

# The Resource Management Framework: A System for Managing Metadata in Decentralized Networks Using Peer to Peer Technology

Alan Southall and Steffen Rusitschka

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

**Abstract.** Given the current explosion in peer-to-peer based protocol implementations, application developers require a means of abstracting the individual characteristics of specific peer-to-peer implementations away from their application logic. The *Resource Management Framework* provides a unified model of peer-to-peer computing which is independent of the underlying protocols. To this end, a data model, supporting Semantic Web style applications, and APIs for both application and peer-to-peer protocol developers are provided which hide many of the intricate details, such as database management and redundancy, inherent in all peer-to-peer systems. Central to the Resource Management Framework's design is an object oriented XML database which can be queried using an XML query language. By using XML query language based event subscriptions, an application developer can monitor add, remove and modify states within a Resource Management Framework database, thus enabling content-based publish-subscribe applications.

## 1 Introduction

The peer-to-peer (P2P) paradigm is attracting an ever increasing amount of attention in both industrial and academic spheres. Over the past few years there has been an explosion in the number of P2P based routing protocols and platforms, e.g. Chord [6], Pastry [2], CAN [9], and JXTA [7]. Given such a large number of systems, one major concern for application developers is, however, which system to choose and whether applications written for one specific P2P solution will be usable with other P2P platforms. Similarly, P2P protocol developers often tend to re-implement the same features realized several times before in previous P2P protocols, e.g. database support, leasing and data redundancy.

The primary objective of the Resource Management Framework (RMF) is to provide an active and collaborative information space for heterogeneous data which can be browsed or modified in order to share information between groups of all sizes based on a unified model of peer-to-peer computing which is independent of the underlying protocols. To this end, an extensible data model based on XML has been created which defines a base class called *resource*. Resources may be inter-linked to

form trees and lists, and inherited to form new user defined types. Semantic Web [12] style applications are supported by the embedding of, for example, RDF [13] metadata in a resource.

The RMF architecture defines two APIs: one for application developers and one for P2P protocol developers. These APIs provide access to RMF functionality such as resource registration, lookup, data redundancy, etc. Additionally, it defines the requirements for an XML database and an XML querying language which must be independent of the structure and content of the database.

A powerful feature of the RMF is a content-based publish-subscribe [8, 1] mechanism which is based on XML query language event subscriptions. Event subscriptions allow an application developer to monitor add, remove and modify states within an RMF database and provide a way of performing inter-process communication between applications.

This paper presents a brief introduction to the RMF based on our prototypical implementation written in Java. A first attempt to describe the RMF is given in [11]. Section 2 gives an overview of the RMF architecture. Section 3 describes the resource data model, whilst Sections 4, 5 and 6 describe the operations which can be applied to a resource. Section 7 gives an overview of applications which are enabled by the RMF platform and Section 8 describes related work. Conclusions and a description of future work are given in Section 9.

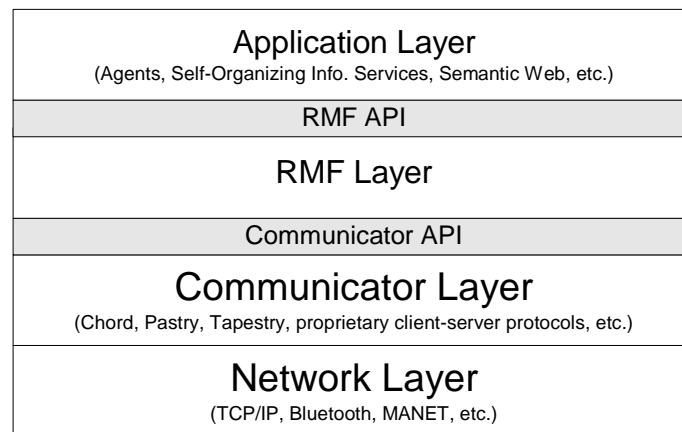
## 2 System Architecture

The RMF architecture aims to provide P2P application developers with a simple, interoperable, well defined model of P2P computing, i.e. an application should be written to take advantage of the P2P paradigm and not a specific P2P protocol implementation. Similarly, the architecture supports the P2P protocol developer by providing many of the features required by P2P systems, e.g. a database and a redundancy mechanism. This not only simplifies the development of new P2P protocols, i.e. the protocol developer can concentrate on the network and routing features of the protocol, but also prevents the development of monolithic P2P systems all supporting the same features with varying, non-interoperable, implementations.

The RMF architecture has therefore been designed so that a level of abstraction between RMF applications and the underlying P2P protocol(s) is provided which supports the following features:

- A simple to use API which shields the user from any details of the P2P protocol(s) being used.
- An extensible data model for defining resources, i.e. a base class for resource definitions and a set of semantics for the base class which can be extended by the user in order to create user defined data types.
- 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 eventing model which can be used to monitor the state of registered resources.
- A redundancy mechanism to reliably store registered resources in the event of RMF peer failures.



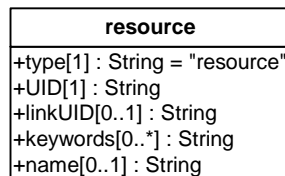
**Fig. 1.** The RMF architecture consists of four layers where each layer represents a functional unit required in the construction of non-monolithic P2P systems.

The diagram given in Fig 1. shows the layers of the RMF architecture. The application layer represents all applications which are developed to make use of the RMF. The RMF layer, which implements the resource registration, database, redundancy, leasing, lookup and event subscription mechanisms is sandwiched between the *RMF API* and the *Communicator API*. The RMF API, provides the registration, lookup and event subscription features of the RMF layer to the application developer. The Communicator API builds a layer of abstraction between the details of specific P2P protocol implementations and the RMF layer by providing an interface which must be implemented by a specific P2P protocol before it can be used by the RMF layer. In addition to the Communicator API itself, a set of semantics relating to the use of the Communicator API have been defined to which the P2P protocol developer must adhere. The Communicator layer represents all P2P protocols which correctly implement the Communicator interface whilst the network layer represents all network protocols which are used by the communicator layer. It is the responsibility of the communicator layer to distribute resources, queries and event subscriptions to peers in an RMF network based on keys assigned to resources by the RMF layer. In our prototype, keys are integer values generated by a hashing algorithm and the communicator layer is an internally developed Chord [6] protocol implementation.

### 3 RMF Resources

The RMF provides a mechanism for describing *entities*, e.g. things which exist in the real or electronic world, in terms of metadata. To this end, a universal, extensible data model for entity descriptions is given using XML. The base type of this model is defined by the *base resource definition*. By using inheritance, the user may extend the base resource definition by creating *user resource definitions*<sup>1</sup>. Resource definitions consist of typed properties which represent the metadata information of an entity. Resource definitions may, for example, be described using RDF or XML schemas [17]. Currently our prototype implementation uses Java classes for this purpose. In the RMF, a *resource* is an instance of a resource definition. Therefore, a resource can represent, in terms of metadata, entities such as: files, web pages (e.g. by using web site descriptions in RDF [13]), services (e.g. by using WSDL [14] descriptions), people, books, cars, abstract data and associations.

Every application may define its own set of resource definitions. The RMF however only knows the semantic meaning of the properties, known as the *base properties*, described in the base resource definition as shown in Fig. 2. The base properties are used to identify resources and to build data structures between them.



**Fig. 2.** The base resource definition visualized with UML. The base resource definition contains all the base properties required by the RMF.

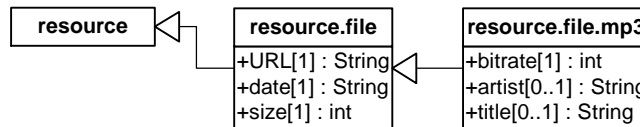
The base property `type`, which is mandatory, contains the fully qualified name of the resource definition this resource is an instance of and therefore represents the *resource type*. The `UID` property, which is mandatory, is used to uniquely identify a resource. Resources with the same `UID` are considered to be equal. The `linkUID` property, which is optional, is the `UID` of another resource to which this resource is linked. The semantic meaning of this link is application and resource type dependent. The RMF only uses the `linkUID` to group linked resources together on the same peer as described in Section 4. The `keywords` property, which is optional, is used to maintain a list of keywords to which the resource is related. It is used by the registration process described in Section 4 and allows the user to efficiently discover resources in the network. The `name` property, which is optional and is not interpreted by the RMF layer, is a user readable description of a resource.

---

<sup>1</sup> User resource definitions may also extend other user resource definitions.

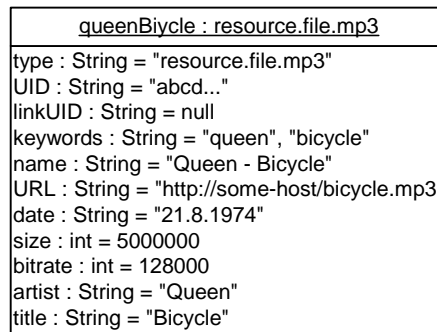
### 3.1 Resource Definition Inheritance

As previously stated, the fully qualified name of a resource definition is made available to the RMF via the base property `type`. To avoid naming conflicts and other problems, a strict naming scheme for resource types has been defined and must be adhered to. With every inheritance of a resource definition its `type` must be appended. The example given in Fig. 3 explains this mechanism.



**Fig. 3.** A resource definition that represents a file (`resource.file`) may be inherited from the base resource definition. Its `type` property, used as the UML class name, is the `type` of the base resource definition which is “resource” appended by “.file”. The file resource definition has additional properties: URL, date and size. The `resource.file.mp3` definition extends `resource.file`.

An application has the knowledge about the resource types it can handle. A file browser for example may display sets of files and their additional information (file size, date, etc.) because it knows the type “resource.file”. It can also display resources with the type “resource.file.mp3” since it is inherited from “resource.file”. It will treat these resources as normal file resources. An example instance of the `resource.file.mp3` definition is shown in Fig. 4.



**Fig. 4.** The `queenBicycle` resource. This resource represents an MP3 file located on some-host. It contains the metadata description of an MP3 file for the song “Bicycle” by “Queen”.

### 3.2 Resource Linking

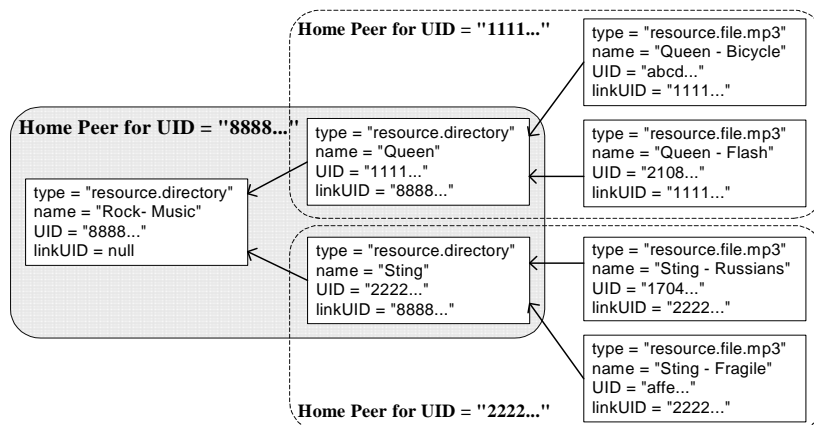
By using the `linkUID` base property, resources may be linked together. It is possible to build linked lists as well as hierarchical trees of resources. Although the `linkUID`

does not have a semantic meaning for the RMF, we call a resource linked to some other resource a *child* resource of the resource being referenced. Likewise, the resource to which some resource is linked is known as the *parent* resource of the linking resource.

## 4 Resource Registration

All RMF resources can be made available to a network of RMF peers via the `register` method provided in the RMF API<sup>2</sup>. This method is responsible for registering a given resource into the network.

Given a resource, the `register` method will register, as shown in Fig. 5, the resource with: the RMF peer responsible for the key generated from the resource's UID (known as the resource's home peer), the RMF peer responsible for the key generated from the resource's `linkUID` and all the RMF peers addressed by the keys generated from the resource's keyword list.



**Fig. 5.** The resources with the UIDs “1111...” and “2222...” are registered, in addition to their own home peer, with the peer responsible for the key generated from the `linkUID`, i.e. the home peer for the resource with the UID “8888...”. This grouping is represented in the above by the gray box. By grouping resources in this way, the user can find all resources linked to a given resource in one atomic action.

### 4.1 The Resource Database

Each RMF peer owns a resource database which is used to store registered resources during the lifetime of a given RMF peer, i.e. if an RMF peer fails or terminates, the

<sup>2</sup> In addition to the `register` method, the RMF API supports the ability to remove resources from or modify resources in an RMF database.

contents of its database may be lost without any adverse effects for the network or the RMF peer itself<sup>3</sup>. The RMF database supports five key data operations: `store`, `update`, `retrieve`, `find` and `remove`. In addition to data operations, the RMF database is required to generate four types of events, i.e. `add`, `update`, `remove` and `lease-update`, which are handled by the redundancy, leasing and eventing mechanisms.

The `store` operation adds resources to the RMF database using the resource's UID as the primary key. Resources which are added to the database for the first time generate an `add` event, whilst duplicated adds generate a `lease-update` event. The `update` operation updates a resource in the database using the resource's UID as the primary key. Successful updates generate an `update` event. The `retrieve` operation returns a resource given for a given UID and the `find` operation is used for querying or searching the database given a query statement. Given a UID, the `remove` operation removes a resource from the database and generates a `remove` event if the operation was a success.

## 4.2 Redundancy

To the application programmer, redundancy within the RMF is transparent, i.e. (s)he may register resources which are automatically made redundant by the RMF layer.

Redundancy is, however, not transparent to the P2P routing protocol developer. The RMF requires that the communicator layer maintains a group of redundant peers for a given key or range of keys and that the size of the group, i.e. the *redundancy factor*, is variable. Any changes to the state of the redundancy group, e.g. new or removed members, must be reported to the RMF layer so that appropriate action can be taken, i.e. the databases synchronized. The communicator layer is also required to reliably deliver messages to redundancy group members.

## 4.3 Resource Leases

Resource leases are transparent to both application and P2P routing protocol developers, i.e. resource leasing is carried out automatically by the RMF layer<sup>4</sup>. A resource lease, as in Jini [10], is used to make a resource available to a network of RMF peers for a limited amount of time and thus provides a form of garbage collection. If the lease for a specific resource identified by resource's UID is not renewed within the time limit set at the time of registering the resource, then the resource is removed from the lease table and the `remove` method of the database is called. A resource lease is renewed in the RMF by carrying out a resource registration with an exact copy of the resource to be renewed.

---

<sup>3</sup> This is due to the RMF redundancy mechanism provided transparently to all RMF databases. The RMF redundancy mechanism is described in section 4.2.

<sup>4</sup> It is however possible to explicitly lease resources so that a resource remains persistent if required.

## 5 Resource Lookup

Once a resource is registered in the network, applications are able to find it by using the `search` method provided in the RMF API. To perform a search, an application must know at least one peer where a resource has been registered. An application may already know such a peer because, for example, it knows the resource's parent and can therefore find the resource itself by querying the parent's home peer. Alternatively, an application must know at least one of the resource's keywords. The search request can then be directed to the peer responsible for this keyword.

The query language used by the RMF operates on the resource database as described in Section 4.1. Since this is an XML database, the query language must operate on XML nodes and be independent of the semantic meaning and structure of resources. Several query languages for XML documents exist, e.g. XPath [15], XQuery [18] and the XQL [16]. Currently our prototype uses XQL as engines exist, specifically the GMD-IPSI XQL Engine [5], which may be used in open source releases of the RMF. Additionally, the GMD-IPSI XQL Engine allows the user to create new, highly desirable, operators such as `contains` and `startsWith`.

The following XQL query when sent to the peer responsible for "queen" specifies that only resources of the type "resource.file.mp3" (or resources inherited from "resource.file.mp3" because of the `$startsWith` operator) and where `artist` contains "queen", `title` contains "bicycle" and `bitrate`  $\geq$  128000 should be returned, e.g. the resource from the example given in Section 3.1

```
/resources/resource[
  /type $startsWith "resource.file.mp3" $and$
  /artist $contains$ "queen" $and$
  /title $contains$ "bicycle" $and$
  /bitrate >= 128000]
```

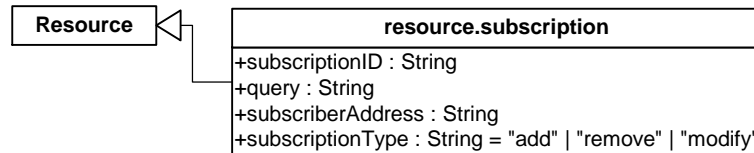
## 6 Event Subscriptions

The RMF eventing mechanism allows an RMF peer application to receive events relating to the state of database entries at a certain peer for a given query, i.e. the RMF provides a content-based publish-subscribe mechanism. The event subscription mechanism is provided to applications via the RMF API `subscribe` method. This method takes as its parameters a list of keywords which are used to address the peers which will host the subscription<sup>5</sup>, a query in the form of, for example, XQL and the type of database event which is to be monitored. When called, the `subscribe` method builds a subscription resource and registers it under the given keyword list.

---

<sup>5</sup> The keywords should be chosen carefully so that the peers hosting the subscription are likely to store resources of interest.

The UML diagram in Fig. 5 shows the resource definition used for subscription resources.



**Fig. 6.** The `resource.subscription` definition models an RMF resource for event subscriptions. The `subscriptionID` property is used by the subscribing peer application to identify which subscription a received event belongs to. The `query` property is used by the hosting peer as a filter. The `subscriberAddress` property identifies the subscribing peer. The `subscriptionType` property defines the database event which is to be monitored.

Within the RMF, the “`resource.subscription`” type is known and therefore when received, a database-modifier<sup>6</sup> is built and registered with the database in addition to the resource being stored within the database. The advantage of using a resource for event subscriptions is that, since leasing applies to all resources, event subscriptions are leased.

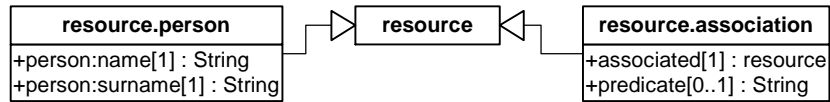
## 7 Semantic Web Style Applications

The RMF allows programmers to build Semantic Web [12] style applications, i.e. users may publish information that can be accessed and analyzed by humans or computer programs such as software agents [4]. This information may, for example, be world wide yellow pages to find a specialist, e.g. a doctor, a job applicant or a mechanic, with specific properties like location, expertise, etc. If, for example, a doctor also registers an appointment service, an agent may automatically arrange an appointment for its user.

It is also possible to publish knowledge about a group of people, e.g. the employees of a company. The following example shows how this can be accomplished with the RMF by using resource definitions for both persons and their relationships. Such knowledge is made available by using the RMF registration mechanism. Users or agents are able to access this knowledge by using the RMF lookup and eventing mechanisms.

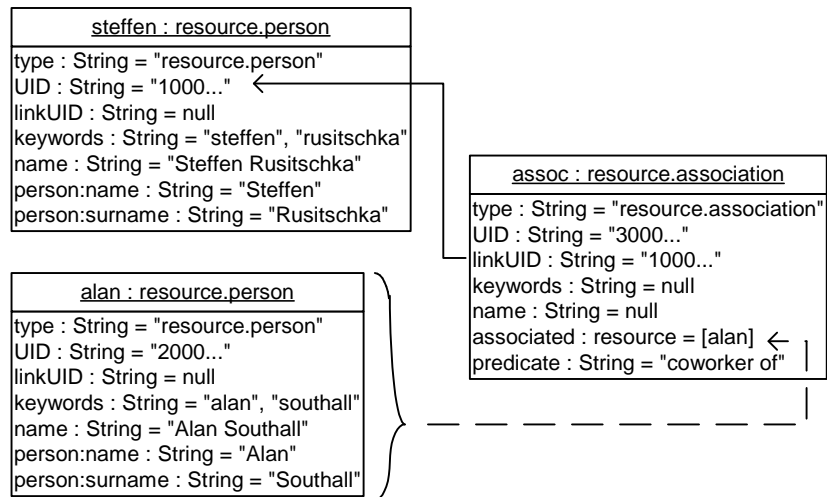
---

<sup>6</sup> A database-modifier is specific to the RMF database and allows new database event handlers, in addition to the redundancy, leasing and eventing mechanisms, to be registered.



**Fig 7.** A person is represented by the `resource` definition. In this example, every person has a name and a surname. For properties, a naming scheme has been used to resolve the name property conflict. A relationship is modeled with the `resource.association` definition which has an `associated` resource and a `predicate`.

As seen in Fig. 7, an association resource appears to contain only one associated resource, when in actual fact an association associates two resources. The second resource is extracted from the `linkUID` property of the `resource.association`. The UML diagram given in Fig. 8 visualizes this mechanism.



**Fig 8.** The resources `steffen` and `alan` are instances of `resource.person`. `assoc` is an instance of `resource.association`. The `linkUID` of the `assoc` resource is equal to the `UID` of `steffen`, i.e. it is a child of `steffen`; its `associated` property contains a copy of the `alan` resource and the `predicate` is "coworker of". An application with the knowledge of the semantic meaning of the resource definitions can extract the knowledge "Alan is a coworker of Steffen".

With the resources described in Fig. 8 registered in an RMF network, it is possible to use the following RMF query to retrieve all the coworkers of "steffen" (`UID = "1000..."`) by extracting the `associated` property from the returned association resources:

```

/resources/resource[
  /type $startsWith$      "resource.association" $and$
  /associated/linkUID = "1000..." $and$
  /predicate             = "coworker of" ]
  
```

In this example, no RDF descriptions have been used to model the metadata. However using RDF is already possible with the RMF, e.g. by embedding RDF instances in a resource. It is also already possible to use the RMF to search for these RDF instances based on their content.

## 8 Related Work

Pastry [2] provides a mechanism for building a self-organizing network of peers in which messages are routed using keys. It allows the user to publish and find objects in the network. Objects are identified by an object identifier which is supplied by an application when registering an object to the network. Objects can only be searched with their object identifiers. It is not possible to search objects based on their content.

Scribe [3] extends Pastry with a publish-subscribe mechanism, i.e. it allows the user to subscribe to object identifiers, and is optimized for an efficient routing of the event notifications with an application layered multicasting algorithm. As Pastry it does not allow the user to specify/search for semantic information.

Unlike the subject based subscription mechanism in Scribe, content based event notification services (e.g. Siena [1] or READY [8]) allow the user to subscribe for events based on their content, i.e. their metadata. These systems are also optimized for an efficient routing of the events e.g. by using IP multicasting. However they do not support self-organization or a permanent registration of resources. Concepts addressed by these systems may be integrated in the RMF's eventing mechanism in the future.

JXTA [7] is a Java framework for building P2P applications. It provides mechanisms for announcing and finding peers, peer groups and services and for communication between peers. It does not provide any of the concepts of the RMF, e.g. resources, content-based publish-subscribe, searching and redundancy. However it is possible to implement an RMF communicator layer based on JXTA using, for example, a JXTA Chord implementation to simplify the bootstrap process of peers.

Jini [10] supports the registration, lookup and subscription of Java services based on metadata attributes. The matching algorithm however is very simple and does not, for example, allow sub-string searches. Additionally it does not efficiently scale to large networks, e.g. the Internet.

## 9 Conclusions

We have given a brief introduction to the RMF. The RMF attempts to offer a unified model of P2P computing which is independent of underlying protocols and offers support for both the P2P protocol *and* application developers. Unlike other P2P frameworks, the RMF offers a highly generic, extensible data model which allows complex, inter-linked data structures to be distributed and efficiently discovered across a network of peers. By using this data model in combination with an XML database and query language, the RMF provides application developers with a powerful resource lookup capability and a content-based publish-subscribe mechanism. The

layering structure of the RMF is such that all of these features are encapsulated into one reusable unit. To summarize, the RMF, in combination with an appropriate communicator layer, provides a fully distributed, self-organizing, object-oriented, active XML database.

As a network of RMF peers is active, it seems logical that a user cannot, and indeed has no desire to, personally handle all events generated by the network. Autonomous applications, i.e. agents, which can collect, analyze and react to events will, we believe, play a vital role in RMF applications. The demonstration of Semantic Web style applications enabled by the RMF as shown in Section 7 tends to support this proposition. In a future project, we intend to explore the possibility of combining the RMF into the agent management layer of the FIPA [4] architecture.

The RMF prototype developed by Siemens Corporate Technology is currently under evaluation in an internal project. During the course of this project we intend to address security, reliability and scalability issues in large, inter-corporate, networks.

## References

1. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf: Design of a Scalable Event Notification Service: Interface and Architecture. Technical Report CU-CS-863-98, Department of Computer Science, University of Colorado, August 1998
2. A. Rowstron and P. Druschel: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001
3. A. Rowstron, A-M. Kermarrec, M. Castro and P. Druschel: SCRIBE: The design of a large-scale event notification infrastructure. NGC2001, UCL, London, November 2001
4. FIPA: The Foundation for Intelligent Physical Agents. <http://www.fipa.org>
5. GMD-IPSI: XQL Engine. <http://xml.darmstadt.gmd.de/xql>
6. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, San Deigo, CA, August 2001, pp. 149-160
7. Project JXTA. <http://www.jxta.org>, <http://www.sun.com/software/jxta>
8. R. Gruber, B Krishnamurthy, and E. Panagos: An Architecture of the READY Event Notification System. In Proceedings of the Middleware Workshop at the International Conference on Distributed Computing Systems
9. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker: A scalable content-addressable network. In Proc. ACM SIGCOMM, San Diego, 2001
10. Sun Microsystems: Jini Network Technology Specifications <http://www.sun.com/software/jini/specs/>
11. T. Friese: Selbstorganisierende Peer-to-Peer Netzwerke. Diploma Thesis, Department of Computer Science, Philipps-University Marburg, March 2002
12. Tim Berners-Lee, James Hendler and Ora Lassila: The Semantic Web. Scientific American, May 2001
13. W3C: Resource Description Framework (RDF). <http://www.w3.org/RDF>
14. W3C: Web Services Description Language (WSDL) 1.1. <http://www.w3.org/TR/wsdl>
15. W3C: XML Path Language (XPath). <http://www.w3.org/TR/xpath>
16. W3C: XML Query Language (XQL). <http://www.w3.org/TandS/QL/QL98/pp/xql.html>
17. W3C: XML Schema. <http://www.w3.org/XML/Schema>
18. W3C: XQuery 1.0: An XML Query Language. <http://www.w3.org/TR/xquery>