A Framework for Resource Management in Peer-to-Peer Networks

Thomas Friese, Bernd Freisleben
Department of Mathematics and Computer Science
University of Marburg
Hans-Meerwein-Str., D-35032 Marburg, Germany
{friese,freisleb}@informatik.uni-marburg.de

Steffen Rusitschka, Alan Southall
Siemens AG, Corporate Technology
Intelligent Autonomous Systems
Otto-Hahn-Ring 6, 81730 Munich, Germany
{Steffen.Rusitschka,Alan.Southall}@siemens.com

Abstract. The huge amount of recent activities in peer-to-peer (P2P) computing led to the development of a number of applications and protocols all dealing with similar problems and providing different non-interoperable implementations. In this paper, we present a framework for resource management and information storage in P2P networks. Our framework aims at providing an easy to use and flexible model for P2P computing encouraging more modularized application development and permitting reuse of components. To provide efficient localization of resources and representation of information, our resource management framework (RMF) forms a distributed information space enabling not only flat storage of information but forming graph structures that can be used to build new kinds of applications enabling the users to browse the contents of a P2P network.

1 Introduction

The popularity of the file sharing service *Napster* [1] has brought the term "Peer-to-Peer" (P2P) into broad public attention. Since then, the P2P paradigm is attracting an ever increasing interest in the industrial as well as the academic world.

In analogy to [2], we characterize a P2P network as a system of distributed entities (*peers*) that share their own resources (e.g. CPU time or storage space) and directly use resources provided by other peers over a communications network. A typical property of P2P networks are frequent joins and departures of peers due to node and link failures as well as a large fraction of users with only

temporary (dialup, ADSL) network connections.

The interest in the P2P paradigm is additionally fostered by the development of powerful personal computers. More and more resources like CPU time, storage space, data, and even services (e.g. print services) are available at the client computers of the WWW. The increase in installed broadband connections to the internet makes the effort reasonable to enable easy access to these resources and ease the collaboration of a group of users, sharing a common interest without the need to install and maintain a central server. Due to its distributed system design, P2P networks are often used as a means to increase overall system resillience against component failure or attack. On the other hand it is more difficult to administer the system and implement access controls without a central instance.

To share and access resources in such a dynamically changing environment, every P2P application has to employ some mechanism to announce its own resources and discover resources provided by other peers. A vast number of P2P based routing protocols and platforms have been developed over the past few years, e.g. Freenet [7], Gnutella [3], Pastry [8], Chord [11], CAN [13] and JXTA [14]. Most of these systems align their peers in an overlay network to existing network infrastructures, mostly IP based networks. Different approaches have been used to provide self-organizing behaviour of the peers and thus relieve the users of configuring the rapidly changing network manually. This work in the field of P2P networks is closely related to ad-hoc networking solutions that provide resource localization facilities with only limited scalability like Jini [5] or UPnP [6]. The methods for resource management used in these systems have been formally collected as design patterns [16, 17, 18, 19].

Considering the variety of systems, it is hard for developers of P2P applications to choose the best suited platform. An application developed using one platform might not be usable with another P2P platform. Therefore, it is often difficult to adopt newly developed and more efficient P2P routing mechanisms. On the other hand, developers of P2P protocols tend to reimplement features such as data redundancy and leasing that have been realized many times before.

To address these aspects of P2P development, we propose the *Resource Management Framework* (RMF). The design of the framework is based on a unified model of P2P computing derived from certain design principles observed in many P2P platforms and monolithic P2P applications. These observations were used to encapsulate a collection of important resource management concepts in a higher level framework, thus seperating different concerns in the development of P2P systems. By intoducing an additional level of abstraction it is easier to exchange e.g. the routing mechanism used by a P2P application built on top of the RMF. To provide a means for storing rich and flexible metadata about resources, we propose a data model based on XML standards. For the current implementation of the RMF we chose our own data model over the W3C Resource Description Framework standard [24] to reduce the overall complexity of the system.

The RMF allows storing and retrieving resource descriptions associated with

certain keywords in a distributed information space. In addition to a synchronous search mechanism to locate previously stored information, an application can register an asynchronous search query to receive an event notification upon a matching change of the information stored within the distributed information repository. A leasing strategy is used by the RMF to purge outdated information and queries from the repository. Furthermore, the RMF provides the ability to build associations between resources and thereby arrange and structure the collected data. The capability to form more regular topologies by self organization of peers has been improved to create more efficient routing mechanisms in P2P networks, enhancing their scalability and performance. Similarly, the user interaction with P2P applications built upon the RMF should lead to the self-organized structuring of information. The RMF also encourages a more modularized architecture of P2P applications, simplifying the adaptation to different underlying network infrastructures.

This paper is organized as follows. Section 2 gives an introduction to previous work in the field of P2P computing and presents the basic ideas and routing mechanisms we intend the RMF to use. Section 3 presents our design of the framework describing the functional units of the RMF as well as the basic operational models. In section 4, we discuss some aspects of our prototypical Java implementation. A sample application based on the RMF is described in section 5. Conclusions and directions for future work are presented in section 6.

2 Previous Work

In contrast to the client/server architecture, there is no single central system (the server) in a P2P network that provides all data and services to the consumers (clients). Information is directly (or over a number of transit systems) exchanged between providers and consumers. The term "peer" emphasizes the equality of all nodes in a P2P network; since they can act as server as well as client, the term *servent* has been adopted for the peers of a Gnutella network. Equality of all peers is no absolute necessity for a system to qualify as a P2P network with respect to the previously mentioned characterization. There is often a distinction between *pure* P2P networks that are fully decentralized and consist of "equal" nodes, and *hybrid* P2P networks where certain peers fulfil tasks distinguishing them from other peers in the network (or even centralized components are being used). In the following we consider examples for hybrid P2P systems, as well as pure P2P systems - with or without using some sort of query routing mechanisms. Furthermore, we present projects that aim at developing routing mechanisms for distributed hashtables.

2.1 First generation P2P applications

Napster, Gnutella and the Freenet project have inspired many developments in the field of P2P computing. They can be seen as a first generation of P2P systems and introduce several basic principles for the design of such systems.

2.1.1 Napster

The Napster system which contains a central index server to provide resource lookup is an example for a hybrid P2P network. Peers (often referred to as Napster clients) utilize the central server to register mp3 files they are willing to share with other peers and to gain network addresses of other peers providing a certain mp3 file they are looking for. The file itself is then transferred via a direct TCP connection between two peers, relieving the network connection of the central server from the burden of this transmission. Even though actual data transmissions take place without straining the server, the index server is still a central point of failure and therefore a target of network attacks, and in the case of Napster, legal attacks on the system. Due to the popularity of file sharing services, the large number of users can generate a substantial amount of resource registration and lookup requests as they enter and use the network.

2.1.2 Gnutella

Shortly after the great success of Napster, the first Gnutella servent was published by AOL-Nullsoft. Even though AOL almost immediately stopped the distribution of the original open sourced Gnutella application, the Gnutella protocol was used to implement a large number of servents[1] and is since then being used and improved by a large community of developers.

The Gnutella network is an example of a pure P2P network since it is designed as fully decentralized system. Every servent holds a number of connections to other peers in the network to receive and transmit protocol descriptors. To locate resources, the peers in a Gnutella network follow a relatively simple policy. A search query is issued by a peer to all neighbors. Upon first reception of such a query, it is forwarded over every connection the peer holds to neighbors. To limit the lifetime of a request, a time to live (TTL) counter is decremented prior to forwarding it. Peers do not forward queries if the TTL field has reached 0 or they have previously forwarded the request, therefore the peers record descriptor-IDs associated with every incoming request as well as the connection it was received on. If the request can be fulfilled by a peer, this information is transmitted using the reverse path of the query. The file itself is transmitted over a direct connection between the peers. While the Gnutella protocol eliminates the central point of failure, it uses flooding of the network as its primary search mechanism. The problem this parallel breath first search strategy imposes to the network became apparent when a lot of users switched from the legally troubled Napster system to the Gnutella network. A so called "bandwidth barrier"' was discovered where slow (analog modem) connections were saturated by frequent requests and could not transmit any results such that the network became unresponsive.

To tackle this problem, a more hierarchical topology of the network was introduced where peers with faster connections - so called *reflectors* or *supernodes* - act as hubs for peers with modem connections. As the reflectors themselves form

[1] We will further on refer to "servents" as "peers", since the terms can be considered to have synonymous meanings.

a Gnutella network, other bandwidth barriers exist for every class of network connections (DSL, Cable, T1 etc.).

2.1.3 Freenet

While Freenet comprises no search mechanisms but rather concentrates on aspects such as anonymity of publishers and consumers of information and resilience against censorship, it is often quoted as an inspiration to the systems presented in section 2.2. Every peer in Freenet provides some space for data storage (the data store). A key is assigned to every dataset published in Freenet and recorded in the data store along with the encrypted information itself and the address of the peer it was received from. To store or access data in Freenet, every query has to be assigned a key. Every peer inspects its data store upon reception of a query to determine the address of the peer associated with the stored key that is (numerically) closest to the key of the query. It then forwards the query to this peer. In case the target of the query can not be reached using this route, an error message is propagated back to the peer that chooses the route associated to the second closest matching key and so on. Instead of flooding the network, only a single message is propagated through the network to access a resource, following a depth first search strategy always using the route most likely leading to the destination.

The data transmitted through Freenet is stored along the path of a request while older information is removed from a peer's data store using a "least recently used" strategy. This mechanism ensures that information moves through the network towards regions of more frequent requests for a particular set of data. The multihop propagation of actual data is chosen for privacy reasons and results in an increased amount of network traffic and access latencies. On the other hand, the mechanisms of Freenet show strong self organizational properties that lead to optimized path lengths to access certain data as it is replicated towards the origin of requests.

2.2 A Second Generation of P2P Systems

The idea of distributed hash tables (DHT) as index structures in P2P networks has been adopted by a number of recently developed P2P routing mechanisms. These new systems are often aimed at internet deployment, scalability to a large number of peers and resilience of the network against attack or network failure. The basic idea of all these systems is the assignment of a key range to a single peer or group of peers. Different routing schemes are used to transmit insert and query messages through the network using keys as destination addresses. By these means it is possible to store and retrieve key-value-pairs distributed over a large number of peers.

Ratnasamy et al. [13] propose a "Scalable Content-Addressable Network" (CAN) that associates coordinates in a d-dimensional Cartesian coordinate space on a d-torus to the informational units stored in the system. The coordinate space is divided into zones each of which is assigned to a peer in the network.

Every peer stores the address of its neighbors and forwards any messages to its neighbor closest to the target zone.

The routing mechanisms used by Tapestry [9] and Pastry [8] bear many similarities and are both closely related to a scheme often referenced as "Paxtons algorithm"' [4]. These systems assign a unique number as a key to every peer in the network. A message is routed through the network by choosing a peer from each peer's routing table, whose key shares a longer prefix or suffix with the key used as target address. This method ensures fast routing towards the target, even though a second routing mechanism has to be used to determine the actual recipient of a message over the last few hops. The complexity of the routing mechanism together with the cost to build the routing tables and ensuring their consistency is often considered as a drawback of these systems.

Scribe [10] extends pastry with a publish-subscribe mechanism. It allows users to subscribe to certain object identifiers and is optimized for an efficient routing of the event notifications with an application layer multicast routing algorithm.

The Chord [11] system assigns keys of a circular key space to the peers of the network. Data is stored by the peer with the smallest key succeeding the entries' key. Every peer maintains routing tables with entries pointing to its predecessor as well as successor in the key space. To ensure faster routing, the routing tables also contain long distance shortcuts through the key space, so called "finger" entries.

Common to all these systems is the redundant storage of information to prevent data loss in case of peer or network failures. Redundancy is also often used as a caching technique to meet request hotspots at a peer holding some highly demanded information. This technique can, however, only lower the impact of frequent requests for some information. It can not lower the impact of data hotspots due to key collisions (if a lot of data is associated with the same key). While all these systems implement the basic requirements of a DHT, they share no common scheme of data representation. No mechanism to search for certain data items in the network is incorporated in these systems; like Freenet, they provide the mechanism to access information whose key is known. They also pose different demands on the underlying network infrastructure. While the Chord system with its finger tables uses long distance links in the network infrastructure, routing in a two dimensional Content-Addressable Network scheme maps naturally to a mobile ad-hoc network.

2.3 JXTA

Project JXTA was initiated by Sun Microsystems to define a framework for building P2P applications. Currently, a Java implementation of the JXTA protocols is maintained by Sun and subject to further development efforts. JXTA provides mechanisms to advertise and find peers, peer groups, services and content information. It also provides primitives for the communication between peers. The higher level concepts of our RMF proposal (e.g. resource representation, content-based publish-subscribe, searching and redundancy) are not

provided by the JXTA platform. However, the JXTA platform can be used to provide the communication facility for the RMF. Thus, the RMF presented in the next section could be implemented as a JXTA Service.

The recently presented EDUTELLA Project [15] aims at building a "P2P Network Infrastructure" based on RDF as a JXTA service. Its creators address the need to store and query rich metadata about scientific publications (often with several hundred single properties) allowing institutions and libraries the direct, electronic exchange of those publications.

3 RMF Architecture

Since resource localization and especially the ability to search for resources by keywords is a major concern of P2P networks, it is necessary to provide developers with a simple, flexible and well defined model of P2P computing that ensures interoperability of many different applications by providing the basic primitives for resource management in P2P networks. Rather than using a particular protocol implementation, developers should take advantage of the P2P paradigm itself. Providing an additional layer of abstraction simplifies the development of new protocols and applications, since protocol developers can concentrate on routing and network features, and the development of monolithic non-interoperable applications is prevented.

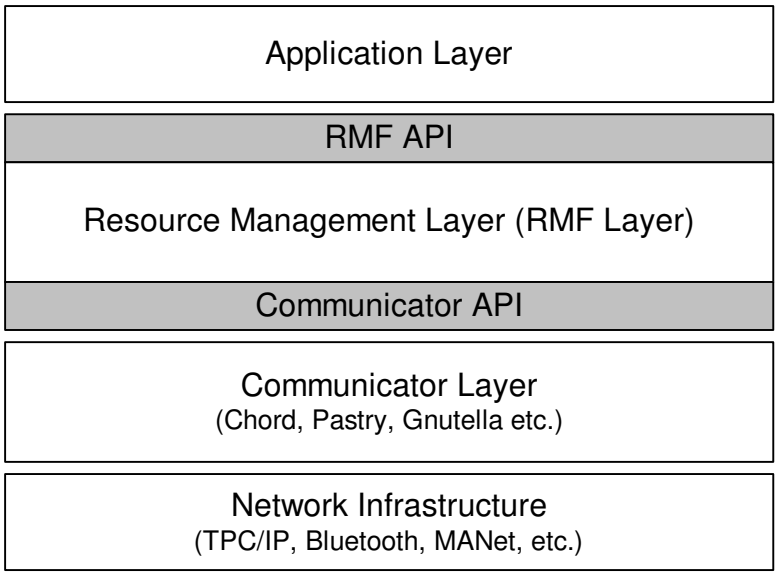


Fig 1. Architecture of the RMF consisting of four layers each representing a functional unit required in the development of non monolithic P2P applications.

Figure 1 shows the four layers important to the RMF architecture. The re-

source management or RMF layer implements resource registration, redundancy, leasing, lookup and event subscription mechanisms and is the heart of our design. The key features provided by the RMF are:

- A simple to use API which shields the user from any details of the P2P protocol(s) being used.
- A data model for defining resources.
- A way of linking resources into list or tree structures which can be efficiently navigated by the user.
- A means of registering resources in a network of RMF peers.
- A means of querying a network of RMF peers, in order to find specific registered resources, using a text based query language which is independent of the structure and content of new user defined data types.
- An event model which can be used to monitor the state of registered resources.
- A redundancy mechanism to reliably store registered resources in the case of RMF peer failures.

Two APIs are defined for the RMF as interfaces to the application layer above as well as the underlying communicator layer. Developers use the registration, lookup and event subscription methods of the RMF API to implement their P2P applications. The communicator API defines the interface that must be implemented by a specific P2P protocol implementation to be used by the RMF layer. In addition to properly implementing the communicator interface, P2P protocol implementations have to follow certain semantics of the operations to be usable with the RMF layer. These semantics have been derived by investigating several existing P2P protocols. The communicator layer delivers resource registrations, queries and event subscriptions to the peers based on keys assigned to resources by the RMF layer.

3.1 Communicator Layer

The actual P2P protocol and thus the communicator chosen to be used for an application may depend on the underlying network infrastructure. As stated in section 2.2, some protocols may be more suited to a particular physical network than others. Therefore, the communicator layer provides a level of abstraction, presenting particular communication primitives to the RMF.

The basic functionality to be implemented is the delivery of messages to a single peer or a group of peers based on a key assigned to the message. Regardless of the addressing scheme of the underlying network protocol, the communicator layer implements a routing infrastructure based on addresses used by the RMF. The communicator determines the self-organizational properties of an RMF network regarding the topology formed by the peers. In addition to the key based

routing approach, a method to send a message directly using the network address of a peer is provided, e.g. for direct delivery of query results. During its entry into a network, every peer may decide to be a passive member and only use the services provided by other peers if it is a limited device (e.g. a mobile phone) without providing its own RMF services.

3.2 Resource Management Layer

The resource management layer is the central part of our architecture as it provides the features to be used by application as well as protocol developers. Every entity to be registered in the RMF network is represented by a *resouce description*. Resources can either be items of the physical world (i.e. "things" that exist somewhere such as printers, sensors etc.) or belong to the virtual electronic world such as data or web services.

It is the responsibility of the RMF layer to handle resource registration and lookup on behalf of applications built on top of it, as well as handling the messages received from the network. That includes requests for resource registration, resource lookup and event registrations addressed to the peer. Results of such requests initiated by applications are handed over to the application upon reception.

3.2.1 Resource Descriptions - Representation of Metadata

To define the structure of resurce descriptions, the RMF uses a generic extensible model of *resource definitions* that consist of typed properties which represent the metadata information of an entity. Figure 2 shows the base resource definition which contains the properties needed by the RMF to identify resources and to structure them. This base resource definition may be extended into user resource definitions by inheritance (user resource definitions may also be inherited). A resource description or *resource* is an instance of a resource definition.

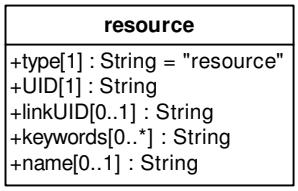


Fig 2. UML diagram of the base resource definition containing all base properties used by the RMF.

Since the `type` of a resource description determines the semantic interpretation of other properties, it is a mandatory property of every resource. It is the fully qualified name of the related resource definition describing its path of inheritance. Every resource is identified by a mandatory unique identifier (`UID`) that

is also used as the primary key for database storage of the resource description. Links between resources are defined by storing the UID of another resource in the optional `linkUID` property. We refer to the resource containing the linkUID as *child* while the resource referenced by the linkUID is called the *parent*. A number of keywords may be assigned to every resource allowing the localization of the resource by using these keywords. The optional `name` property may be used to assign a user friendly name that can be used to visualize resources of unknown type in a user interface.

Our naming scheme for resource descriptions demands the name of the newly defined type to be appended to the type of the base definition (separated with a dot). An example illustrating the process of inheriting and naming new resource definitions is shown in figure 3.

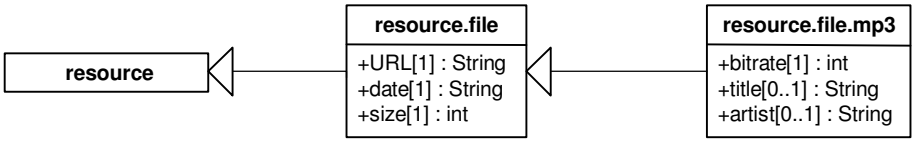


Fig 3. The base resource definition may be extended to a file (`resource.file`) definition. Then a MP3 file resource definition (`resource.file.mp3`) may be inherited from the file resource adding properties typical to MP3 files.

3.2.2 Associating Resources

Figure 4 shows the UML notation of an association resource. It contains a resource (`associated`) for reference and can be linked to another resource by its linkUID property (inherited from the base resource definition).

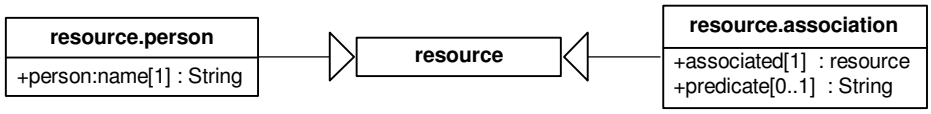


Fig 4. Definition of an association resource.

The optional `predicate` may represent further knowledge about the association. An example is shown in figure 5. The knowledge can be extracted from the network by an application aware of the semantics of the association resource.

By forming associations between resources, users of the RMF can represent knowledge that may not only be analyzed by humans but also by autonomous software agents, thus enabling the development of applications as found in the vision of the Semantic Web [12]. An agent could, for example, query a yellow pages service to find a mechanic with special skills and automatically make an

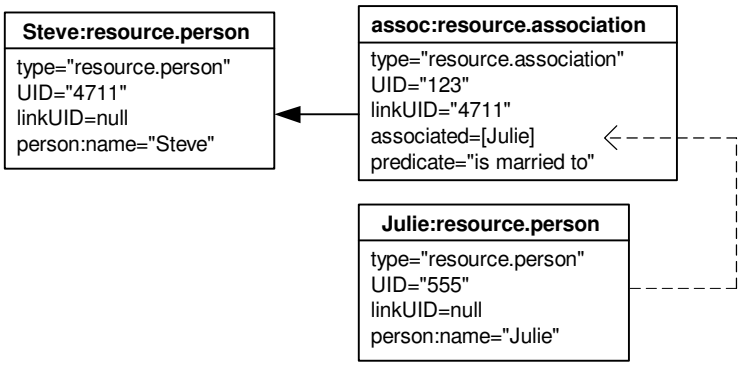


Fig 5. Using the resource definition from Figure 4 this association represents the fact: "Steve is married to Julie".

appointment if the mechanic also provides an appointment service. In addition, the use of associations might prove useful in solving the problem of key collisions, as applications should encourage users to build meaningful trees or graphs of resource descriptions with links they can follow to identify the resource they were looking for (section 3.2.6 might clarify this concept).

3.2.3 Resource Registration

In order to announce a resource to the network, a P2P application may use the `register` method provided by the RMF API. This method derives a key from the resources `UID` by applying a hash function and using the communicator to transmit a register request with this key as the destination address (this peer is known as the home peer of the resource). The same registrations are carried out deriving target addresses from the `linkUID` of the resource as well as keys derived by applying a well known hash function to all the entries in the resources keyword list. This multiple registration of one resource description with different peers results in the grouping of linked resource descriptions on one peer. This ensures that a single atomic query operation returns all linked resources (which is the typical query for a browsing application). The state of the information space after registering a few resources is shown in figure 6. In addition to the registration, applications can use the `remove` as well as `update` methods of the RMF API to change the state of the information space.

3.2.4 Redundancy

To achieve reliable storage of information within the information space even in case of peer or network failures, the RMF layer implements redundancy. This feature is implemented transparently to the application developer. Resource registrations can be assumed to be carried out redundantly on different peers of

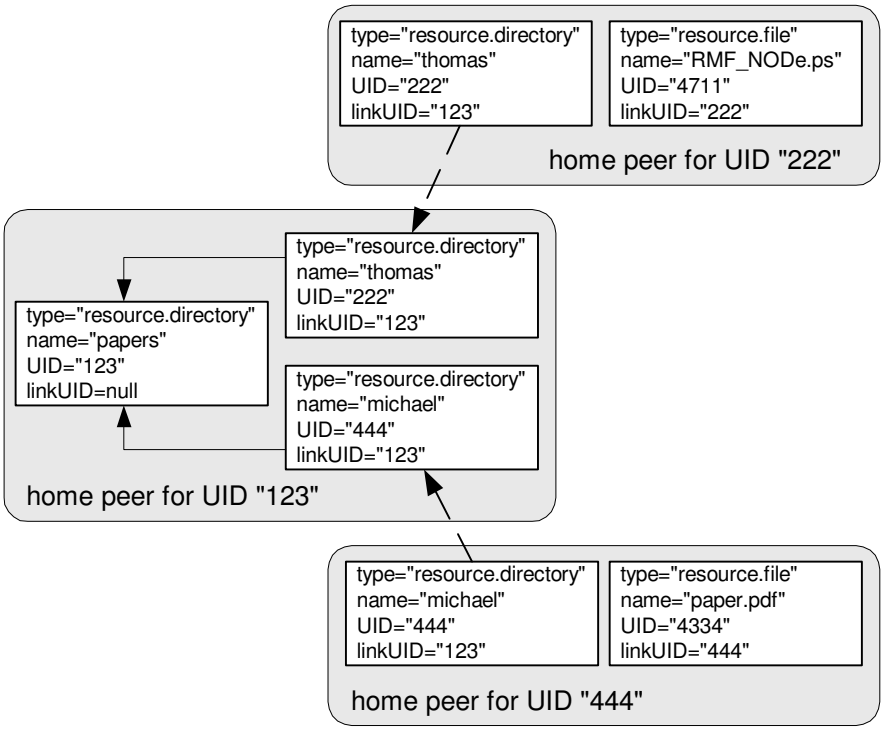


Fig 6. In addition to its home peer the resources are registered at the peer referenced by the linkUID (as well as the list of keywords). This redundancy is used to ensure lookup of all linked resources in one atomic operation.

the network.

To the communicator layer, redundancy is, however, no transparent feature but has to be supported by the implementation of the P2P protocol. In order to achieve redundancy, the communicator is required to implement a mechanism to deliver certain messages not only to a single peer but a group of peers that a key or range of keys maps to. In order to preserve a desirable level of consistency of redundantly stored information, the communicator layer is required to reliably forward a message to all members of a redundancy group.

The number of peers in such a redundancy group is designed as a configurable parameter to accommodate different environments (e.g. for internet deployment, a higher level of redundancy is desirable than in a corporate network as fluctuation of peers on the internet tends to be higher than in a LAN). Any changes to the redundancy group departure/arrival of peers has to be signaled to the RMF layer so that appropriate actions can be taken.

3.2.5 Leasing

To prevent the information space from being flooded with outdated information left behind by peer or network failure or even maliciously injected false information, a leasing strategy for resource descriptions is used. Like in Jini [5], every resource description is published to the network for a certain time of validity. If the lease granted to the registering peer is not renewed prior to the expiration of the lease, the information is deleted from the database, thus forming a mechanism of garbage collection. Lease renewal takes place by repeating the original resource registration and is automatically carried out by the RMF layer. Repeated resource registrations by applications do not break integrity of the information space, instead they are filtered by the RMFs lease renewal facility. Therefore, the leasing process is transparent to applications as well as P2P protocols and may be changed without breaking compatibility to existing RMF applications. We are currently examining different leasing mechanisms evaluating their properties under different network conditions (e.g. lease renewal of keyword registrations by the home peer might save network bandwidth collecting registrations and renewals by many providers of a resource).

3.2.6 Resource Lookup

The RMF API provides the `search` method in order to perform resource lookups in the network. A resource description can be found using two methods. The obvious and most common way is to know a keyword associated with the resource and direct a search query to the peer addressed by the hash key of this keyword.

Additionally, the links between resources may be used to find a resource. If a parent resource is known, its children can be queried to find the appropriate information. This feature could be especially useful for solving the problem of key collisions. Since resources are registered using the hash keys of all associated keywords, assigning the same keyword to many resources can lead to a large amount of information stored at that peer. Consider, for example, a file sharing application where the keyword "mp3" is assigned to every resource. To prevent this information hotspot, applications should build hierarchical information trees instead of allowing the association of such common terms to resources.

3.2.7 Events

In addition to synchronous searches by submitting a query, an application can use the RMFs event feature to be informed of changes to the information space based on a given query. An application subscribes to an event by calling the `subscribe` method of the RMF API passing a list of keywords, the actual query (as in search queries) and the type of change to be monitored (addition, removal or modification of information). A subscription message is sent to every peer addressed by the list of keywords.

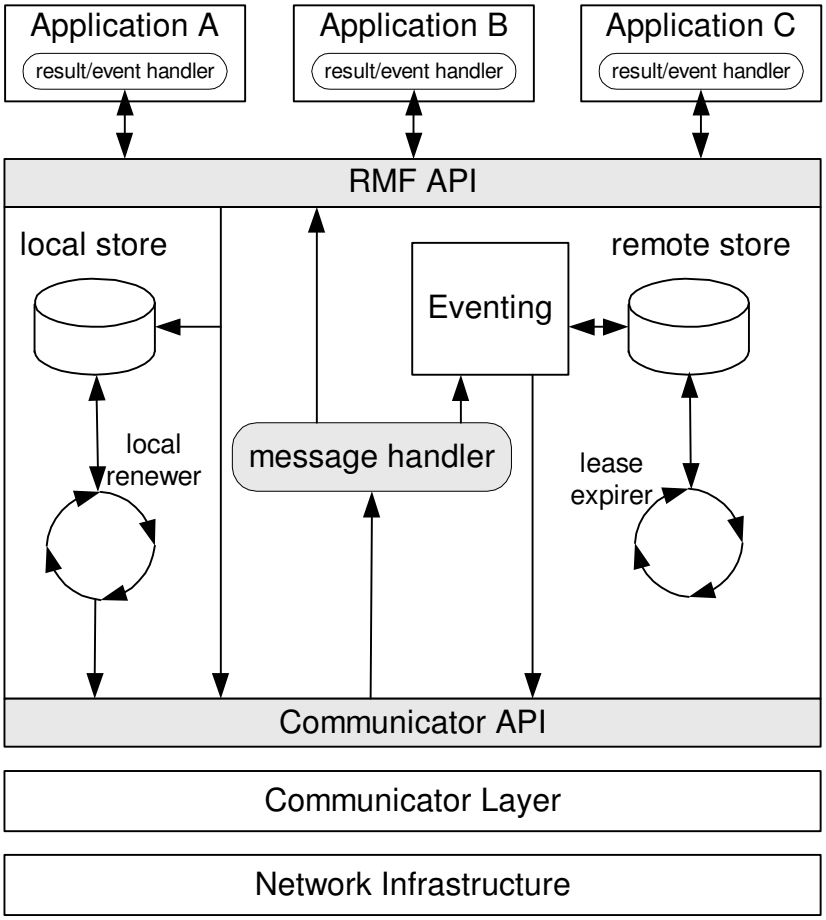


Fig 7. Functional components of the resource management layer.

3.2.8 Functional Components of the RMF Layer

The functional components of the RMF layer present on every peer are shown in figure 7. In addition to remote resource registrations, the resource description together with the expiry time for the lease is stored in the *local store*. This local store is constantly watched by the *local renewer* thread that refreshes leases on behalf of P2P applications using the RMF layer. Similarly, a database - the *remote store* - is provided to store resource information received by the communicator layer. Upon reception of a register request, the communicator layer hands the message to the central message handler of the RMF layer. The resource description is matched against all registered event subscriptions prior to storing it in the database. In case of a match, the appropriate message is

generated and sent to the subscriber using the direct message delivery feature of the communicator layer. The remote store is constantly watched by the lease expirer thread for expiration of leases that leads to removal of the information.

4 Implementation

A prototypical implementation of the RMF in Java has been developed at Siemens Corporate Technology in cooperation with the University of Marburg. The RMF has been deployed in practice to a limited number of workstations running Windows on the Siemens corporate LAN.

The RMF uses a custom CHORD [11] implementation as communicator, calling the `init(redundancy, active_member)` method to join the network. The parameters are passed to control the behavior of the network respectively peer (i.e. the number of members in a redundancy group and whether it supports registration of resources or wishes to only passively use the RMF). The event handler of the RMF layer is registered with the communicator via the `setEventHandler` method. In order to process events generated by the communicator (e.g. if a registration request is received or the peer joins a redundancy group), it must implement the methods `handleMessage` and `replicationRequest`. The address passed along with a message is the network address of the originator of the transmission. It may be used by the RMF layer to directly send a reply. A replication request contains a range of hash keys and a (number of) peers that should be queried for all information to replicate it.

The methods to `send` messages come in three flavours: they support relay of a message to a single peer based on its network address or based on a hash key as target address, as well as `sendGroup` method that delivers the message to all peers of a redundancy group. All methods take a hook to an acknowledgement handler that receives information about the state of message delivery.

As with the main message handler of the RMF, most communication between functional layers is implemented by events. A query to the RMF using the `search` method of the RMF API results in a query to the network that can take substantial time to finish and come up with results. Therefore, a result handler is registered with the RMF and results are passed to this handler upon reception.

Since resource descriptions are basically collections of metadata using XML, a number of choices for query languages existed, e.g. XPath [21], XQuery [22] and XQL [23]. In our prototypical implementation we chose XQL, since engines such as the GMD-IPSI XQL engine [20] exist. XQL queries can easily be passed as strings to the `search` and `subscribe` methods of the RMF.

5 Application

As a first prototypical application, a basic distributed storage system has been developed. It provides a decentralized directory tree as known from Windows folders or the UNIX filesystem tree. The tree is navigated and modified using a

command shell. Users can list the content of a directory, change into subdirectories and add local files. Apart from browsing the directory tree, the user can search for files based on keywords associated with the file.

To represent a directory, an instance of a resource definition of type "resource.directory" is used. Other resources like files as well as subdirectories are connected to this directory by using the linkUID property of the resources. An example of such a directory tree is shown in figure 6 where the subdirectories "thomas" and "michael" are linked to the directory "papers".

An additional property of file resources is the URL to actually access the file when desired. This can be any kind of URL defining different methods of access (most common will be the transmission by HTTP or FTP, since server implementations are available and can easily be integrated into applications to make local files available to network users). Issuing the command to copy a remote file to a local directory, the file is directly acquired from the remote peer.

While the directory structure resides in the information space formed by the RMF, all files are stored locally by the peers. Getting the list of contents of a directory requires only a single request to be transmitted over the network, as stated in section 3.2.3.

6 Conclusions

In this paper, we presented the basic concepts of a resource management framework (RMF) for P2P networks. The RMF defines an easy to use model of P2P computing that can be used by application developers as well as protocol developers. By introducing an additional level of abstraction, both groups can benefit from using the RMF, as it provides features common to most P2P implementations, seperating concerns in the development of P2P networks. The RMF offers a generic, extensible data model to describe metadata linked to resources and store these resource descriptions in an information space distributed among the nodes of a P2P network. Besides "things" in the real world, resources can be items of the virtual or electronic world. This information space not only stores flat resource descriptions, but allows to form links between resources allowing the organization of resources into graph or tree structures by the users of the system. We are currently examining the future use of RDF for metadata representation while keeping a lightweight and easy to use system design. By leveraging advanced routing mechanisms, the system can efficiently discover resources and information in the P2P network formed by the RMF peers while its layered architecture allows it to be easily adaptable to different physical network infrastructures.

Since P2P networks typically grow to a large number of peers, the amount of information stored by the system as well as number of events generated by all peers will be quite substantial. Since we believe that autonomous agents that act on the infrastructure provided by the RMF will play a vital role in the future, the data model defined for the RMF supports the development of semantic web [12] like applications suggesting the use of agent technology to

harvest the information represented in the network.

The RMF prototype is currently under evaluation at Siemens Corporate Technology and the University of Marburg. Security, reliability and scalability issues in large, inter-corporate networks are topics to be addressed during this project. We also intend to improve the mechanisms provided to circumvent information hotspots by structuring the data and leveraging the collaborative effort of all users of the system for this task.

References

[1] P2P-Filesharing-System, Napster Inc., `http://www.napster.com`

[2] Schollmeier, R., "A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications", in Proc. of the IEEE 2001 International Conference on Peer-to-Peer Computing, Linköping, August 2001, pp. 101-102.

[3] The Gnutella Project, `http://www.gnutellanews.com`

[4] Plaxton, C. G., Rajaraman, R., Richa, A. W., Accessing Nearby Copies of Replicated Objects in a Distributed Environment, in Proc. of ACM SPAA, ACM, June 1997, pp. 311-320.

[5] Jini Network Technologies, Sun Microsystems, `http://www.sun.com/jini`

[6] The Universal Plug and Play Forum, `http://www.upnp.org`

[7] Clarke, I., Sandberg, O., Wiley, B., Hong, T. W., Freenet: A Distributed Anonymous Information Storage and Retrieval System, in Designing Privacy Enhancing Technologies: International Workshop on Design Issues in Anonymity and Unobservability, LNCS 2009, ed. by H. Federrath. New York, 2001, pp. 46+.

[8] Rowstron, A., Druschel, P., "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, November 2001, pp. 329-350.

[9] Zhao, B. Y., Kubiatowicz, J. D., Joseph, A. D., Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing, University of California at Berkeley Technical Report UCB//CSD-01-1141, April 2001.

[10] Rowstron, A., Kermarrec, A.-M., Castro, M., Druschel, P., SCRIBE: The Design of a Large-Scale Event Notification Infrastructure, Proceedings of the Third International COST264 Workshop (NGC 2001), Springer, pp. 30-43.

[11] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H., Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, San Diego, August 2001, pp. 149-160.

[12] Berners-Lee, T., Hendler, J., Lassila, O., The Semantic Web, Scientific American, May 2001, pp. 28-37.

[13] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S., A Scalable Content-Addressable Network, in Proceedings ACM SIGCOMM, San Diego, August 2001, pp. 161-172.

[14] Project JXTA. http://www.jxta.org

[15] Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmr, M., Risch, T.: EDUTELLA: A P2P Networking Infrastructure Based on RDF. in Proc. of The Eleventh International World Wide Web Conference, Maui, Hawaii, 2002, pp. 604-615.

[16] Jain, P., Kircher, M., Leasing Pattern, PLoP 2000 conference, Allerton Park, Illinois, USA, 2000

[17] Kircher, M., and Jain, P., Ad Hoc Netoworking Pattern Language, EuroPLoP 2001 conference, Kloster Irsee, Germany, 2001

[18] Jain, P. and Kircher, M., Partial Acquisition Pattern, PLoP 2002 conference, Allerton Park, Illinois, USA, 2002

[19] Kircher, M., and Jain, P., Lookup Pattern, EuroPLoP 2000 conference, Irsee, Germany, July 5-9, 2000

[20] GMD-IPSI XQL Engine: http://xml.darmstadt.gmd.de/xql

[21] XML Path Language (XPath), W3C, http://www.w3.org/TR/xpath

[22] XQuery 1.0: An XML Query Language, W3C, http://www.w3.org/TR/xquery

[23] XML Query Language (XQL), W3C, http://www.w3.org/TandS/QL/QL98/pp/xql.html

[24] Resource Description Framework (RDF), W3C, http://www.w3.org/RDF/