordinate space). The entire coordinate space is dynamically partitioned among all the peers (N number of peers) in the system such that every peer possesses its individual, distinct zone within the overall space. Each peer maintains $O(D)$ neighbors and the lookup procedure requires $O(D\sqrt[D]{N})$ steps. In the following, we discuss CAN for the simplest case of a two-dimensional coordinate space where $D = 2$.

The closeness metric d corresponds to the Euclidian distance. The distance between the two rectangular zones is the Euclidean distance between their central points. Identifiers of resources are assigned to those peers whose zone contains the point that is an identifier of the resource.

Figure 2 represents an example of a CAN that consists of 10 peers. The neighborhood set $\mathcal{N}(p)$ of each peer is represented in a box and an arrow that points from the peer's zone to the corresponding box. A routing algorithm always routes a request for an identifer i towards the neighboring peer from $\mathcal{N}(p)$ whose zone is the closest to the point $F_R(i) = (x, y)$.

Peers join the CAN system by splitting a zone with an existing node into two halves and each taking the responsibility for one of the two halves. The new node obtains the neighboring links from the old one and informs the neighbors about the change.

Discussion

The scalability properties of CAN are not as attractive as other DHTs in the general case where D is statically defined a priori. It is hard to select an optimal value for D that can fit to the needs of the deployed system. Therefore, it is not a very attractive solution to consider for Nepomuk.

3.2.3 Pastry

Pastry [RD01a] makes use of Plaxton-like prefix routing, to build a decentralized self-organizing overlay network. Plaxton et al. [PRR97] proposes a distributed data structure, known as the "Plaxton mesh", optimized to support a network overlay for locating named data objects which are connected to one root peer.

The identifier space in Pastry consists of strings based on an alphabet with radix, i.e., alphabet size $b = 16$. There are two closeness metrics defined over the identifier space. The main closeness metric d in Pastry is basically the longest matching prefix between two identifiers. Besides the main closeness metric d, a specific feature of the Pastry design is that an additional closeness metric d_a is defined. The closeness metric d_a is proportional to the geograph-

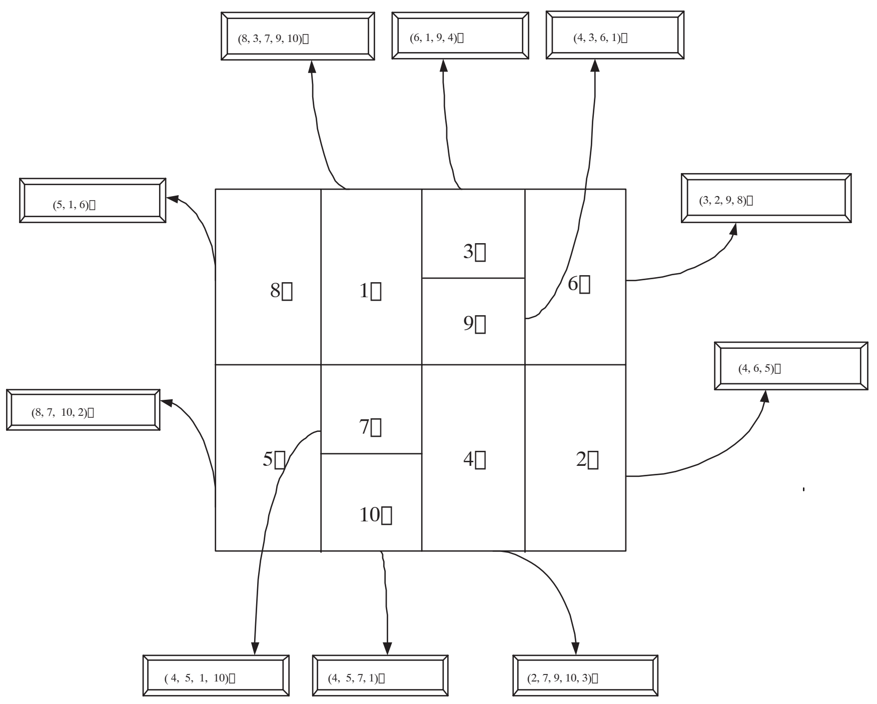


Figure 2: Example of CAN system consisting of 10 peers.

ical distance between two peers. Identifiers are ordered on a circle and are assigned to the existing peers with numerically closest identifiers. Such a peer should have an identifier that shares the longest matching prefix with the resource's identifier.

Each peer has three types of neighbors. The first type of neighbor $(routeset(p))$ is essential for performing the locate operation. Each peer maintains neighbors organized in a table with 32 rows, and $b - 1 == 15$ columns where b is a system parameter usually set to 16. The second and the third type of neighbors of peer p, $nbset(p)$ and the $leafset(p)$ are for maintaining geographic locality properties and for achieving more efficient search, respectively.

Discussion

Similarly to Chord, Pastry does not preserve key-order in its design and thus, it cannot efficiently support complex queries. Therefore, it does not meet the identified Nepomuk requirements.

3.2.4 Minerva

Minerva is a Web search engine based on a P2P system [BMT$^+$05, BMWZ05]. Every peer has its own dataset with the crawled or imported Web pages. A local index is built based on this dataset, containing inverted lists with the URLs and the terms of the dataset items. A DHT is used to create a distributed directory managing the meta-information from the local indexes of the peers. The DHT follows the Chord approach and thus each peer is responsible for the meta-information relating to a subset of the terms found in the distributed directory.

Querying using Minerva, begins from the local index of the querying peer. When the results of the query over the local index are not satisfactory, the query goes to the distributed directory. This returns a list of useful peers to which the query is forwarded. Upon the receiving of the query, each peer executes it on its local index and sends the results to the querying peer. The querying peer is responsible for combing the results and creating the final selection.

Discussion

Although Minerva is considered as successful P2P system, it still can not be used for Nepomuk. The reason is that Minerva is mainly used as a Web search engine to distribute information about the Web pages crawled by the peers.

3.2.5 Further Approaches

In this section, a short description of alternative P2P overlay networks is supplied to form a complete picture of the design space.

Tapestry [ZHS+04] has similar properties as Pastry. It employs decentralized randomness to achieve both load distribution and routing locality. In contrast to Pastry, Tapestry uses a suffix-based routing mechanism. Moreover, the handling of network locality and data object replication is performed in a different way. The architecture of Tapestry improves the Plaxton mesh structure with additional mechanisms to provide availability, scalability, and adaptation in the presence of failures and attacks (multiple roots for each data object are used by Tapestry to avoid single point of failure).

Kademlia [MM02] is a symmetrical DHT-based overlay that uses a XOR-based distance metric to construct its topology and assign the resource advertisements to peers. Kademlia's symmetrical architecture enables the usage of query messages for maintenance purposes, thus, reducing the required out--of-band maintenance signalling. Kademlia allows peers to select their neighbors from sets of peers sharing the same prefix. Kademlia, Pastry and Tapestry have operation complexity comparable to that achieved by Chord.

Omicron [DMS04] is a two-tier de Bruijn based overlay network. Omicron has low fixed node degree for low maintenance cost. Moreover, it introduces the concept of peer clusters that guarantee network stability. This hybrid topology provides a tightly structured network. In parallel, it gives the freedom of selecting neighbor peers from several members of the neighbor clusters. Additional mechanisms have been proposed for the intra-cluster organization that deals with peer heterogeneity. For this issue, a role-based approach has been investigated. More specifically, four core roles have been identified: Routers, Cachers, Indexers and Maintainers. Peers are assigned with roles based on their capabilities and their predicted behavior so that each peer can contribute in an efficient way without hindering and degrading the overall performance.

SkipNet [HJS+03] and SkipGraph [AS03] are two very similar structured overlay networks (though they have been developed independently) that extend skip lists [Pug90], a probabilistic data structure. While they are similar to Chord, their basic difference is that they release the requirement that fingers must be exponentially distributed. SkipNet and SkipGraph permit peers to have fingers that are randomly located shortcuts.

Viceroy [MNR02] is a structured network based on the butterfly topology. Viceroy requires only a constant number of neighbors with high probability while its diameter is growing up logarithmically. Though, the construction and maintenance procedures are relatively complex.

AGILE (Adaptive, Group-of-Interest-based Lookup Engine) [MS03] is a DHT-based structured overlay network that invests on the human interests to design an efficient system. As its name suggests, AGILE clusters peers based on their interests (Group of Interest - GoI). GoI are also discussed in [SMZ02] and are investigated in the context of unstructured networks in [RKP02].

Koorde [KK03] is a proposal that deploys the Chord design over de Bruijn digraphs. The authors suggest the construction of de Bruijn digraphs with node degree proportional to the logarithmic size of the network to avoid the robustness limitations of constant degree connectivity. This requires a good estimation of the size of the network and it obligates the most attractive feature of the de Bruijn digraphs (which is the combination of having logarithmic diameter and constant node degree). Koorde suggests the introduction of "imaginary nodes" to address the incremental extendability limitation of the de Bruijn graphs.

D2B [FG03] is a content addressable network that employs de Bruijn graphs to construct its overlay network. Although the proposed topology is a variation of de Bruijn graphs, they provide an interesting graph operation analysis. In D2B a procedure is suggested that allows nodes to have variable length identifiers of more than one symbol. This is even the case for linked neighbors. The resulting digraph is not always a de Bruijn one.

As it can be observed, the design of P2P overlay networks attracted a great interest from the research community. The list can be extended (though not exhaustively) to include Kelips [GBL$^+$03], Warp [JP03], AntHill [BMM02], HyperCup [SSDN02], Coral [FM03], and pSearch [TXM02]. Moreover, several interesting surveys provide comparisons among most of the well-known systems (cf. [ATS04], [LCP$^+$04], [CF04], [HAY$^+$05]).

3.3 Summary

For different reasons (e.g., complexity, different focus, not correctly validated properties, scalability, ability to support rich queries, etc.), these approaches are not optimal selections to be considered as the P2P overlay network infrastructure for Nepomuk and the identified case studies.

As it will become more clear in the following sections, P-Grid is a very attractive candidate, for several reasons. It has excellent scalability properties, it has been validated in many aspects by some first prototypes, it can efficiently support rich and complex queries, it is totally decentralized, it has no requirements in peer joining process, etc. These capabilities will be discussed in further detail in the following sections.

4 Distributed Search Component

WP4000 aims at providing a system, which leverages distributed search in a decentralized and heterogeneous network environment. Search operations can be defined in several ways with different degrees of complexity. As it has already become clear, users might look for a particular resource or person or they might be interested in a range of results. Therefore, the Distributed Search Component aims to provide all these alternatives for querying. However, the basic functionality offered in the first prototype is mostly focused on lookup operations about particular resources or persons. In fact, such operations can be performed very efficiently for certain environmental conditions, as it will become clear in this section.

The basic idea behind such a system is to create references to resources a user wants to share, called index entries, and distribute them into a distributed index. Lets imagine a simple usage scenario where a user needs to retrieve a specific resource. He issues a query[5] to the distributed index to get back its index entry, aka resource reference. This entry contains information related to its target which are: a globally unique identifier, some application specific data which are used for querying purposes, and the address of the actual resource owner. Based on this, the user is able to contact the peer hosting the desired resource in order to download it through a direct connection. More complex usage scenarios can make use of range queries to retrieve a slice of index entries or perform index manipulations through insert, update, or delete functionalities. In order to provide the needed features, the distributed search system needs to fulfill two basic sets of functionality, which are shortly described hereafter:

Index entry manipulation Index entries are at the very heart of any search system. Our component provides a mean to insert, update, and delete entries in the distributed index. It is important to note that manipulation operations are made in a best effort manner and, due to high constraint found in distributed environment, none of the ACID[6] transaction properties can be guaranteed.

Index entry retrieval Our component provides a way to query the distributed index. Since queries are resolved locally by peers without global knowledge, not all types of queries are possible. The final version of our component will provide exact queries, range queries and structured queries.

4.1 Component specification

All functionality hereinbefore cited are included in a single component called Distributed Index Component. It offers a distributed index of resource references shared by Nepomuk users. To ensure a high availability, index entries are replicated among participating computers. The resource itself remains at the owner computer and is not replicated, therefore not accessible if the owner goes offline.

In the final implementation, we will provide three query types: exact queries, range queries, and structured queries. The first one takes a keyword as search criterion and retrieves all matching index entries, the second takes a lower bound and a higher bound to retrieve all index entries in between whereas the last one take a structured search criterion and retrieves all matching index entries.

Hereafter is a short description of the available functions followed by their WSDL definition:

[5] An exact query in this case.

[6] ACID stands for Atomicity, Consistency, Isolation, and Durability.

Insert Stores an index entry into the distributed index. The entry is replicated to create a fault tolerant network.

Update Updates an index entry found in the distributed index. In order to keep the distributed index as consistent as possible, updates are propagated among all reachable peers where index entry has been initially propagated.

Delete Deletes a given index entry out of the distributed index. As for updates, deletion is done as consistently as possible in the distributed index.

Search Takes a search criterion (for exact query: a string, for range query: a lower bound and a higher bound), and uses the underlying overlay network to retrieve all index entries corresponding to this criterion.

Our component uses several types which are: Peer, IndexEntry, and ResultSet. Those types are used by P-Grid which is deeply explained in the following of this chapter. For the sake of understanding, a high level overview is given here: a Peer encapsulate all network information needed by the underneath overlay network. The IndexEntry type represents an index entry in the distributed index. As previously cited, an index entries has some application specific data used for searching, a globally unique identifier, GUID, a key, and a reference to the peer hosting the resource, OwnerPeer. Finally, a ResultSet is a set of index entries returned by a search.

```
<?xml version="1.0"?>
<definitions name="Comp-DistributedIndex"
  targetNamespace="http://nepomuk.semanticdesktop.org/2006/wsdl/Comp-
      DistributedIndex"
  xmlns:tns="urn:DistributedIndex"
  xmlns:ghns="http://nepomuk.semanticdesktop.org/2006/schemas/Comp-
      DistributedIndex"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <!-- this section defines all used types -->
  <types>
    <documentation>Types for search - result elements</documentation>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="urn:DistributedIndex">
      <xsd:complexType name="IndexEntry">
        <xsd:all>
          <xsd:element name="GUID" type="xsd:string"/>
          <xsd:element name="data" type="xsd:string"/>
          <xsd:element name="key" type="xsd:string"/>
          <xsd:element name="OwnerPeer" type="tns:Peer"/>
        </xsd:all>
      </xsd:complexType>

      <xsd:complexType name="ResultSet">
        <xsd:annotation><xsd:documentation>
          A set of matching index entries.
        </xsd:documentation></xsd:annotation>
        <xsd:sequence>
          <xsd:element name="item" type="tns:IndexEntry"
            minOccurs="0" maxOccurs="*"/>
        </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="Peer">
        <xsd:annotation><xsd:documentation>
          A peer in the P2P network.
        </xsd:documentation></xsd:annotation>
        <xsd:all>
          <xsd:element name="GUID" type="xsd:string"/>
          <xsd:element name="path" type="xsd:string"/>
        <xsd:element name="ip" type="xsd:string"/>
        <xsd:element name="port" type="xsd:int"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:schema>
</types>

<!-- this section defines which messages are used -->
<message name="insert">
  <wsdl:part name="indexEntry" type="tns:IndexEntry"/>
</message>
<message name="update">
  <part name="indexEntry" type="tns:IndexEntry"/>
</message>
<message name="delete">
  <part name="indexEntry" type="tns:IndexEntry"/>
</message>
<message name="search">
  <part name="keyword" type="xsd:string"/>
</message>
<message name="searchResponse">
  <part name="rs" type="tns:resultset"/>
</message>

<!-- this section defines the actual WSDL interface operations -->
<interface  name = "I4-Distributed Index" >
  <operation name="opInsert"
    pattern="http://www.w3.org/2006/01/wsdl/in"
    style="http://www.w3.org/2006/01/wsdl/style/iri">
    <input messageLabel="insert" />
  </operation>
  <operation name="opUpdate"
    pattern="http://www.w3.org/2006/01/wsdl/in"
    style="http://www.w3.org/2006/01/wsdl/style/iri">
    <input messageLabel="update" />
  </operation>
  <operation name="opDelete"
    pattern="http://www.w3.org/2006/01/wsdl/in"
    style="http://www.w3.org/2006/01/wsdl/style/iri">
    <input messageLabel="delete" />
  </operation>
  <operation name="opSearch"
    pattern="http://www.w3.org/2006/01/wsdl/in-out"
    style="http://www.w3.org/2006/01/wsdl/style/iri">
    <input messageLabel="search" />
    <output messageLabel="searchResponse" />
  </operation>
</interface>
</definitions>
```

Listing 1: WSDL interface for the distributed search component

4.2 Relations with other Components

Since the DI component is at the lowest layer of Nepomuk architecture, it does not depend on other components. As the other data sources, it is accessed through the data services middleware, which serves as a bridge between data sources and their consumers.

The following components are already planing to use the DI components.

Comp-Services The service component will use our component for its presence service which should inform users upon friends or colleague availability.

Comp-DesktopSemanticHelpDesk The semantic Helpdesk will distribute its knowledge base through the DI component.

Comp-ComDetLab The detection and labeling of communities component will use the distributed index for storing community related information.

Then it will rely upon the search functionality of our distributed index for its distributed RDF triples query functionality.

Comp-MetaExcRec Metadata Exchange and Recommendations system will take care of exchanging metadata between users. The prototype will be tightly related to the DI component.

Comp-TMI The goal of TMI component will be to create a knowledge/data management platform for TMI knowledge workers based on the Nepomuk technologies. Comp-TMI will use the distributed index to share and retrieve information with filtered access based on business roles.

Comp-PersonalTaskManager The personal task manager is planning to use the DI component during the second year of Nepomuk. No details are available at the moment on how it will take advantage of the distributed index.

4.3 P-Grid

In this sub-section, we present P-Grid [Abe01], an efficient infrastructure to perform lookup and search operations in large-scale and highly dynamic peer-to-peer systems. P-Grid has been designed to account for several requirements and features of P2P systems. It provides a decentralized solution where adaptability to environmental conditions is a driving factor. On the one hand, P-Grid provides an efficient structured overlay based on the concept of a distributed trie to achieve highly efficient lookup operations. Moreover, it achieves key order preservation to support range queries. On the other hand, P-Grid can utilize the underlying unstructured substrate to permit its operation in highly dynamic environments where structured approaches require high maintenance cost of the topology and the distributed index.

However, in this document, the focus is mainly on the structured P-Grid design that can perform predictably well in relatively stable environments. Although the unstructured substrate is available, it requires further fine tuning of its parameters to avoid the problems identified in the previous sections and it will be further exploited in the second Nepomuk prototype.

The basic functionality of the structured P-Grid is building a decentralized index of shared resources remaining at resource providers and performing efficient queries. The basic concepts of P-Grid will be described in more details in the following subsection, followed by an architecture overview and the implementation description. At the end we will present a first evaluation of the system.

4.3.1 Concepts

P-Grid is a binary trie-structured overlay network on top of the physical Internet network using prefix-based routing to provide efficient lookup operations. There are several other structured overlays which topologically resemble P-Grid and use prefix-based routing variants, for example, Pastry [RD01a] and particularly Kademlia [MM02] whose XOR distance metric results in the same tree abstraction and choice of routes from all peers in complementary sub--trees as in P-Grid. The important distinguishing features of P-Grid include the emergent nature of the P-Grid network based on randomized algorithms, support for substring queries, the disentanglement of peer identifiers from the associated key space, and the adaptive, structural replication (multi-faceted load-balancing of storage and query load) [ADHS05].

There is another motivation for having a trie-structured overlay network instead of a standard distributed hash table: The real advantage of traditionally using a hash table in main memory is the constant time of lookup, insert, and delete operations. But to facilitate this, a hash table sacrifices the order-relationship of the keys. However, over a network, where only parts of the hash table are stored at each location, we need multiple overlay hops anyway. For most conventional DHTs the number of hops is logarithmic in the network size. Thus the main advantage of constant-time access no longer exists in DHTs. This made P-Grid a natural choice for us to use it as the underlying routing network to support key search, substring search and range queries in the future, since it provides normal key search for same order of message complexity as a DHT, but in addition can be naturally extended to support range queries.

Peers construct the binary trie by pair-wise random interactions dividing gradually the key space in partitions defined by binary strings, the so-called peers' paths. For search, each peer maintains references to other peers/partitions at each level of the trie. Figure 3 shows a simple example of a P-Grid tree consisting of 6 peers responsible for 4 partitions, e.g., peer F's path is '00' leading to two entries in its routing table: peer E with path '11' at the first level and peer B with path '01' at the second level. This means that at each level of the trie the peer has references to some other peers that do not pertain to the peer's subtrie at that level which enables the implementation of prefix routing. Each peer constructs its routing table such that it holds peers with exponentially increasing distance in the key space from its own position. This technique basically builds a small-world graph [Kle99], which enables search in $O(\log N)$ steps.

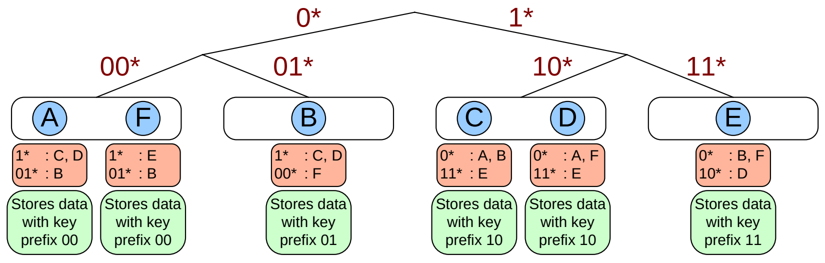

0*
1*
00*
01*
10*
11*
A
F
B
C
D
E
1* : C, D
01* : B
1* : E
01* : B
1* : C, D
00* : F
0* : A, B
11* : E
0* : A, F
11* : E
0* : B, F
10* : D
Stores data with key prefix 00
Stores data with key prefix 00
Stores data with key prefix 01
Stores data with key prefix 10
Stores data with key prefix 10
Stores data with key prefix 11

Figure 3: P-Grid overlay network

Each peer stores a set of index items which have the peers' path as prefix but it is not excluded that temporarily also other index items are stored at a peer, e.g., in Figure 3, peer F is responsible for all data with key prefix '00'. P-Grid's hash function maps application data to binary strings. In the reference implementation we assume application data to be strings for simplicity, but in fact any data type can be used. The hash function is order-preserving, i.e., it satisfies the following property for two input strings s_1 and s_2:

$$s_1 \subseteq s_2 \;\Rightarrow\; key(s_1) \subseteq key(s_2)$$

where $\subseteq$ means is-prefix-of.

To enable this mapping, we first constructed a balanced trie from a sample string database consisting of unique, lexicographically sorted strings of equal length (sample string databases can be provided by the user). The database is recursively bisected into equally-sized partitions until each partition is smaller than a threshold. The keys P-Grid uses are then calculated by using the application key to "navigate" character-wise through this trie and appending "0" to the generated key for each "left-turn" or "1" otherwise.

Moreover, for fault-tolerance, query load-balancing, and hot-spot handling, multiple peers are associated with the same key-space partition (structural replication), and peers additionally also maintain multiple references to peers with the same path (data replication). Both replication factors can be tailored towards domain requirements: higher structural and data replication guarantees better resilience against node failures and query-load imbalances at the cost of reducing the available capacity of the system and increasing the required system maintenance effort to keep replicas in sync and exchanging larger routing tables.

4.3.2 Search in P-Grid

P-Grid, like any other structured overlay approach, supports two basic operations: Retrieve(key) for searching a certain key and retrieving the associated index item and Insert(key, value) for storing new index items. Since P-Grid uses a binary tree, Retrieve(key) is of complexity $O(\log N)$, measured in messages required for resolving a search request, in a balanced tree, i.e., all paths associated with peers are of equal length. Skewed data distributions may imbalance the tree, so that it may seem that search cost may become non-logarithmic in the number of messages. However, in [Abe02a, Abe02b] it is shown that due to the randomized choice of routing references from the complimentary sub-tree, the expected search cost remains logarithmic $(0.5 \log N)$, independently of how the P-Grid is structured. The intuition why this works is that in search operations, keys are not resolved bit-wise but in larger blocks thus the search costs remain logarithmic in terms of messages. This is important as P-Grid uses order-preserving hashing to compute keys, which may lead to non-uniform key distributions.

The basic search algorithm is shown in Algorithm 1. p in the algorithm denotes the peer that currently processes the request.

Algorithm 1 Search in P-Grid: Retrieve(key, p)

```
1: if $\pi(p) \subseteq key$ then
2:     return($d \in \delta(p) | key(d) = key$);
3: else
4:     determine $l$ such that $\pi(key, l) = \overline{\pi(p, l)}$;
5:     r = randomly selected element from $\rho(p, l)$;
6:     Retrieve(key, r);
7: end if
```

The algorithm always terminates successfully, if the P-Grid is complete (ensured by the construction algorithm) and at least one peer in each partition is reachable (ensured through redundant routing table entries and replication). Due to the definition of the routing table ρ and $Retrieve(key, p)$ it will always find the location of a peer at which the search can continue (use of completeness). With each invocation of $Retrieve(key, p)$ the length of the common prefix of peer p's path $\pi(p)$ and key increases at least by one and therefore the algorithm always terminates. Note that, while the network has a tree/trie abstraction, the system is not hierarchical, and all peers reside at the leaf nodes. The peer responsible for the query, i.e. the peer's path is a prefix of the key ($\pi(p) \subseteq key$), can finally answer the query by responding with all matching index entries d in the local index table $\delta(p)$.

If we consider the P-Grid tree example in Figure 3, a search initiated at peer F for key '100' would first be forwarded to peer E because it is the only entry in F's routing table at level '1*'. As peer E is responsible for '11' and not for the key '100', peer E further forwards the query to peer D, which can finally answer the query.

4.3.3 Architecture

This section will present the architecture of the distributed search and index infrastructure taking into account the gathered user requirements and possible future extensions. The architecture design was driven mainly by the following requirements:

Scalability

The main reason for having a distributed infrastructure instead of a centralized one is the higher scalability of decentralized solutions. The peer-to-peer architecture has to be able to support millions of users in the future sharing their knowledge and data currently only accessible at the local desktop. Whereas a centralized solution is only able support a limited number of users, scalability a decentralization is a major design requirements even though it is sometimes in conflict with other requirements for the architecture.

Fault-tolerance

A decentralized system consisting of unreliable loosely coupled nodes (e.g., user desktops and laptops) has to be able to deal with failures such as network churn or node failures. The architecture has to take this into account to support those failures up to a certain degree keeping the system available without any loss of service quality.

Flexibility

The distributed search and index infrastructure has to be able to support future extensions by new technologies, such as security, social aspects, etc. A flexible architecture is therefore required which is able to integrate those extensions and offering new functionalities by already defined APIs. Apart from new technologies, the infrastructure also has to take into account domain specific requirements, i.e., users have to be able to tailor the distributed search and index infrastructure to their needs by providing application-specific handlers.

User requirements

Last but not least, user requirements have to be considered during the design of the architecture as long as they are not already covered by other requirements or can be integrated by future extensions or application-specific handlers.

The architecture of P-Grid, the distributed search and index infrastructure for Nepomuk, consists of two components: (i) the P2P basic layer and the (ii) P2P index layer as shown in Figure 4. Both of them provide interfaces for applications which can either use the lower level functions of the P2P basic layer directly or the higher level functionalities of the P2P index layer on top of the basic layer.

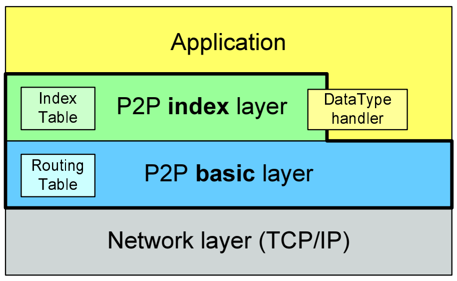


Figure 4: P-Grid architecture

P2P Basic layer

The basic layer provides core functions to exploit the P2P network such as first of all joining and leaving a network. Therefore it is only necessary to know one peer of the P2P system a user wants to join. The main functionalities this basic layer provides are lookups for peers, i.e., finding a peer responsible for a given key, and routing messages to peers either given a key or already a destination peer. The lookup operation is useful for applications that wish to send messages to peers responsible for a key in a direct point-to-point manner avoiding routing the information around in the network. This is especially important for large messages that are otherwise sent via multiple hops to their destination, or confidential messages which should only be seen by the sender and the designated receiver. Messages can further be routed in the network given a key, a set of keys, and a key range. Routing a message to a single key is the basic operation of a structured overlay as described earlier to resolve a query for a known key. If the destination is defined by a set of keys or a key range, P-Grid routes the message to all peers responsible for the given keys or all peers in the given key range. It is thereby insured that all targeted receive exactly one message. Additionally, applications can route messages to all replicas of the local peer or simply retrieve a list of replicas and references the local peer uses to route its messages (RoutingTable).

P2P Index layer

The index layer adds indexing functionalities on top of the basic layer to index and find shared content in the P2P network by reusing core functionalities from the basic layer. Applications can insert, update, and delete their content which will be hashed to the underlying key-space and routed to responsible peers which will store the new index items in their IndexTable. The application itself remains thereby responsible for which parts of the content are hashed and applications can even provide application-specific hash functions to benefit from the knowledge of the expected key distribution, improving the overall quality of the distributed search and index infrastructure. This functionality has to be implemented by the DataType handlers and be provided by the application for each type of content it wishes to index and share. The DataType handler therefore enables applications and users to tailor the P2P system to their needs and optimize the infrastructure using domain-specific knowledge.

4.3.4 Implementation

The following section presents the Java implementation of the P-Grid architecture. We will mainly focus on the two P2P layers, basic and index, and neglect details about the core P-Grid implementation. More details about P-Grid can be found on the P-Grid web site[7] and in the publications available there. The two layers provide two interfaces which allow applications to use all functionalities of P-Grid and extend or tailor it to their domain-specific needs. We will therefore first introduce the two interfaces in more detail and finally describe how the DataType handler is to be used by applications.

P2P Basic Interface

The core P2P functionalities of P-Grid are accessible via the P2P basic interface. The interface itself is defined in the `p2p.basic` package whereas the P-Grid implementation of it can be found in the `pgrid.interfaces.basic` package. The offered functions are shown in Listing 2.

```
// Local operations
public void init(Properties properties);
public Peer getLocalPeer();
public Peer[] getNeighbors();
public boolean isLocalPeerResponsible(Key key);
public void shutdown();

// Join and leave functions
public void join(Peer peer);
public void leave();

// Lookup operation
public Peer lookup(Key key, long timeout);

// Routing functions
public void route(Key key, Message message);
public void route(Key[] keys, Message[] message);
public void route(KeyRange range, Message message);
public void routeToReplicas(Message message);

// Direct Point-to-Point communication
public void send(Peer peer, Message message);

// Listener registration and removal
public void addP2PListener(P2PListener listener);
public void removeP2PListener(P2PListener listener);
```

Listing 2: The P2P basic interface

[7] http://www.p-grid.org/

The local operations allow the application to first initialize and customize the P2P facility, e.g., by providing a listening port, before it can join the P2P network by providing a bootstrap peer, i.e., a peer which is known to the application. Further, applications might be interested in the properties their local peer has in the P2P network or which neighbors, which routing table, it uses to route messages. The lookup operation takes a key and returns a responsible peer which can be used for example by the send operation to send a message directly to a peer without routing it around in the network, e.g., with respect to the previously presented scenarios, Claudia could use the send method to directly send a task to Dirk as soon as she knows the current IP address of Dirk. The four route functions route a message to a destination key, a set of keys, a key range or simply to all replicas of the local peer, respectively. Keys are in P-Grid binary strings, e.g., '010101', and peers are identified by an unique identifier, their IP address and port as well as their path, i.e., the key partition they are responsible for. Messages can be any string or binary data applications wish to send around in the P2P network. Finally, applications can register a listener to be notified about new message arrivals with the message content and the sending peer. A notification is only created if the peer is finally responsible for a message or a direct point-to-point message was received, messages routed over a peer don't result in a notification for the application.

P2P Index Interface

The P2P index interface uses the P2P basic interface to provide higher level indexing functionality to applications. It basically allows applications to index their local content in the P-Grid network so that other users can find it efficiently. The interface itself is defined in the `p2p.index` package whereas the P-Grid implementation of it can be found in the `pgrid.interfaces.index` package. The offered functions are shown in Listing 3.

```
// Local operations
public Collection getLocalIndexEntries();
public void shutdown();

// Data modification operations
public void insert(Collection entries);
public void update(Collection entries);
public void delete(Collection entries);

// Search function
public void search(Query query, SearchListener listener)
            throws NoSuchTypeException, NoRouteToKeyException;

// Listener registration and removal
public void addIndexListener(IndexListener listener, Type type);
public void removeIndexListener(IndexListener listener, Type type);
```

Listing 3: The P2P index interface

As one of the local operations, the interface can provide a list of locally shared index entries which are currently also indexed in P-Grid. To insert, update or delete any of those items, three according functions are provided. The insert operation simply inserts the provided entry into the P-Grid network whereas the delete operation deletes all index items from the P-Grid network. The update function updates all index entries with the same index entry identifier in the P-Grid. Applications can search for content using the search function by providing a query containing either a simple keyword, multiple keywords, an upper and lower bound for range queries, etc. Results will be provided to any search listener which is registered to the query identifier, i.e., additional search listeners can be registered too. Additionally to search results, applications can receive notifications about added, removed, or updated index entries if they register as index listener.

Listing 4 shows all events a search listener can receive during an issued search operation identified by a global unique identifier (GUID). As P-Grid's communication itself is asynchronous, i.e., a query messages is sent and routed in the

P-Grid network but a sending peer is not directly expecting a response, we also chose an asynchronous information flow for the implementation, realized in this case by listeners. The use of listeners has the big advantage that the search function is not blocking anymore and that more than one class implementing the search listener interface can receive search results respectively be notified about the search progress. As soon as the query message reached a responsible peer and matching items were found, the peer and therefore the listener starts to receive `newSearchResult` events. As multiple peers can be responsible for the search key space, search results may arrive sequentially leading to separate `newSearchResult` events. If no matching items could be found at a peer, the `noResultsFound` event is raised, i.e., matching items could be still found at other peers. A search is complete with the `searchFinished` notification or the `searchFailed` notification in case of network or node failures, i.e., no peers responsible could be reached.

```
public void newSearchResult(GUID guid, Collection results);
public void noResultsFound(GUID guid);
public void searchFailed(GUID guid);
public void searchFinished(GUID guid);
```

Listing 4: The SearchListener interface

DataType Handler

The data type handler enables users and applications to tailor P-Grid to their application-specific needs using prior-knowledge usable to improve the performance of P-Grid. A data type defines a type of information an application wants to share in P-Grid and is defined by an unique string, e.g., 'text/files' for simple file sharing. Applications have to create their own data types and provide a data type handler for each of them implementing the interface given in Listing 5. They therefore become responsible for core functions of P-Grid such as handling search requests, local ones as well as remote ones. The handler interface is composed of notification methods to inform the application about new or removed index items the local peer became responsible for. The idea behind letting the application be aware of those events is, that applications can extract application specific information from the index item to insert it additionally in the local database or to keep in memory. P-Grid itself is not aware of this information and therefore could not make use of it during a search. The same holds for searches, P-Grid first uses the data type handler to create query objects for a user query (`AbstractQuery[] search(pgrid.QueryInterface query)`) before it routes the queries to their responsible peer(s). There, P-Grid informs the local handler about the received query which will return the matching index items according to the query keywords. Thereby can the query contain additional information for the data type handler to further refine a query. Currently, there are search interfaces for exact keyword-based queries and range queries.

```
// modification notifications
public void indexEntryAdded(IndexEntry item);
public void indexEntryRemoved(IndexEntry item);
public void indexTableCleared();

// search handlers
public Collection handleSearch(ExactQueryInterface query);
public Collection handleSearch(RangeQueryInterface query);
public AbstractQuery[] search(pgrid.QueryInterface query);

// update handler
public boolean handleUpdate(IndexEntry item);
```

Listing 5: The DataTypeHandler interface

Usage Example

This subsection will give a short usage example of how to use P-Grid as demonstrated in Listing 6. After defining all local variables we add a new bootstrap host to the list of bootstrap hosts used by P-Grid during the bootstrap process. Those hosts are only required if a peer joins the first time a P-Grid network. After that, the basic P2P facility is acquired and initialized with the bootstrap host as property. Having a P2P basic facility, we can initialize the P2P index facility with it. No further configuration is required at this point.

To be able to share our own index entries, we have to create and register our own data type 'DemoType'. This type will be used by all index entries we will create later and by the query we will create and issue in the end. Additionally to the data type, we use the default type handler provided by P-Grid to handle our data type. The default data type handler is sufficient to share and retrieve data without making using any additional application-specific knowledge. At this point we could have provided our own data type handler to tailor P-Grid's indexing and search functionalities. Once a data type and its handler are created and registered, we are able to create index entries and inserting them in the P-Grid system. The index factory returns an index entry including an unique identifier and a binary key used to place the entry on a responsible peer.

We now join the network without the need of giving a bootstrap host as we defined one already during the initialization phase of the P2P facility. It is also possible to call the `join()` already before index entries are created but then probably additional insert messages have to be sent if a peer already joined a network and developed its own path. At the end, we create and issue a query for the keyword 'Example'. The query itself is created by the index factory given the data type we are interested in and the keyword. To receive matching items, in this example at least our two previously inserted index entries, this class also has to implement the search listener and results will be provided in the `newSearchResults(p2p.basic.GUID guid, Collection results)` function.

```java
private java.util.Properties properties = new java.util.Properties();
private P2PFactory p2pFactory;
private P2P p2pService;
private IndexFactory indexFactory;
private Index indexService;
private PGridP2P pGrid = PGridP2P.sharedInstance();

// add a bootstrap peer
properties.setProperty(Properties.BOOTSTRAP_HOSTS, "myPeer.myDomain.org:1805")
    ;

// init P2P basic facility
p2pFactory = PGridP2PFactory.sharedInstance();
p2pService = p2pFactory.createP2P(properties);

// init P2P index facility
indexFactory = PGridIndexFactory.sharedInstance();
indexService = indexFactory.createIndex(p2pService);

// creating and registering data type
p2p.index.Type type = indexFactory.createType("DemoType");
TypeHandler handler = new DefaultTypeHandler(type);
indexFactory.registerTypeHandler(type, handler);

// creating some example index items
Vector items = new Vector();
IndexEntry indexItem1 = indexFactory.createIndexEntry(type, "Example 1");
IndexEntry indexItem2 = indexFactory.createIndexEntry(type, "Example 2");
items.add(indexItem1);
items.add(indexItem2);

// inserting the index entries
indexService.insert(items);

// join the P-Grid network using previously defined bootstrap hosts
pGrid.join();

Query query = indexFactory.createQuery(type, "Example");
indexService.search(query, this);
```

Listing 6: A simple usage example of P-Grid

4.4 Evaluation

This sub-section will present a first evaluation of P-Grid in a local environment. The aim of our experiments was to test the basic functionality (keyword lookups) of P-Grid in a controllable environment before it is evaluated in a more realistic environment. In the following, we will present details about the experimental setup before we present our results.

4.4.1 Experimental setup

We used one computer, a MacBook1.1 with an Intel Core Duo processor at 2GHz and 2GB memory running MacOSX 10.4.8, to start 40 instances of P-Grid, each in an individual Java Virtual Machine (JVM). Each instance was therefore independent of each other and could have also been deployed and ran on individual machines. At the beginning, 10 keys were assigned to each peer, i.e., a key represents a statement, meta-data, a filename, etc., resulting in overall 400 keys in the P-Grid system. This relatively low number of keys was chosen to speed up experiments. To validate our experiments, we also performed tests with larger numbers and used different key distributions, including uniform random distribution and Pareto distribution.

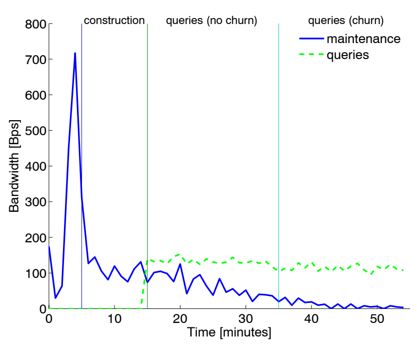
The time-line of the experiments was as follows: In an initial phase starting at time t peers join the system by contacting a bootstrap peer (until $t+20sec$) and the peers form an unstructured overlay network (from t until $t+45sec$) which is used afterward to mutually replicate their data a fixed number of times to increase availability (from $t+45sec$ until $t+3min$). In this replication phase peers randomly choose 3 peers from the unstructured overlay network to replicate their data. Subsequently, from $t+3min$ to $t+15min$, the structured overlay network is constructed. We were especially interested in evaluating the bandwidth consumption during this phase and to compare it to the consumption required to resolve queries later. Then we run queries on the constructed overlay network ($t+15min$ to $t+35min$) to analyze search performance. Each peer performed a search every 10–30 seconds. In the final phase ($t+35min$ to $t+55min$) network churn is simulated to evaluate the failure resilience of P-Grid. Each peer independently decides to go offline 20–60 seconds every 1–3 minutes which causes considerable churn the system has to compensate. During this phase, on average only 25–30 peers were online alternately.

4.4.2 Experimental results

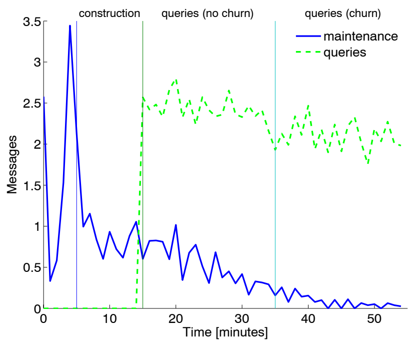
We now report some system measurements that we made to evaluate the performance of the overlay network, both during the construction phase, as well as its operational life both in a static situation (no change in peer population) as well as under churn (peers leave and join the network).

Figure 5 shows the aggregate bandwidth and message consumption of all peers (maintenance and queries) in Bytes/sec. During the construction phase the bandwidth consumption reaches a peak of 700 Bytes/sec per peer. The maintenance consumption decreases quickly down to less than 100 Bytes/sec and becomes negligible compared to the bandwidth consumed by queries. The same pattern is observable for the messages, with a peak of almost 3.5 messages during the construction phase and decreasing quickly down to less than one message. The plots in Figure 5 show that the query consumption is dominant compared to the maintenance cost for P-Grid once the P-Grid network is constructed, even during network churn.

Figure 6(a) shows the average query latency and its standard deviation. The



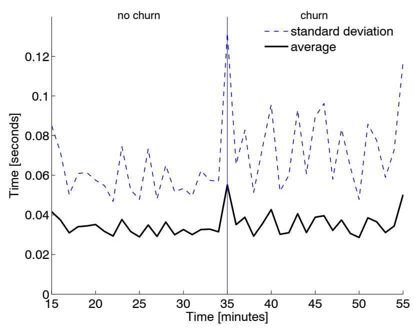
(a) Bandwidth



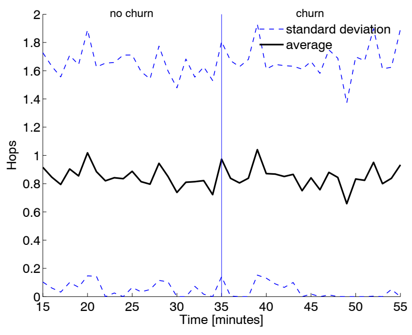
(b) Messages

Figure 5: Aggregate bandwidth and message consumption

absolute values are relatively low and essentially reflect the local setup. The response time is slightly higher with a larger deviation during the network churn because requested peers may be offline which has to be compensated. Figure 6(a) shows that the implementation is able to respond quickly to queries and latencies may only derive from network latencies on the Internet. Figure 6(b) shows the the average number of hops (and its standard deviation) required to resolve a query, i.e., how many peers were involved in routing a query before the responsible peer was reached. This is important as higher hop counts mean higher query latencies due to network latencies. We observed that the number of query hops per query is as low as theoretically expected, i.e, approx. half of the mean path length, even during churn. The average path length was slightly above 3 and the average number of query hops per query was approximately 1. Moreover after the construction phase has led to full evolution of the overlay network, all peers discovered all their replicas, and the system had an expected mean replication factor of 4, as intended.



(a) Latency



(b) Hops

Figure 6: Query latency and query hops

Finally we present in Figure 7 the observed success rate for issued queries, both for a static peer population and during churn. We can see that all queries were resolved successfully when all peers remained online and only four queries failed during the churn phase, i.e., one at minute 47 respectively one at minute 53 and two at minute 49. This is explainable by the relatively low replication of some partitions due to the small network size and the deviation of the replication factor. Queries only failed in the rare cases when all peers of those partitions were offline at the same time. We did not want to further increase the replication factor as this would have reduced the number of partitions and resulted in a too small P-Grid network. Evaluations in the future with larger networks will show P-Grid's failure-resilience as higher average replication factors can be chosen to address this problem.

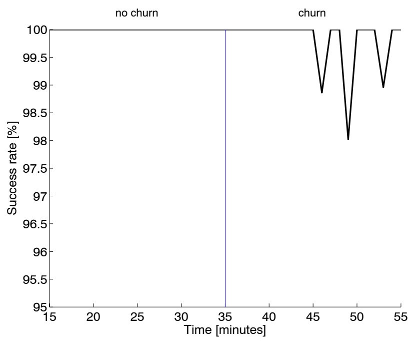


Figure 7: Query success rate

4.4.3 Discussion and Future Work

The evaluation of the first prototype of the DI component, i.e., the structured overlay of P-Grid, shows that it is able to efficiently build an index which is later used to resolve queries reliably even if not all peers of a network are online all the time. The first Nepomuk prototype was focusing on resolving exact queries only neglecting some other requirements raised by users and mentioned earlier.

Those requirements will be considered in future prototypes of P-Grid, e.g., support for more complex queries, security and storage capabilities. To reach those requirements, P-Grid's structured overlay will be extended by further functionalities to support more complex queries such as range queries and similarity queries. For more complex queries, e.g., full-text search on PDF documents, the unstructured overlay network of P-Grid will be exploited enabling users to issue any kind of queries to selected desktops, e.g., of friends or colleagues. The collected user requirements showed a need for more functionalities not covered by distributed search and storage. Some of them will be enabled by P-Grid's functionality to directly communicate with selected peers, e.g., to enable applications to delegate tasks or to inform friends about updated files.

Apart from those functional improvements for the distributed search, P-Grid will be also extended to better meet the non-functional requirements from users. Security will be addressed by encrypting messages and forming private sub-networks only accessible by group members. Scalability and availability will be further improved by implementing a two-layer super peer architecture enabling peers with limited capabilities, e.g., users using a slow Internet connection, to participate in several P-Grid networks.

4.5 Social Extensions

Previous sections explained the concepts behind P2P systems and the selection of P-Grid as the P2P system for the Nepomuk project. As explained, P-Grid will provide a distributed search and storage system for Nepomuk by mapping each personal workspace into a peer and thus enabling the interconnection of the different workspaces. It will provide services for storing and querying of the stored information and accessing the corresponding resources. Using these services we will design a number of innovative algorithms/tasks aiming at providing functionalities to the workspaces.

In this section we explain how Nepomuk will use P-Grid to provide a distributed storage using the information available on the workspaces and list possible problems that could arise in this scenario. More specifically, in Section 4.5.1 we present the distributed storage in respect to the personal workspace defined in WP2000 (creator of the information that should be distributed) and the social algorithms/tasks defined in WP4000 (the main consumer of the information stored in the distributed storage). We then present two possible problems of this scenario and provide initial proposals for addressing them. The first problem is about security in this social network (Section 4.5.2) and the second problem is heterogeneous metadata (Section 4.5.3).

4.5.1 Distributed Storage

Each personal workspace in Nepomuk has a set of desktop resources and a local store with metadata describing these desktop resources. A detailed description of the personal workspace and local store are out of the scope of this document (for more information please refer to WP2000). Our focus is on aspects of the local store influencing the P2P system.

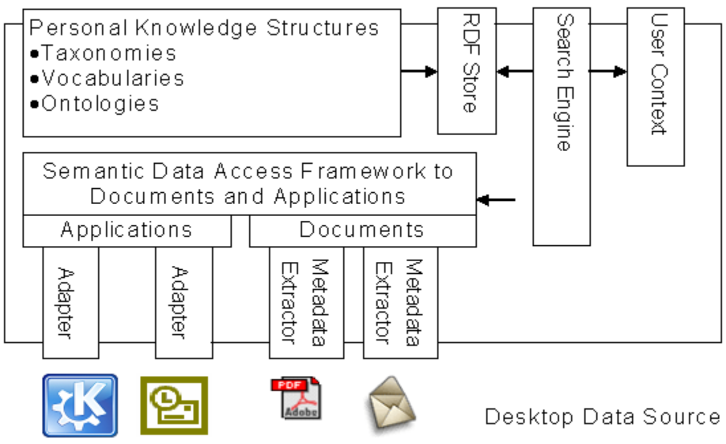


Figure 8: An illustration of the personal workspace, as provided by WP2000.

Figure 8 provides an illustration of the personal workspace, as provided by WP2000. Important aspects of the local store are:

Metadata. The local store (labeled as RDF Store in the above figure) maintains metadata describing the desktop resources found on the personal workspaces. A set of adapters and metadata extractors are used for retrieving these metadata. Example resources for adapters are IMAP emails and PDF documents, and for extractors are Microsoft Outlook and Monzilla Suite. Adapters and metadata extractors are automatically executed upon the creation or modification of desktop resources and the generated metadata are added to the local store.

Ontology. Metadata stored in the local store must conform to the Nepomuk ontology. According to the current proposal the ontology will contain sub-ontologies that should be followed by all workspaces and sub-ontologies that the workspaces will define as needed.

Resource Identification. Each resource will by identified using a globally unique URI, also stored as metadata in the local store. According to the current proposal the resource identifier will conform to HTTP and will be resolvable by workspaces.

The inter-connection of workspaces will create a social network which we will use for executing social algorithms. Some examples of these algorithms are "community identification" and "recommendations" (defined and explained in WP5000). These algorithms focus on exploiting and analyzing the information exchanged between the workspaces of the social networks. The main requirement posed on the distributed storage is to be able to access the stored information and relate it to the workspaces of Nepomuk.

The Nepomuk distributed store is responsible for the inter-connection of the workspaces. Users will be able to select the resources of its workspace that they would like to share with other workspaces. The metadata describing these resources will be stored not only in the local store but also in the distributed store. The resource identifier will be also given in the metadata and it used for relating the resources to the workspaces. The workspaces (or algorithms executed by the workspaces, i.e. recommendations) will query the metadata of the distributed store and through the use of the resource identifier, identify which workspace has resources related to the specific metadata.

4.5.2 Security

Creating a social network for sharing semantic desktop context arises security issues in Nepomuk. These security issues will be addressed both by WP4000 and WP6000. The following presents an initial list of issues that must be considered in order to create a successful mechanism for security.

Restrictive by default. When talking about conditionally sharing information, one of the main requirements is typically to assume that everything is by default private, meaning that nothing is shared unless explicitly stated. It is normally the case that the danger and consequences of sharing sensitive data is greater than not sharing public information.

Metadata vs. Resource. We have identified two levels in which information must be protected. The first are the metadata shared between workspaces to inform about the available desktop resources. The second are the actual desktop resources, corresponding to the "access paths" given by the metadata. Even if a workspace receives metadata about a specific resource, therefore knowing about the resource existence, this does not imply that the resource itself is also disclosable for the workspace.

Resource Access. Access to resources may be based on an identity (e.g. "my boss") or group membership (e.g. member of "my friends" group). However, in many cases we may not know the identity of the group or some of its members. Thus, it should also be allowed to specify access based on more general properties of the requester (e.g. "student younger than 25" or "an employee of my company") which are more general and not necessarily known in advance. A good source for these properties is the ontology of the specific workspace.

User Awareness. It should not be forgotten that we are not talking about server security and therefore it is not possible to assume qualified administrators taking care of defining all these conditions. Normal users are the ones who will personalize and configure their protection and decide what is shared to whom. Therefore, it should be ease to do it and without assuming expertise.

A possible mechanism for the Nepomuk security is policy-based access control. A policy is a statement specifying the behavior of a system. Different kind of policies allow to regulate information disclosure (privacy policies), to control access to resources (security policies) and to estimate trust based on user's properties (trust management policies). Policies are abstracted from the implementation and dynamic, change without the need to restart the system. In addition, they are typically declarative (describe the "what" but not the "how") and thus closer to the way users think.

In the Nepomuk scenario, policies will be access rules for both the metadata and the desktop resources. Accessing rules can be defined in respect to the users of social network (individual users or group of users) or in respect to the user's ontology. A 'trust negotiation' mechanism [NOW04, GNO+04] will then use these policies to establish the trust between two peers.

4.5.3 Heterogeneous Metadata

The Nepomuk scenario tries to extend the capabilities of individual workspaces by enabling sharing and interconnections between the different workspaces. This is done by merging the metadata of different local stores into a single dataset, the distributed store. This distributed store provides a uniform way to access the metadata, but since the context of the metadata is not also unified it remains heterogeneous.

Having heterogeneous metadata in the distributed store limits the functionalities by Nepomuk. For example consider the social algorithms executed using the metadata of the distributed storage, such as community identification and generation of recommendations. These algorithms perform an analysis on the context of the distributed store mainly for generating information to describe these metadata. Obviously, the success of these algorithms depends on the actual context of the distributed store and if the metadata are not heterogeneous the results produced will be more complete and valuable.

In the following paragraphs, we present two approaches for converting the context of heterogeneous metadata and propose initial approaches for addressing them in the Nepomuk scenario. Although managing heterogeneous metadata is not directly addressed by Nepomuk, we plan to investigate the problem in the extend that this appears in the "Metadata extraction and self--organized metadata alignment" task of WP5000.

Instance level

At the instance level the metadata are heterogeneous manly because the different workspaces use different naming to describe the same object. For example consider the metadata describing the emails of two workspaces. The user of the first workspace uses only the first name of persons in its address book whereas the user of the second workspace uses the full name of persons. The persons defined by the emails of the first user can not be automatically related to the persons defined by the emails of the second user, and thus the exchange of the metadata is not enough for the successful execution of social algorithms.

A possible method to converge the context of the metadata at the instance level is to process the exchange metadata to discover the existing entities along with the objects that refer to each entity. Initial proposals in the area of Personal Information Management used string similarity measures [BMC+03] to identify the similar objects. The recent proposals follow a different approach, they argue that it is important to facilitate the associations and move towards discover semantic relationships between similar objects [DHM05, MCN06].

For the Nepomuk scenario, the most suitable approach is the one that discovers the entities through the associations between the exiting objects. An important requirement is to design an algorithm that will be executed upon the exchange metadata, before the execution of the social algorithms. The results of the algorithm should be a set of entities and the corresponding set of objects for each entity, represented as metadata accompanying the original metadata.

Ontology level

As explained in Section 4.5.1 each peer can use a different ontology for describing its metadata. This becomes a problem when a peer tries to interpret metadata produced by another peer with a different unknown ontology. Consider for example one peer using an ontology in which a scientific paper is denoted by the term "paper". Another peer uses another ontology where the same concept is called "article". If the first peer now receives metadata with the term "article" from the other peer, it does not "know" that it identifies actually when it refer to a 'paper'. For exploiting at the maximum the exchange matadata from other peers one needs to somehow detect that "article" is similar to "paper".

One specificity of peer-to-peer systems is also that we do not know a priori the other peers, i.e., a new peer, using a new unknown ontology, can appear at any time. We don't only have to handle several ontologies, but we also need the possibility to take dynamically into account new unknown ontologies.

One possible solution to address those problems is a malleable schema approach [DH05]. The idea is to consider the ontology used by one peer as a malleable schema. That is, when new ontologies (or ontology elements) appears, we just add them to the peer's malleable ontology, so that it evolves over time. In parallel to the malleable ontology we maintain metadata to describe the relations between the elements of the malleable ontology. For example the metadata about the malleable ontology can be something like 'the class "paper" is a synonym of the class "article" with a probability 0.8' or 'the class "paper" is a hyponym of the class "article" with a probability 0.6'.

Note however that the approach described above does not make any assumption on the nature or provenance of the relationships between schema elements. A reasonable extension of the above approach would be to combine evidences coming from several different techniques, such as instances based, name based or user feedback.

5 Conclusions

The DI component has been designed to provide the basic functionality on locating users and remote resources in a distributed and scalable way. It provides a complete and rich interface to allow several kinds of queries (i.e., exact, ranged, structured) by meeting several critical requirements raised be the analysis of the Nepomuk case studies as well as commonly deployed P2P applications. The set of non-functional requirements includes scalability, fault--tolerance, availability and privacy. This is exactly what it has to be provided to enable meaningful and useful social semantic desktop applications in the context of the investigated case studies.

The functionality has been developed based on the P-Grid overlay network and it is provided to the other Nepomuk components via a WSDL interface. The current implementation is in a stable state and integration of the Nepomuk components is in progress. The current implementation supports exact queries only (i.e., looking for hash values of the advertised resources).

In the following period, the focus will be to extend the provided functionality to enable richer and more complex queries (ranged and structured). Moreover, the system will be extended to support the remote storage of the resources and provide a higher availability by implementing replication mechanisms. Privacy will be also the focus of the development team by providing several independent overlays that can be customized to the needs of the cooperation groups of Nepomuk users.

References

[AB02] R. Albert and A.-L. Barabasi. Statistical Mechanics of Complex Networks. Reviews of Modern Physics, 74(47), 2002.

[Abe01] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In 6th International Conference on Cooperative Information Systems (CoopIS), Pages 179–194, London, UK, 2001. Springer-Verlag.

[Abe02a] K. Aberer. Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees. Technical Report IC/2002/79, Ecole Polytechnique Fédérale de Lausanne (EPFL), 2002.

[Abe02b] K. Aberer. Scalable Data Access in P2P Systems Using Unbalanced Search Trees. In 4th Workshop on Distributed Data and Structures (WDAS'2002), 2002.

[ADHS05] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. In 31st International Conference on Very Large Databases (VLDB), August 2005.

[ADS02] J. Aspnes, Z. Diamadi, and G. Shah. Fault-tolerant Routing in Peer-to-Peer Systems. In Proceedings of the 21th Annual Symposium on Principles of Distributed Computing, Pages 223–232. ACM Press, 2002.

[Ald03] D. Aldous. A Stochastic Complex Network Model. Electronic Research Announcements of the Ammerican Mathematical Society, 9:152–161, 2003.

[ALH02] L. A. Adamic, R. M. Lukose, and B. A. Huberman. Handbook of Graphs and Networks: From the Genome to the Internet, Chapter Local Search in Unstructured Networks. Wiley, Berlin, 2002.

[ALPH01] L. A. Adamic, R. M. Lukose, A. R. Puniyani, and B. A. Huberman. Search in power-law networks. Physical Review E, 64(046135), 2001.

[AS03] J. Aspnes and G. Shah. Skip Graphs. In Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, USA, 12–14 2003.

[ATS04] S. Androutselis-Theotokis and D. Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys, 36(4):335–371, 2004.

[BB03] A. Barabasi and E. Bonabeau. Scale-Free Networks. Scientific American, 288(5):60–69, 2003.

[BDET00] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, Pages 34–43. ACM Press, 2000.

[BMC$^+$03] M. Bilenko, R. J. Mooney, W. W. Cohen, P. Ravikumar, and S. E. Fienberg. Adaptive Name Matching in Information Integration. IEEE Intelligent Systems, 18(5):16–23, 2003.

[BMM02] O. Babaoglu, H. Meling, and A. Montresor. Anthill: A Framework for the Development of Agent-Based Peer-to-Peer Systems. In Proceedings of the 22th International Conference on Distributed Computing Systems, July 2002.

[BMT$^+$05] M. Bender, S. Michel, P. Triantafillou, G. Weikum, and C. Zimmer. MINERVA: Collaborative P2P Search. In VLDB, Pages 1263–1266, 2005.

[BMWZ05] M. Bender, S. Michel, G. Weikum, and C. Zimmer. The MINERVA Project: Database Selection in the Context of P2P Search. In BTW, Pages 125–144, 2005.

[BSS02] A. Brinkmann, K. Salzwedel, and C. Scheideler. Compact, Adaptive Placement Schemes for Non-Uniform Requirements. In Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures, Pages 53–62. ACM Press, 2002.

[CDG$^+$02] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Security for structured peer-to-peer overlay networks. In Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), December 2002.

[CF04] C. Cramer and T. Fuhrmann. On the fundamental communication abstraction supplied by P2P overlay networks. European Transactions on Telecommunications, December 2004.

[CSWH00] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In ICSI Workshop on Design Issues in Anonymity and Unobservability, 2000.

[Dat02] M. Datar. Butterflies and Peer-to-Peer Networks. In Proceedings of ESA 2002 (LNCS), June 2002.

[DGM02] N. Daswani and H. Garcia-Molina. Query-Flood DoS Attacks in Gnutella. In Proceedings of the ACM Transactions on Information Systems (TOIS '02), 2002.

[DH05] X. Dong and A. Y. Halevy. Malleable Schemas: A Preliminary Report. In WebDB, Pages 139–144, 2005.

[DHM05] X. Dong, A. Y. Halevy, and J. Madhavan. Reference Reconciliation in Complex Information Spaces. In SIGMOD Conference, Pages 85–96, 2005.

[DKK$^+$01] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In Proceedings of the 18th ACM Symposium on Operating Systems Principles, Pages 202–215. ACM Press, 2001.

[DMS04] V. Darlagiannis, A. Mauthe, and R. Steinmetz. Overlay Design Mechanisms for Heterogeneous, Large Scale, Dynamic P2P Systems. Journal of Networks and System Management, 12(3):371–395, 2004.

[DMS05] V. Darlagiannis, A. Mauthe, and R. Steinmetz. Optimizing Overlay Network Stability using Burn-In Methods. Submitted for publication, March 2005.

[eDo05] eDonkey2000. http://www.edonkey2000.com, 2005.

[FG03] P. Fraigniaud and P. Gauron. The Content-Addressable Network D2B. Technical Report 1349, LRI, Univ. Paris-Sud, Paris, France, January 2003.

[FM03] M. J. Freedman and D. Maziéres. Sloppy Hashing and Self-Organizing Clusters. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03), Berkeley, CA, February 2003.

[FV02] M. J. Freedman and R. Vingralek. Efficient Peer-to-Peer Lookup Based on a Distributed Trie. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02), Cambridge, MA, March 2002.

[GBL+03] I. Gupta, K. Birman, P. Linga, A. Demers, and R. van Renesse. Kelips: Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03), February 2003.

[GNO+04] R. Gavriloaie, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett. No Registration Needed: How to Use Declarative Policies and Negotiation to Access Sensitive Resources on the Semantic Web. In 1st European Semantic Web Symposium (ESWS 2004), 2004.

[Gnu05a] Gnutella. http://www.gnutella.com, 2005.

[Gnu05b] Gnutella 2. http://www.gnutella2.com, 2005.

[HAY+05] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, and R. Campbell. A Survey of Peer-to-Peer Storage Techniques for Distributed File Systems. In Proceedings of the IEEE International Conference on Information Technology (ITCC), April 2005.

[HJS+03] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS '03), March 2003.

[HK03] K. Hildrum and J. Kubiatowicz. Asymptotically Efficient Approaches to Fault-Tolerance in Peer-to-Peer Networks. In Proceedings of 17th International Symposium on Distributed Computing, October 2003.

[HKRZ02] K. Hildrum, J. D. Kubiatowicz, S. Rao, and B. Y. Zhao. Distributed Object Location in a Dynamic Network. In Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures, Pages 41–52. ACM Press, 2002.

[HMKR04] M. Hollick, I. Martinovic, T. Krop, and I. Rimac. A Survey on Dependable Routing in Sensor Networks, Ad hoc Networks, and Cellular Networks. In Proceedings of the 30th EUROMICRO Conference, Pages 495–502. IEEE Computer Society Press, Los Alamitos, September 2004.

[Hol04] M. Hollick. Dependable Routing for Cellular and Ad hoc Networks. PhD Thesis, Department of Electrical Engineering and Information Technology, Technische Universität Darmstadt, Germany, December 2004.

[HSSS04] M. Hollick, J. Schmitt, C. Seipl, and R. Steinmetz. On the Effect of Node Misbehavior in Ad Hoc Networks. In Proceedings of IEEE International Conference on Communications, ICC'04, Paris, France, volume 6, Pages 3759–3763. IEEE, June 2004.

[ITU94] T. S. S. ITU. Terms and Definitions related to Quality of Service and Network Performance including Dependability. Technical Report ITU-T Recommendation E.800, August 1994.

[JP03] S. Jagannathan and G. Pandurangan. Stochastic Analysis of a Fault-Tolerant and Bandwidth-Efficient P2P Network. Technical Report TR-03-029, Purdue University, 2003.

[Kab01] M. Kabanov. In Defense of Gnutella. Online-Artikel, http://www.gnutellameter.com/gnutella-editor.html, 2001.

[KK03] F. Kaashoek and D. R. Karger. Koorde: A Simple Degree-optimal Hash Table. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03), February 2003.

[Kle99] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. Cornell Computer Science Technical Report, 1776(99), 1999.

[KLL$^+$97] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, Pages 654–663. ACM Press, 1997.

[KWX01] B. Krishnamurthy, J. Wang, and Y. Xie. Early Measurements of a Cluster-based Architecture for P2P Systems. In Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement Workshop, Pages 105–109. ACM Press, 2001.

[LBK02] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Observations on the Dynamic Evolution of Peer-to-Peer Networks. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02), 2002.

[LCP$^+$04] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. IEEE Communications Survey and Tutorial, March 2004.

[LN97] W. Litwin and M.-A. Neimat. LS*S: A High-Availability and High--Security Scalable Distributed Data Structure. In Proceedings of Research Issues in Data Engineering (RIDE), 1997.

[LNBK02] D. Liben-Nowell, H. Balakrishnan, and D. Karger. Analysis of the evolution of peer-to-peer systems. In Proceedings of the 21th Annual Symposium on Principles of Distributed Computing, Pages 233–242. ACM Press, 2002.

[LNS96] W. Litwin, M.-A. Neimat, and D. A. Schneider. LH*?a scalable, distributed data structure. ACM Transactions on Database Systems (TODS), 21(4):480–525, 1996.

[LRS02] Q. Lv, S. Ratnasamy, and S. Shenker. Can Heterogeneity Make Gnutella Scalable? In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02), March 2002.

[LRW03] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. Deconstructing the KaZaa Network. In 3rd IEEE Workshop on Internet Applications (WIAPP'03), June 2003.

[MAC04] K. S. Mikael Akerholm, Johan Fredriksson and I. Crnkovic. Quality Attribute Support in a Component Technology for Vehicular Software. In Proceedings of the Fourth Conference on Software Engineering Research and Practice in Sweden, October 2004.

[MCN06] E. Minkov, W. W. Cohen, and A. Y. Ng. Contextual search and name disambiguation in email using graphs. In SIGIR, Pages 27–34, 2006.

[MM02] P. Maymounkov and D. Maziéres. Kademlia: A Peer-to-peer Information System Based on the XOR metric. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02), 2002.

[MNR02] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy: a Scalable and Dynamic Emulation of the Butterfly. In Proceedings of the 21th Annual Symposium on Principles of Distributed Computing, Pages 183–192. ACM Press, 2002.

[MS03] J. Mischke and B. Stiller. Peer-to-Peer Overlay Network Management Through AGILE. In Kluwer Academic Publishers, IFIP/IEEE International Symposium on Integrated Network Management (IM), March 2003.

[MTG03] H. D. Meer, K. Tutschku, and P. T. Gia. Dynamic Operation of Peer-to-Peer Overlay Networks. Praxis der Informationsverarbeitung und Kommunication (PIK), 2003(2):65–73, 2003.

[Nap05] Napster. http://www.napster.com, 2005.

[Neu94] B. C. Neuman. Readings in Distributed Computing Systems, Chapter Scale in Distributed Systems, Pages 463–489. IEEE Computer Society Press, 1994.

[NG01] W. H. Nicholas Gibbins. Scalability Issues for Query Routing Service Discovery. In Proceedings of the Second Workshop on Infrastructure for Agents, MAS and Scalable MAS at the Fourth International Conference on Autonomous Agents (ICMAS2001), Pages 209–217, 2001.

[NOW04] W. Nejdl, D. Olmedilla, and M. Winslett. PeerTrust: Automated Trust Negotiation for Peers on the Semantic Web. In Secure Data Management, Pages 118–132, 2004.

[NS03] M. Nilsson and W. Siberski. RDF Query Exchange Language (QEL) - Concepts, Semantics and RDF Syntax. http://edutella.jxta.org/spec/qel.html, 2003.

[NWQ$^+$02] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch. Edutella: A P2P Networking Infrastructure Based on RDF. In Proceedings of WWW 2002, May 2002.

[Ora01] A. Oram. Harnessing the Power of Disuptive Technologies. O'Reilly, Sebastopol, CA, 2001.

[Ove05] Overnet. http://www.overnet.com, 2005.

[PRR97] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures, Pages 311–320. ACM Press, 1997.

[PSAS01] M. Portmann, P. Sookavatana, S. Ardon, and A. Seneviratne. The cost of peer discovery and searching in the Gnutella peer-to-peer file sharing protocol. In Proceedings of the International Conference on Networks, Pages 263–268, 2001.

[Pug90] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. Communications of the ACM, 33(6):668–676, 1990.

[RD01a] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Pages 329–350, 2001.

[RD01b] A. I. T. Rowstron and P. Druschel. Storage Management and Caching in PAST, A Large-scale, Persistent Peer-to-peer Storage Utility. In Symposium on Operating Systems Principles, Pages 188–201, 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable Content Addressable Network. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Pages 161–172. ACM Press, 2001.

[RH02] W. J. Reed and B. D. Hughes. From gene families and genera to incomes and internet file sizes: Why power laws are so common in nature. Physical Review E, 66(067103), December 2002.

[Rit01] J. Ritter. Why Gnutella Can't Scale - No, Really. Online-Article, http://www.darkridge.com/jpr5/doc/gnutella.html, 2001.

[RKP02] M. K. Ramanathan, V. Kalogeraki, and J. Pruyne. Finding Good Peers in the Peer-to-Peer Networks. In Proceedings of International Parallel and Distributed Computing Symposium (IPDPS), April 2002.

[SCKH04] O. Sporns, D. R. Chialvo, M. Kaiser, and C. C. Hilgetag. Organization, development and function of complex brain networks. Trends in Cognitive Sciences, 8(9):418–425, 2004.

[SGG02] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In Proceedings of Multimedia Computing and Networking 2002 (MMCN '02), 2002.

[SM02] E. Sit and R. Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS02), Cambridge, MA, March 2002.

[SMK+01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-to-Peer Lookup Service for Internet Applications. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Pages 149–160. ACM Press, 2001.

[SMLN+03] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable Peer-to-Peer Lookup Service for Internet Applications. IEEE Transactions on Networking, 11(1):17–32, February 2003.

[SMZ02] K. Sripanidkulchai, B. Maggs, and H. Zhang. Efficient Content Location and Retrieval in Peer-to-Peer Systems by Exploiting Locality in Interests. ACM SIGCOMM Computer Communication Review, 32(1):80–80, 2002.

[SP03] J. Shneidman and D. Parkes. Rationality and Self-Interest in Peer to Peer Networks. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS03), February 2003.

[SSDN02] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. HyperCuP - Hypercubes, Ontologies and P2P Networks. In Proceedings of the Agents and Peer-to-Peer Systems, July 2002.

[SW04] S. Sen and J. Wang. Analyzing Peer-to-Peer Traffic Across Large Networks. IEEE/ACM Transactions on Networking, 12(2):219–232, April 2004.

[TN97] P. Triantafillou and C. Neilson. Overlay Design Mechanisms for Heterogeneous, Large Scale, Dynamic P2P Systems. IEEE Transaction in Software Engineering, 23(1):35–55, January 1997.

[TXM02] C. Tang, Z. Xu, and M. Mahalingam. pSearch: Information Retrieval in Structured Overlays. In Proceedings of the 1st Workshop on Hot Topics in Networks (HotNets-I), October 2002.

[WC03] X. F. Wang and G. Chen. Complex networks: Small-World, Scale-Free and Beyond. IEEE Circuits and Systems Magazine, 3(1):6–20, 2003.

[ZHS$^+$04] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications, 22(1):41–53, 2004.