Peer to Peer Overlay Network for Sensor Networks

Master Thesis

Submitted to
Institute of Operating Systems and Computer Networks
Technical University Braunschweig

by

Husam Alzaq

7. February 2007

1. Referent:	Prof. Dipl.-Ing. Michael Beigl
2. Referent:	Prof. Dipl.-Ing. Lars Wolf

Erklärung

Ich versichere, die vorliegende Masterarbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Ort, Datum	Unterschrift

Acknowledgment

This Master of Science Thesis was written during winter 2006 at the Institute of Operating Systems and Computer Networks at the University of Braunschweig. I want to thank Prof. Dr.-Ing. Michael Beigl and Dr. Christian Decker for the support and guidance given during the writing of this thesis. I want to thank also Mr. Frank Strauss, IBR network and systems administration, for his technical support in Linux. I would like to thank Jxta Community for their valuable support. Finally, I would like to express my deepest gratitude to my family for their endless love and support during my study at the Technical University of Braunschweig.

Abstract

With the advent of smart sensors, it is now possible to monitor and observe the world at unprecedented levels of granularity. These sensors form a wireless sensor network of tens or hundreds of nodes that are deployed in a remote location. Such sensor networks require a significant approach for the data management. Based on Peer to Peer System, we proposed a new middleware layer for these concealed ubiquitous devices. The flexible architecture of Peer to Peer enables wireless sensor network to be connected to the Internet through a dedicated gateway. Jxta is preferred among other Peer to Peer systems since it allows any connected device on the network to communicate and collaborate. The main function of this middleware is to enable users to discover service advertisements and then to communicate with the sensor gateway. Users can obtain sensor data with simple declarative query that is distributively processed.

Zusammenfassung

Mit dem Erscheinen der intelligenten Sensoren ist es jetzt möglich, die Welt auf einem noch nie da gewesenem Niveau von Granularität zu überwachen und zu beobachten. Diese Sensoren bilden ein drahtloses Sensor-Netz von zehn oder hunderten von Knoten, die an entfernten Stellen eingesetzt werden. Solche Sensor-Netze erfordern einen besonderen Ansatz für die Datenverwaltung. Basierend auf Peer-to-Peer-System schlagen wir eine neue Middleware-Schicht vor für solche Universal-Anlagen mit verborgenen Sensoren. Die flexible Architektur des Peer-to-Peer-Systems ermöglicht den Anschluss des drahtlosen Sensornetzes an das Internet über einen zugewiesenen Gateway. Jxta wird vor anderen Peer-to-Peer-Systemen bevorzugt, da es alle möglichen verbundenen Geräte im Netz zusammenarbeiten und kommunizieren lässt. Die Hauptfunktion dieser Middleware ist es, den Gateway Dienst zu erkennen und mit diesem zu kommunizieren. Sie erlaubt dem Benutzer, Daten von den Sensoren durch eine einfache erklärende Anfrage zu erhalten, die über einen Verteilungsprozess verarbeitet wird.

Contents

Abbreviations

AVTree	**A**ttribute-**V**alue-**Tree**
BNF	**B**ackus-**N**aur **F**orm
BOINC	**B**erkeley **O**pen **I**nfrastructure for **N**etwork **C**omputing
DA	**D**irectory **A**gent
DHT	**D**istributed **H**ash **T**able
DLS	**D**istributed **L**ookup **S**ervice
IETF	**I**nternet **E**ngineering **T**ask **F**orce
INR	**I**ntentional **N**aming **R**esolver
INS	**I**ntentional **N**aming **S**ystem
NAT	**N**etwork **A**ddress **T**ranslator
PDA	**P**ersonal **D**igital **A**ssistant
RMI	**R**emote **M**ethod **I**nvocation
SETI@home	**S**earch for **E**xtra-**T**errestrial **I**ntelligence
SA	**S**ervice **A**gent
SD	**S**ervice **D**iscovery
SDP	**S**ervice **D**iscovery **P**rotocol

SLP	**S**service **L**ocation **P**rotocol
SRDI	**S**hared Resource Distributed Index
SSTP	**S**imple **S**ymmetrical **T**ransmission **P**rotocol
TCP/IP	**T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol
UA	**U**ser **A**gent
UDP	**U**ser **D**atagram **P**ackets
UDDI	**U**niversal **D**escription, **D**iscovery and **I**ntegration **P**rotocol
URI	**U**niform **R**esource **I**dentifier
URL	**U**niform **R**esource **L**ocator
UUCP	**U**nix to **U**nic **C**opy **P**rotocol
UPnP	**U**niversal **P**lug-a**n**d-**P**lay
WSN	**W**ireless **S**ensor **N**etwotk
XML	**E**xtensible **M**arkup **L**anguage

List of Figures

List of Tables

Chapter 1

Introduction

1.1. Motivation

Information technology becomes increasingly widespread and technologies disappear into the environment. These technologies will become more interactive and attractive when they are user friendly. Internet will facilitate accessing their information from any place; however, most of these technologies rely on PC system, where data should be transferred to dedicated PC.

The basic units in Ubiquitous Computing are Wireless Sensor Nodes, small electronic components capable of sensing many types of information from the environment. They are equipped with a certain amount of computational power, communication, storage and often actuation resources.

Wireless Sensor Nodes can be found in each device, in each room, in Small Office/Home Office (SOHO) or even in our dresses. In Wireless Body Area Sensor Network, a wearable health monitoring system, is built for observing patients situation. It requires a cheap way to convey patients data regardless of their place of residence. A smart home is another example, which requires ubiquitous and location-context-aware computing to allow the environment to process information as if computational devices are everywhere, rather than actually embedding devices everywhere.

Coordinating the communication across many nodes is complex. Communication is based on topology of network, network connectivity and mobility of node. Moreover, end user want to discover and interact with sensors without concerning on their location.

Access Point(AP) may provide a better solution because it manages the communication between the users and the sensor networks. It is supplied with efficient middleware that hides system complexities in order to enhance the living style of people.

1.2. Problem Definition

Wireless Sensor Networks (WSNs) that are deployed in wide area are not able to exchange information directly. The problem will be aggravated worse when different WSNs are required to communicate with each other since a middleware layer that is able to connect these different WSNs is essential. Another hindrance is that distributing information about all objects in the WSN does not scale well as the number of sensors in the network increase. Moreover, sensors generate a bulk of data, in which users are not interested.

1.3. Objective

The objective of this thesis is to design a new scalable and robust approach for bringing sensor's information to end user over Internet, which should hide the complexity of sensor network. It must cope with dynamic change of both peers and wireless sensor.

1.4. Research Question

At the end of this theses, we will have answers for the following questions,

1. What is the best peer to peer system that can be used for constructing a middleware for sensor network?

2. How should the network architecture look like?

3. How can a distributed query processing for sensor network be achieved?

4. How does the network deal with the dynamic change of sensor nodes and the increasing number of user?

1.5. Thesis Organization

The organization of this thesis is as follows:

Chapter 2 provides an overview of the Peer to Peer System. It presents their classification and characteristics. A brief description of many well known Peer to Peer systems is also given as an example in this chapter.

Chapter 3 introduces the state of art in querying wireless sensor network. It presents the middleware and database approaches that are used in the sensor networks. It also describes different service directory protocols that are used to discover network resources.

Chapter 4 contains a thorough analysis of our proposed system through use cases and explains the functional as well as non-functional requirements for the system identified by us. In this chapter, the architecture and the system design is also described.

Chapter 5 explains some implementation details. First we describe the sensor gateway implementation. Then we describe the user agent classes.

Chapter 6 describes the evaluation of the system and presents our preliminary results. Various test criteria were defined in order to verify the proposed architecture.

In *Chapter 7* the future extensions are outlined and enhancements of the current implementation in respect of practical issues.

Chapter 2

Background on Peer to Peer Systems

In the recent years, researches on Peer to Peer system have grown significantly and a lot of applications turned on this technology to benefit from its advantages. The goal of this chapter is to present a short introduction to Peer to Peer Systems, which serves as an essential background for this thesis. It helps us to select the best system that fulfill the requirements of designing and implementing a Middleware Layer for Wireless Sensor.

Based on the classification of Peer to Peer Systems, we will take one or more cases from each category and study them. There are three reasons to perform this comparison. First we are going on design an application that will merge Wireless Sensor Application and Peer to Peer System, and this comparative analysis will greatly help us. Second, the selected Peer to Peer System should cope with the interoperability between heterogeneous platform and hardware. Third, embedded devices will gain much by using Peer to Peer overlay network, since they can form their own network independent on the Internet architectures limitations.

The first section gives a short overview on Peer to Peer Systems. After that, Peer to Peer Classification is presented in Section 2.3. Search methods that are used in Peer to Peer overlay network are introduced in Section 2.4. Section 2.6 presents the characteristics of the Peer to Peer Systems. Finally, Section 2.7 shows some of well known Peer to Peer systems followed by a detailed comparison between them.

2.1. Overview

All known Computer Systems are either Distributed or Centralized Systems as depicted in Figure 2.1. Distributed Systems are classified into two paradigms i.e. Client-Server Paradigm or Peer to Peer Paradigm. In Client-Server Paradigm, clients access the main server, such as web server or FTP server, to get data according to specific protocol governing the communication. Scalability and Fault-tolerance are the fundamental problems in Client-Server Paradigm, which motivate researchers to develop Peer to Peer Paradigm as an alternative technique.

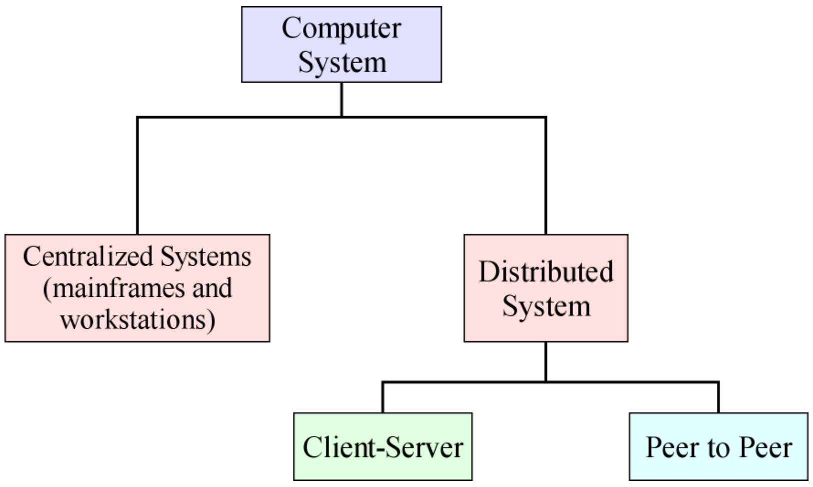


Figure 2.1.: Types of Computer System.

Authors in [1] define the Peer to Peer system as *"a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner. The resources encompass computing power, data (storage and content), network bandwidth, and presence (computers, human, and other resources). The critical function can be distributed computing, data/content sharing, communication and collaboration, or platform services."*

Practically, Peer to Peer Systems are not new. USENET was the first Peer to Peer system that provides newsgroup-service in a distributed manner. USENET was based on Unix to Unix Copy Protocol (UUCP) to allow one Unix machine peer to dial another peer, exchange files,emails or other messages with it, and finally disconnect. USENET uses Network News Transfer Protocol (NNTP), which allows peers on Usenet network to discover new newsgroups efficiently and exchange new messages in each group [2].

Peer to Peer systems have many advantages over centralized systems. Resources of many users and computers can be brought together to produce large pools of distributed information. The computing power of the system is also significantly increased when peers co-operate in solving scientific problems and the overall cost is reduced. Furthermore, the network bandwidth of the system is better utilized because computers communicate directly with their peers instead of centralized node.

Nowadays, Peer to Peer applications, particularly file sharing applications, attract Internet users. Studies in Germany, for example, have shown that from 30% (day-time) to 70% (night-time) of Germany's Internet traffic is produced by these kinds of applications [3]. Peer to Peer systems also permeate science life and many academic fields in order to use all available PC resources. Moreover, embedded devices and sensor networks are planed to benefit from Peer to Peer systems in order to allow all Internet users to access their real time data.

2.2. Basic Definition

This section introduces some of basic terminologies that are used in literatures of Peer to Peer system.

An Overlay Network is a virtual network that is built on top of existing network. This type of virtual network has the ability to route messages to any node whose IP address is unknown. The goal of such network is to offer new network services that are not available in the existing network without changing any old software or hardware.

Distributed hash tables (DHTs) are a class of decentralized distributed systems that perform the functions of a hash table. Data Keys that are used to locate data, are divided among participating nodes.

In this context, it is important to explain the difference between Peer to Peer networks, applications and protocols. Many applications carry its protocol name e.g. BitTorrent [4] is a protocol and an application name. Peer to Peer protocol is just specifications that describe how peers are communicated. Any implementation implements this protocol is called a Peer to Peer application. Peer to Peer Network is a collection of many nodes that run the same protocol by way of same or different applications. However, many well-known applications implement one or more protocol specifications and they are called Multi-network applications. BitTorrent is a Peer to Peer protocol that is implemented by G3 Torrent [5] and Shareaza [6]. Shareaza is a Multi-network application since it implements BitTorrent, eDonkey [7], Gnutella

[8] specifications.

2.3. Peer to Peer Classification

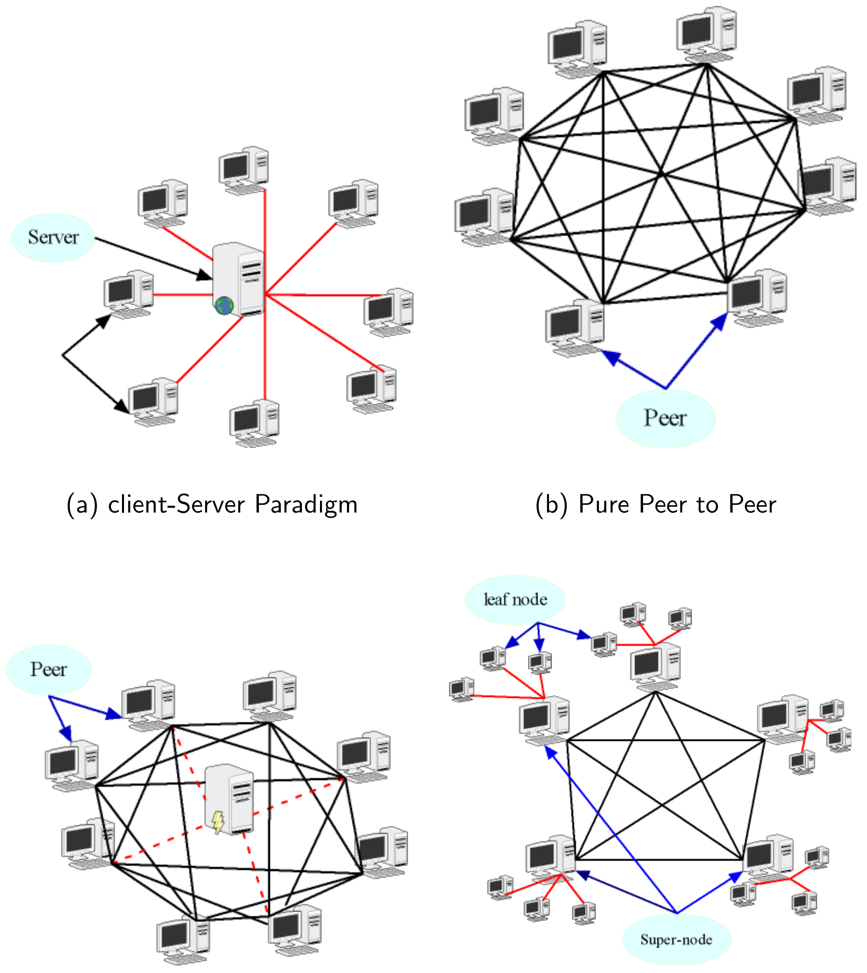
Based on the architecture of distributed nodes, Peer to Peer Systems are classified into structured and unstructured. Unstructured Peer to Peer networks have no coupling between topology and where to store data. Nodes in these networks don't manage any kind of other peers resources, i.e. index information of resource $k \in K(p)$ is managed only by Peer p itself and p doesn't control any information of its neighbors $N(p)$ [9]. This simple scheme has no single point failure, but leads to complex search operations. Unstructured Peer to Peer systems employ a blind search technique, which inundates all peers with a request before resolving it. Gnutella[8] is a typical example of this class.

Structured Peer to Peer networks have a set of tightly controlled nodes. In these networks files are not randomly stored at any node [10]. However, additional expenditure for managing this information becomes necessary, such as maintaining routing table. The structured topology of these networks is usually constructed by using Distributed Hash Table (DHT), which makes searching operations more effective than unstructured one [11]. Examples of this class include Chord [12], CAN [13] and Kademlia [14].

Another classification, which is based on degree of coupling, classified them into tight and loosely coupled system. In tight systems there exist one group at each time, which participant can join and/or leave it. For each joined peer, a new unique, static and logic identification is assigned to them. This identifier assists other peers within the group in storing and retrieving data since there is a uniqe mapping function. Routing query to destined peer is more efficient than joining and leaving an overlay network because of overhead that is needed to maintain the network structure. On the other hand, loosely coupled systems overcome tightly coupled limitation that constrains the *"Peer population"* [9]. Chord [12] and CAN [13] are two examples for the first class whereas Freenet [15] and Gnutella are two examples of the second one.

Peer to Peer systems are also classified into three models i.e. decentralized model, hybrid model or super-peer model (Figure 2.2). In fully decentralized model (also known as pure model), all participants are equal and they have equivalent capabilities and responsibilities, such as Freenet [15]. In hybrid model, there is an intermediate node, which works as a central server or directory to facilitate peers interaction. Napster [16] and Softwax [17] are two examples of hybrid model. These systems are considered as a hybrid model because they combine pure Peer to Peer system and client/server paradigm.

Super-peer model is a compromise solution between decentralized model and hybrid model. It constitutes of super-nodes with higher capabilities than other normal nodes. These super-nodes are known also as search hub, which maintains the directory of resources in order to efficiently resolve user's queries. KaZaa (implements FastTrack protocol) [18] and Jxta [19] are two examples that employ the super-peer model.



(a) client-Server Paradigm (b) Pure Peer to Peer

(c) Hybrid Peer to Peer (d) Super-node Peer to Peer

Figure 2.2.: Client-Server Model vs. different Peer to Peer Models.

Note:
Other literatures, [20], refer to super-peer model as a hybrid model since peers are not

equal in their role. They also refer to the hybrid model as a decentralized model.

2.4. Peer to Peer Search Methods

2.4.1. Searching in Unstructured Peer-to-Peer Networks

The main challenges in unstructured Peer to Peer System are discovering peers and searching for data location in overlay networks. Several methods have treated these difficulties. Search methods are classified into three categories [21], Centralized Directory methods, Blind Search methods and Informed Search methods.

Centralized Directory Method. Centralized Directory method is used in hybrid Peer to Peer system where all peers are connected to a central server that handles all requests. Peers need to know the address of the central server to join and upon leaving the server should only inform. This model attacks the scalability issues and the server becomes a single point of failure. However, search within this central directory is very efficient. All peers forward their requests to central node that also supports complex requests.

Blind Search Methods. In Blind Search methods, peers try to broadcast their requests to all reachable peers. Old pure Peer to Peer Systems relied on Flooded Requests method since peers don't keep information about other peers. In this model, requests are broadcasted and any peer can receive them multiple times, which consumes network bandwidth.

New Peer to Peer Systems have adopted this method. For example, Gnutella uses TTL-based flooding, also known as breadth-first search (BFS) with depth Time To Live (TTL), to search for data while Freenet uses a depth-first search (DFS) with a specified depth. Random walker sends out k query messages, the walker, to an equal number of randomly chosen neighbors [21].

Informed Search Methods. In Informed Search methods a distributed directory service contributes in discovering document location. Super-peer model shows a better scalability since it restricts broadcasting within the overlay network (Figure 2.2). However, in-advance information is required to join this network and leaving it don't need extra notification messages. An important issue in this approach is to keep a high number of leafs per super-peer node in order to reduce broadcasting within the overlay network and to increase the hit ratio.

2.4.2. Searching in Structured Peer-to-Peer Networks

Searching in structured Peer to Peer overlay networks becomes more efficient than searching in unstructured networks since data are placed in determined peer thanks to DHT. Each peer in the overlay network is assigned with a unique random identification and maintains in its own storage its neighbor's identification. Based on mapping function, each published document carries an identification, which is mapped to one peer in the overlay network. For example, pSearch [22] is built on top of CAN (Section 2.7.8).

Jxta employs a loosely consistent DHT walker approach to overcome the inconsistency problem of DHT within the dynamic network. This approach is a hybrid approach that combines the use of DHT to index and locate advertisements with a limited range walker. This approach is very robust because it does not require to maintaining consistency across the rendezvous network, a stable super-peer infrastructure is well adapted for mobile networks with high peer churn rate [23].

2.5. Peer to Peer Applications

- Content Sharing
  Nowadays, the popularity of file sharing applications are significantly increased and most of the current Peer to Peer systems fall within this category. These applications allow people to search for and share any digital file between users. According to the IPOQUE survey, between 30% (day-time) and 70% (night-time) of Germany's Internet traffic in October are produced by file sharing applications (Figure 2.3). Moreover, BitTorrent and eDonkey are dominant over other file sharing applications since they together was produced 95% of all Peer to Peer traffic and they have nearly displaced previously popular networks such as Kazaa's FastTrack [3] (Figure 2.4).

  These protocols are implemented with free-license applications that enable all users to share any documents. Moreover, the ease of using such applications,

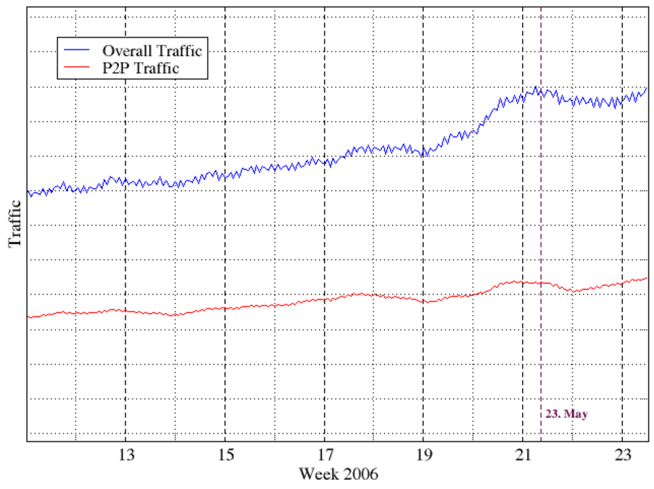


Figure 2.3.: Peer to Peer Traffic of Overall Traffic in Germany at the 2^{nd} quarter of 2006. Source: `http://www.ipoque.com/en/p2p_filter.html`.

fast Internet connection and efficient search are the main factors for popularity of file sharing applications.

- Distributed Computing:
  Peer to Peer applications have attracted a great deal of attention in all fields that need long time to compute results. Distributed computing is defined as a technique, in which peers collaborate in solving a computational problem [24]. They aim to replace parallel computers with many networked computers, which have plenty of unused resources. It was reported in December, 2006, that SETI@home was able to compute over 274.0 TFLOPS [25](1 TFLOPS is equal to 10^{12} FLOPS) while the fastest supercomputer was able to compute 280 TFLOPS.

- Collaborative Systems:
  This class of application allows collaboration between different peers. Collaborative Systems include collaborative on-line games, instance messaging programs such as AOL and MSN Messenger, audio and video chat programs such as Skype [26].

- Platforms:
  They are not real application by itself but they form an underlaying architecture for other applications. Any File sharing or collaborative application can

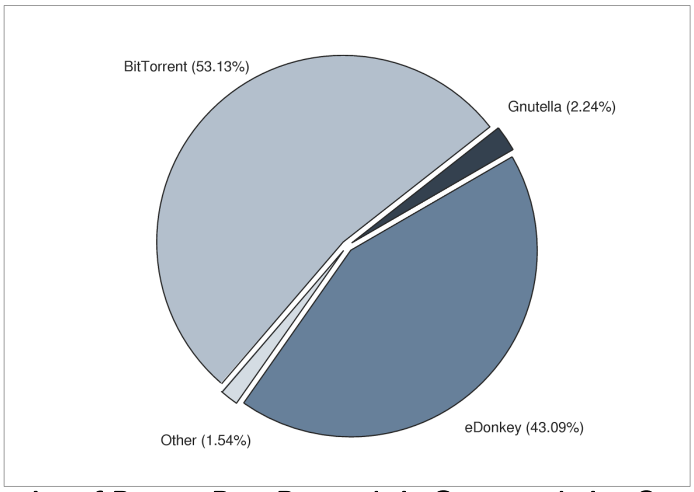


Figure 2.4.: Distribution of Peer to Peer Protocols in Germany during October 2006
Source: `http://www.ipoque.com/en/p2p_filter.html`.

be implemented over Peer to Peer platform. They support the main Peer to Peer components, i.e. naming, peer discovery, communication, security, and resource aggregation. Furthermore, they allow interoperability between different platforms. Jxta and .Net are two examples of Peer to Peer platform.

2.6. Peer to Peer Characteristics

This section addresses issues in Peer to Peer systems. These issues have a major influence on the effectiveness and deployment of Peer to Peer systems and applications. They will help us in selecting the best system that can cope with Wireless Sensor Networks requirements.

1. *Symmetry*
   Symmetry is the ability to work as client and server. It distinguishes Peer to Peer Systems from many conventional distributed system architectures that are based on client/server model. Thus, Nodes, which act as a server for other nodes, can act as a client at the same time. Symmetry aspect paves the way for Peer to Peer System to work in decentralize manner.

2. *Decentralization*
   One of the major characteristics of Peer to Peer Systems is decentralization aspect, because it removes the boundaries between clients and server. Therefore,

peers interact with each other without referring to a central node. The main advantages of this aspect are an increased extensibility, resilience to faults and higher system availability [27].

3. *Scalability*
   Scalability is defined as the ability of system to graceful extended as a new peer join it, without affecting the system performance. Millions of nodes can interact with each other and the performance of the network would not change since there is no central directory or servers.

4. *Fault tolerant*
   Fault tolerant is insured in Peer to Peer systems because the impact of a central node failure and too much load on nodes is disappeared. A central node failure and too much load on nodes decrease the overall performance of the network and may break down the whole network. Pure and Super-peer model can easily compensate the loss of a peer or even a number of peers [27], which is another advantage.

5. *Self-Organization*
   Self-Organization is an attractive aspect of Peer to Peer system because it control the system behavior and grants the system many feathers, such as self-maintenance, adaptability, rearrangement, . . . etc. [28]. Self-Organization is opposite to managed client-server architecture. Peer to Peer networks shift the management to their peer. Also, this aspect allows the network of different connected devices to be built on the fly, without any central control point as well as it automatically adapts to the arrival, departure and failure of nodes.

   Self-Organization is critical issue for Peer to Peer systems because nodes can spontaneously join the system and leave it after a while. J. Ledlie et.al. in [29] observed that about 80% of the nodes exist in a Peer to Peer system for less than one hour. Also number of users and system load is unpredictable, which increases the probability of system failures. Therefore, managing such a fluctuating environment is too hard and too expensive.

6. *Resources Sharing*
   Resources such as computing power, data (storage and content), network bandwidth, and currently, sensor information are shared and public to user through new applications and services. File sharing applications enable user to share all kinds of file. Participants in Grid computing Peer to Peer network increase the computing power capability of the network. This thesis presents a way to publish sensor information by means of a Peer to Peer System.

7. *Fast Resource Location to determine where to find the resource*
   In client/server paradigm clients know the server address, and hence requests are directed to. Peer to Peer systems establish overlay networks that efficiently route requests to any node. Be means of searching methods, peers can discover and locate any resources (Section 2.4). For example, Chord can find data using only $O(log(N))$ messages, where N is the number of nodes in the network.

2.7. Examples of well known Peer to Peer Systems

2.7.1. Napster

In 1999, Napster [16] was launched as the first Peer to Peer file sharing services. Napster has a central directory peer, which contains all required information that are required to search for any file. This peer is the only bootstrap node that admits every node to participate in the netwotk. As a result searching is effectively executed. However, overload and Denial-of-service (DoS) attack the system performance because of a single point of failure.

2.7.2. Gnutella

Gnutella [1, 8] is one of the popular file sharing protocol witch provides a simple reliable distributed system. While Gnutella (V.4) employs a query flooding method for searching, current version of Gnutella (V.6 or Gnutella2) improves searching through employing the concept of super-peer model.

Gnutella is a pure Peer to Peer system and it demonstrates high self organization since peers are not controlled. It is obvious that flood routing method affects Gnutella network and limits scalability to thousands of peers. However, Gnutella2 adapts this shortcoming by designing a super-peer model, in which many nodes are connected to one or more search hubs.

2.7.3. FreeNet

FreeNet [15, 1] is a pure Peer to Peer network for distributed anonymous information storage and retrieval system. To keep the decentralized manner, Freenet develops a heuristic Document Routing Model, also known as Key Based Routing (KBR), as a lookup method to find the closest peer that host data. KBR accomplishes routing by using a steepest-ascent hill-climbing search, where each node forwards queries to the node that it thinks is closest to the target. Each file is characterized by key, which is typically generated by SHA1 algorithm, to assist the routing function.

Freenet demonstrates good scalability because it has no central node. Freenet shows self organized since nodes are not controlled and allow to share what they want.

2.7.4. Search for Extra-terrestrial Intelligence (SETI@home)

SETI@home [30] is one of the oldest and largest distributed computing project that connects thousands Internet computer to long-time independent equations in a distributed manner. From the med of December 2005, SETI@home uses a new distributed computing platform called BOINC, which supports wide range of applications from different science area such as Biology, Climate modeling and astronomy, . . . etc.

BOINC architecture is based on client-server paradigm that consists of a server system and client software that mainly communicate with each other to distribute, process, and return work units. However, a misconception or conflict between client-server and Peer to Peer appears because there is no direct communication between peers. This kind of computing is considered as Peer to Peer paradigm because it involves shifting resource-intensive functions from central servers to workstations and desktop PCs.

2.7.5. Groove

"Groove Virtual Office, the fastest way to get everybody working on the same page" and *"Take your project with you wherever you go"* identify what Groove is! [31]. Groove is a desktop windows based collaborative and instant messaging Peer to Peer application targeted to Internet users. Groove is hybrid model, where a centralized server is used to support resource management and centralized services.

2.7.6. JXTA

The open source project Jxta [19, 32] is a generic overlay network used to create decentralized Peer to Peer network. It supports a wide range of distributed computing applications and enables them to run on multiple devices ranged from cellular phones and Personal Digital Assistant (PDA) to Oersonal Computers (PCs).

Network independent is the main aspect of Jxta, since it is only a network protocol specification that is layered on top of network protocol. Jxta provides a "*general-purpose*" network programming and computing infrastructure. Its goals comprise:

1. Interoperability: On different platform, system or medium, peers are able to communicate and exchange advertisements and messages.

2. Platform Independence: it is not restricted to specific platform, programming language or operating system.

3. Ubiquity: Jxta is intended to run on small devices such as PDAs and cellular phone as well as routers and servers.

Jxta peers form an overlay network, in which each peer has a unique identification. To increase network scalability, Jxta employs the super-peer architecture and defines two different types of Jxta peers, edge peers and super-peers. Super-peers can be further divided into relay and rendezvous peers. These peers are self-organized into Peergroups that represents a dynamic set of peers and share a common set of interests. While edge peers are the most transient player in the overlay network, rendezvous peers are the most stable one that form rendezvous super-peer network in order to maintain a loosely consistent Distributed Hash Table (DHT). In contrast to the rendezvous peers that processes queries from other peers, the relay peer allows the peers that are behind firewalls or Network Address Translator (NAT) systems to take part in the Jxta network.

To join Jxta overlay network, edge peers only need to know an address of at least one rendezvous peer. After that, peers exchange Jxta advertisements in order to discover the environment. All resources are represented by advertisements and encoded as Extensible Markup Language (XML) documents. Using advertisement ID or name, advertisements are indexed by Shared Resource Distributed Index (SRDI) service. Only the indices of the advertisement are pushed to the rendezvous peers by SRDI to minimize the amount of data that needs to be stored on the rendezvous [32].

Advertisement are published with a lifetime that specifies the availability of its associated resource. Resources are deleted from rendezvous peers after time expiration without requiring any centralized control. If the advertisement is republished before the original advertisement expires, then the lifetime of a resource will be extended [32].

Unlike file sharing protocols that restrict access between peers to download files, Jxta peers can directly communicate over Jxta pipe, a virtual communication channel that allows peers to exchange asynchronous, unreliable and unidirectional messages even when some of them are behind firewalls or NAT. Pipe offers two modes of communication [33]:

1. Point-to-point pipes connect only two peers with each other as a unicast connection.

2. Propagate pipes connects one peer to multiple destination peers as a multicast connection.

Messages are the basic unit of data that are exchanged between peers over pipes. Users data can be encapsulated in either XML-based messages or in binary messages. In XML-based messages, each message consists of elements that contains a payload of data formatted to follow XML standards.

2.7.7. Chord

Chord [34, 35, 12] is a structured overlay network that is based on DHT lookup primitive. It is also fully decentralized system that can find data using only $O(log(N))$ messages, where N is the number of nodes in the system. The most cost operation in chord overlay network are joining and leaving, i.e. they need $O((log(N))^2)$ messages.

To find data in $O(log(N))$, nodes in Chord network are arranged in a ring of 2^m where m is an integer constant. Each node maintains information about its successor and predecessor on the circle as well as information about (at most) m other neighbors in a table called finger table. Nodes are identified by a unique key from 0 to $2^m - 1$ and data elements are mapped to nodes based on their keys. If the target node is not found, the request will forward to the next available node.

2.7.8. Content Addressable Network (CAN)

CAN [35, 13] is a decentralized Peer to Peer System that provides a DHT functionality as an algorithm implemented for the document routing model. CAN organizes nodes in a d-dimensional toroidal space (a virtual multi-dimensional Cartesian coordinate space). Each node in CAN is associated with a hypercube zone and its neighbors are the nodes that *"own"* the adjacent hyper-cubes. In CAN algorithm, each data element is deterministically mapped to a point in this coordinate space using a uniform hash function. The data element is stored at the node that owns the zone in which the point lies. Using a simple greedy forwarding scheme, CAN routes a message to the neighbor peer that is closest to the destination coordinates in $O(d.N^{1/d})$ where N is the number of nodes in the overlay network and d is number of dimensions.

The cost of joining the overlay network is $O(d)$ since each zone is equally split between the new node and the old one that was responsible for it. In case of node failures, CAN algorithm reassigns zones to ensure that the structure of the overlay is maintained.

2.8. Discussion

This section attempts to find one or more Peer to Peer systems that are capable of interacting with Wireless Sensor Network (WSN). Based on their classification, we found that WSN is not an area for distributed computation. It is neither file sharing nor collaborative system. Although, file sharing systems have very primitive data model, WSN carries many aspects from both systems. Sensor information is intended to be shared with all users and they should collaborate with the sources of this data. Platform Peer to Peer systems hide network complexity and provide a clear interface.

By aggregating computation resources at hundreds or thousands of nodes, distributed computing systems are able to perform complex operation and reduce the time of computing. Peer to Peer systems that are used for distributed computing are not suitable for the purpose of embedded system for two reasons. First they have a central peer, which manages and controls all interactions. Second peers communicate only with the central peer and they are not allowed to interact with each other.

File sharing systems aggregate resources and provide an efficient method to locate shared files. Peers in WSN don't exchange files with other peers. In particular, the main resource in wireless sensor applications is sensors and sensor data should be

System	System Class	Lookup Method	Architecture	Platform
Napster	Content Distribution	Centralized Directory Model	Peers connected to centralized server	Windows & All POSIX
Gnutella	Content Distribution	Flooded Requests Model (BFS)	Unstructured P2P	All common OS(s)
Freenet	Content Distribution	Flooded Requests Model (DFS)	Loosely P2P	OS support JVM
Groove	Collaboration	Centralized Directory Model	Peers connected to centralized server	MS Windows
BONIC	Distributed Computing	Centralized Directory Model	Peers connected to centralized server	All common OS(s)
Jxta	Platform	Using search hub (Rendezvous Peer)	Super-Peer based Loosely Consistent DHT	All common OS
Chord	DHT	Hash function	Structured P2P	All POSIX
CAN	DHT	Hash function	Structured P2P	OS support JVM

Table 2.1.: Comparison between Peer to Peer systems based on their classification

public and shard with others.

In contrast to most Peer to Peer systems, Jxta is not a specific purpose Peer to Peer system, but any Peer to Peer application can be implemented on Jxta platform, such as file sharing or collaborative application. Because of its advantages, many projects use Jxta. It is reported on 31. December 2006 that 124 projects are resisted [19]. Jxta is not only available as a Java implementation, but also as a Java2 Micro Edition and C implementation. For these reasons, Jxta would serve as a good way to implement sensor application. Furthermore, Jxta search occupies a middle ground between the blind and centralized search methods. Table 2.1 and Table 2.2 compares between Jxta and other systems.

Table 2.1 shows unstructured Peer to Peer systems. Napster and Groove, are not scalable since the central node is the bottleneck one in the system. Gnutella and Jxta are examples for pure Peer to Peer system and super peers respectively. While Gnutella employs a blind search approach, Jxta restricts search for super nodes.

Table 2.2 shows characteristics comparison between different Peer to Peer systems. The comparison is based on the characteristics that has mentioned in Section 2.6. Peer to Peer architecture reflects system scalability, Hybrid systems such as Napster

System	Decentralization	Scalability	Self Organization
Napster	Hybrid	Not Scalable well	N/A
Gnutella	Fully decentralized	Thousands of peers	High
Freenet	Fully decentralized	Theoretical Scalable($log(N)$)	High
Groove	Hybrid	Not Scalable well	High
SETI@home BONIC	Hybrid (Master-Slave)	Millions of peers	Low
Jxta	Super-Peer	Scalable	Medium
Chord	Fully decentralized	Scalable	High
CAN	Fully decentralized	Scalable	High

Table 2.2.: Comparison of Characteristics of different Peer to Peer systems

and Groove do not scale well enough to support a population of more than thousands of peers while SETI@home is a hybrid system that is scalable to millions of peers. Freenet, Gnutella and Jxta are scalable system.

Final word, Jxta is a generic overlay network system that connects devices on the network ranging from embedded system and cellular phones to PCs and servers. It employs a hybrid Informed Search method. Although structured Peer to Peer systems benefit from Distributed Hash Tables (DHTs) in searching and locating objects; however, DHT does not compatible with wireless sensor applications on embedded devices since peers are tightly connected to each other and DHT maintains a consistency of data on each peers. In addition, they keep a large amount of fluctuating data on each peer. For massively scalable, high performance, and reliable Peer to Peer network, Jxta is an important step towards a middleware for sensor network.

2.9. Summary

This chapter has presented a background on Peer to Peer systems. It describes their architecture and classified them according to their architecture and degree of coupling. Search methods that are used in structured and unstructured Peer to Peer systems was also presented. Many characteristics of Peer to Peer systems such as scalability and self-organization are also highlighted .

Finally, many systems are studied in order to select the best one that can cope with Wireless Sensor Network. Thus, we appreciate Jxta as a system that can be a fundamental layer for the proposed architecture.

Chapter 3

State of the Art

With large and mobile environments comes the challenge of scalable resource discovery. Chapter 2 has presented a decentralized architecture that employs wide range of different searching techniques. Besides presenting sensor information management, this chapter aims to explain Service Discovery Approaches for two reasons. First, discovering sensors that are deployed in a building or campus is essential part of this thesis. Second, service discovery can give an advantage for pervasive computing environments, where numerous computing elements and sensors often interact to achieve the desired functionality and intelligence [36].

The first section presents approaches in the sensor data management and the main aspects of sensor networks. Section 3.2 presents the concept of Service Discovery Protocols. It is supported with different well-known service discovery systems.

3.1. Sensor Data Management

A Wireless Sensor Network (WSN) consists of a large number of sensor nodes that are randomly and densely deployed. Retrieving sensor data is the most important operation in WSN that requires query dissemination and data aggregation. This thesis addresses the problem of data dissemination through handling users queries. Sensor nodes does not process any query, but a dedicated gateway interacts between sensor nodes and users.

3.1.1. Sensor Network Architecture Aspects

Sensor network architecture aspects are categorized into hardware, software and middleware aspects. hardware aspects limits the capability of Sensor Nodes. Sensor Nodes are resource-constrained by their size and by available energy sources, usually batteries. They also have limited processing speeds, storage capacities and communication bandwidths. They must be inexpensive to produce, deploy and maintain. As applications quickly involve very large numbers of sensors, scalability is a major issue as well, leading to the use of self-organization principles.

In the software domain, there is a necessity for ultra aggressive low power management due to energy constraints and a need for comprehensive resource accounting due to demand for privacy and security, in a number of cases also support mobility function (e.g. location discovery)[37]. The network protocol is responsible for finding routes, establishing connection and accomplishing communication between sensors nodes as well as the sensor nodes and the Access Point(s). The performance is highly influenced by the network dynamics, as well as by the specific data delivery model employed [38].

The main purpose of a middleware for sensor networks is to support the development, maintenance, deployment, and execution of sensing-based applications [39]. Therefore, the middleware plays a vital role in Wireless Sensor Networks because it glues between the low level of communication and the requirements of high level. Different aspects can be considered, tasks such as sensor data filtering, compression, sensor data fusion, sensor data searching and profiling, exposure coverage and tracking will be ubiquitous [37].

WSN middleware approaches are mainly classified into three approaches[40]. The first approach treats Sensor Nodes (SNs) as a distributed database and users can issue SQL-like queries to access sensor data (Section 3.1.3). The second approach is inspired by mobile code and mobile agents, in which a program is injected into a SN and this program collects local data before migrate itself to other nodes. The last approach is event based, in which an application registers itself in a SN for a future event. When the specified event occurs, the SN notifies the application.

3.1.2. Routing Approaches in Wireless Sensor Networks

Routing is an essential part in wireless sensor network because a user can issue a query to retrieve sensor data and data should be routed to the user. An efficient routing algorithm saves nodes energy because they don't broadcast their data. Therefore, different approaches in ubiquitous sensor network focus on the routing problem and a wide range of routing protocols and techniques have been proposed.

3.1.2.1. Data-Centric Routing Approaches

Data-centric routing approaches [41] convey data to destination by flooding the initial query to the entire WSN. In these approaches, event information is stored locally at a sensor node upon detection, queries are flooded to all nodes to detect an event, and nodes with relevant information respond.

Directed diffusion approaches [42] build a high quality paths, but requires an initial flood of the query for exploration. While it is appropriate to applications with a relatively small number of data receivers, its overhead becomes high when many nodes become interested in data since each one sends its interests throughout the network, and since receivers also send exploratory data throughout the network.

Tiny AGgregation [43] has a distribution phase, in which aggregated queries are pushed down into the sensor network, and a collection phase, where the aggregate values are continually routed up from children to parents. In this approach each node is required to transmit only a single message per epoch regardless of its depth in the routing tree.

GEAR [44] is a Geographically Targeted Queries approach, in which it assumes that the event location is known and the resulted data is locally stored. Instead of flooding packets, GEAR protocol uses energy aware and geographically informed neighbor selection to route a packet towards the target region and the Recursive Geographic Forwarding to disseminate the packet inside the destination region.

3.1.2.2. Rendezvous Routing approaches

Approaches that are based on broadcast or diffusion show a worst performance when any system has too many active sources and active sinks that periodically flood the core with data. Rendezvous protocols adopt this issue by arranging for sources and sinks to meet in some predetermined way.

Data-centric storage (DSC) approach [41] does not require queries flooding; but data is named and stored at a node itself or at other nodes according to the given name to efficiently direct queries to the node that stores events of that name. The communication cost to store the data and querying them take on average $\sqrt{n}$ where n is the number of nodes. Geographical Hash Table (GHT) is a DSC implementation, in which it hashes a key k into a geographic location where data is stored at a node of that location. The problem is that the location defined by a GHT function can be quite far from the source and fundamentally requires that node knows its own geographic position.

3.1.3. Sensor Database Approaches

In these approaches, WSNs are considered as a distributed database. Sensor data is viewed as a single table with one column per sensor type. *"The advantage of the database approach is that it provides a separation between the logical view (naming, access, operations) of the data held by the sensor network and the actual implementation of these operations on the physical network."* [45].

3.1.3.1. Cougar

Cougar adopts a database approach where sensor readings are treated like *"virtual"* relational database tables. Based on user's query, a query optimizer produces an efficient query plan for in-network processing to reduce resource usage [46], so user issues tasks to the WSN with an SQL-like query language and distributed query processing is performed in sensor data aggregation, each sensor returns the data that satisfied user condition.

In Cougar database, each type of sensor in a network is represented as an abstract data type (ADT) whereas an ADT object corresponds to a real sensor. Because measurement are discretized and due to network delays, Cougar introduces virtual relations, in which a new record is inserted into the virtual relation in an append-only manner instead of updating the old one [45].

3.1.3.2. TinyDB

TinyDB [47] is a query processing system for extracting information from a network that made of motes [48], sensors developed in Berkeley Lab. Instead of programming each motes, TinyDB provides a simple, SQL-like interface to specify the data you want to extract data from WSN. Then TinyDB collects that data from motes in the environment, filters it and finally aggregates it together before returning them to the user. The only requirement for TinyDB is that, it needs to install a special operating system on each mote, known as TinyOS.

3.1.4. Query Processing in Sensor Networks

As in Database Management System (DBMS),interacting with a sensor network is preferred by means of declarative queries [49]. This enables users to issue queries without reprogramming the sensor node. However, the main differences between sensor based data sources and standard DBMS are sensors typically deliver data in streams and data is delivered at unreliable rate.

S. Madden and J. Gehrke in [50] have designed a Query Processing Architecture that consists of a base station and a software running on the sensor nodes where users input queries in SQL-like language from the base station. Query is performed through *query optimization*— selecting the best possible plan, *query dissemination* by establishing a tree-based routing rooted at either the base station or a storage point. Finally, *query processing* begins processing in each. The query processing has a simple loop: once per epoch, a special acquisition operator at each node acquires samples, from sensors corresponding to the fields or attributes referenced in the query.

User requests are expressed in a SQL query as ANSI SQL to avoid reprogramming sensor nodes each time; however, SQL for sensor networks has other clauses that aren't used in ANSI SQL such as DURATION and EVERY [49]. Moreover, each clause has a distinct meaning, the SELECT clause specifies attributes and aggregates from sensor records and the FROM clause specifies the distributed relation of sensor type. Filtering sensor records by a predicate is specified by WHERE clause. To arrange sensor records into different segment according to some attributes the GROUP BY clause is used while the HAVING clause eliminates groups by a predicate.

Query processing can be enhanced by a filtering process because filters or operations are able to reduce the amount of data routed through the network. Authors in [49] employ a set of filters, and each new packet must pass through a set of registered functions that can modify (and possibly delete) it. Directed diffusion [51] focuses on application-specific solutions that are not based on declarative query. They don't offer a particularly simple interface, flexible naming system, or any generic aggregation and join operators. Because this operations are viewed as a application-specific operators, they must be coded in a low-level language.

3.2. Service Discovery Protocols

A service is a software entity that can be used by a person, a program, or another service [52]. A service may be a computation, storage, a communication channel to another user, a software filter, a hardware device. For example, printing service may provide a user with the capability to locate and access a color or high resolution printer.

Location Service is a way to locate a specific sensor within few or thousands sensors in efficient and scalable way without using Global Positioning System (GPS) techniques. Usually sensors are attached to other objects or items and these objects need to be located. Service Discovery Protocols (SDPs) are used to enable clients within a network to find services that match their needs by hiding the complexity of administration and configuration from the users. Devices is able to discover their environment, detect and adapt to topology changes, establish communication with each other and share services [53]. They advertise their services, supply details about their characteristics and provide an access interface.

Service Discovery Protocols have many aspects, which make them distinct from other protocols. These protocols are able to determine whether user's service request match a discovered service or not. Since devices and users unexpectedly appear and disappear, Service Discovery Protocols include techniques to detect changes in component availability, and to maintain, within some time bounds, a consistent view of components in a network [54].

Service Discovery Protocols have two components, i.e. manager and user. Manager provides services to users. Each service has a Service Description (SD), which describes the service in terms of device type, service type and a set of attributes. Registry is an optional component, which maintains a service directory for the benefit of users. This optional component determines the discovery architecture of service directory. The Discovery architecture is either registry-based, which involves a registry that acts as a broker between manager and user, or Peer to Peer based, which has no registry and the system acts as an ad hoc network [53].

3.2.1. Service Directory Consistency

In Discovery Protocols architecture, advertisements may lost if they are not stored in a safe place such as a registry. Also user agent may retrieve a false advertisement or service address after the provider stopped working. These are two issues must be handled to maintain a service consistency. Three techniques are adopted for the directory service consistency maintenance; i.e. Hard-state, Soft-state and Hybrid registration.

To reduce the network traffic, Hard-state registration resources are not updated and entries never expire until explicitly deleted. However, it provides lower fault-tolerance. In contrast, soft-state registration requires recourses in service directory to refresh their information periodically, which provides higher tolerance to service failures. Soft-state registration increases the network traffic. The hybrid method is a combination of both hard- and soft-state registration.

3.2.2. Examples of Service Discovery Protocol

3.2.2.1. Salutation

Salutation [55, 56] was being developed by consortium of industrial companies, known as Salutation Consortium in order to solve the problems of service discovery and utilization of software with ambient networked devices. The Salutation architecture consists of two components (Figure 3.1), Salutation Managers (SLMs) and Transport Managers (TM). SLM, the core of the architecture, acts as a broker between service provider and clients. Similar to Directory Agent role in SLP (Section 3.2.2.2), service provider registers its service information with the local or nearest SLM while clients can lookup for a required services within the local or nearest SLM. However, each client has its own SLM and the client initially requests a service from the local or nearest SLM, which in turn may redirect the search to all other SLMs in the network.

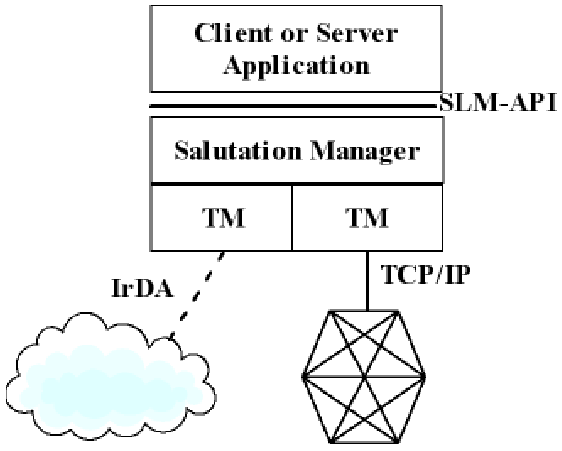


Figure 3.1.: Salutation Architecture.

Salutation supposes various ways to discover SLM, either statically by using a static table that stores the remote SLM address, or directly by specifying the remote address of SLM or by sending a broadcast discovery query. Another way, which is dependent on SLP via inquiring a central server is also possible [57].

3.2.2.2. Service Location Protocol, SLP

SLP [58, 55, 59] is an Internet Engineering Task Force (IETF) standard to enable network based applications to locate a desired service. It is an open source and platform independent protocol. SLP architecture consists mainly of three agents,

- User Agents (UA) perform service discovery on behalf of client application.
- Service Agents (SA) advertises the location and attributes on behalf of services.
- Directory Agents (DA) collect service information from SAs in order to respond to UA's requests.

SLP supports both two- and three-agent-architecture. While three-agent architecture involves all agents, two-agent-architecture involves only UAs and SAs. Figure 3.2 depicts three-agent architecture and shows the interaction between agents. Prior communication, both SAs and UAs must discover DAs via static, active or passive discovers method. In static method, they obtain the DAs through Dynamic Host Configuration Protocol (DHCP) server. In active discovery, they send service requests to the SLP multicast group address where DAs respond to this request via unicast connection with the caller. In the last method, DAs announce periodically an advertisement that carries a complete information for their service [55].

After getting the full address of DA, SA registers its service (step 1 in Figure 3.2) and receives an acknowledgment from DA (step 2) upon success. UA, in turn, sends a request for printer of fax from DA (step 3), which replays with the Uniform Resource Locator (URL) of the match service (step 4).

3.2.2.3. Intentional Naming System (INS)

INS [60] is a resource discovery and service system for dynamic and mobile networks based on effective naming system. Besides Intentional Naming Resolver (INR), which is responsible to route clients requests, INS has two types of agents; service and

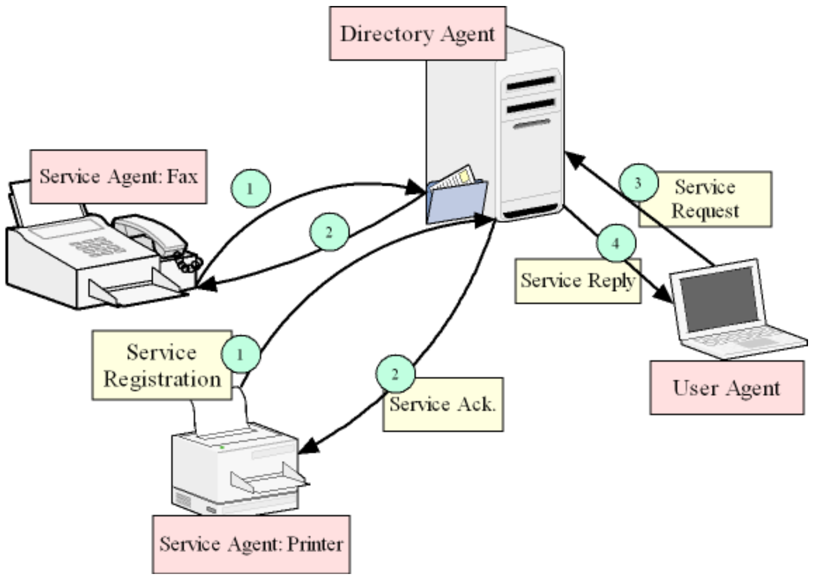


Figure 3.2.: **SLP agents, registration and discovering services.**

client agent. These decentralized resolver in INS form an application-level overlay network.

Application just describes what they are looking for by a simple language that express names into a hierarchy based schema of attribute and value (av-pair). INS service model supports three types of resolution. The first one is early binding, where an application can obtain a list of IP addresses corresponding to a name without message forwarding. The second type is late binding, an intentional multicast, where a message is forwarded to all resources that satisfy the given query, and intentional anycast, where a message is forwarded to the best resource that satisfy the query condition [60].

3.2.2.4. INS/Twine

INS/Twine [61, 62] is a resource discovery system based on INS. M. Balazinska et al. construct a directory service (called resolver) using Chord as a Distributed Hash Table (DHT) in order to partition resources description amongst a set of symmetric peer resolvers [61]. In this approach, source description is converted to the Attribute-Value Tree (AVTree) to ease storing source description and query resolving.

3.2.3. Comparison between Jxta and Service Discovery Protocols

Jxta is considered as a discovery approach, in which resources can be discovered through searching for advertisements. Table 3.1 shows a comparison between different examples of Service Discovery Protocols and Jxta.

SDP	Type of network	Architecture	Storage of service information	Search methods
Jini	Enterprise Network	Centralized	On Lookup Services	Active and Passive Discovery
UPnP	Enterprise Network	Peer to Peer	On every control point	Active and Passive Discovery
SLP	Enterprise Network	1.Centralized (with DA) 2. Peer to Peer (without DA)	1. On DA 2. On UA and SA	1.Active and Passive Discovery 2.Active Discovery of services
Salutation	Any Network	Flexible (Centralized OR P2P)	Service Registry on every SM	Salutation Manager Protocol between two SLMs
INS	Dynamic and Mobile Network	Spanning Tree overlay network	Overlay network formed by INRs	-Domain Space Resover for INR discovery. -Passive service discovery -Early and latebinding.
INS/ Twine	Large and dynamic environments handle $O(10^8)$	Overlay net- work of res- olvers fo- rming a DHT	Each resolver holds a range of keys and their values	Service discovery messages are routed in $O(logN)$ hops
Jxta	Wide-area P2P	Virtual Net- work overlay	Rendezvous peers	Loosely consistent DHT Walker

Table 3.1.: Comparison between different Service Discovery Protocols and Jxta
Source: Adopted from Marin-Perianu et al. (2005) [63].

Jini [52] and SLP employ the centralized directory of service descriptions. Salutation, SLP without DA and Universal Plug-and-Play (UPnP) [64], have no central node and act in ad hoc manner. Each device on the network can be queried about services that it offers. All these protocols are adopted for LAN (Enterprise Network). On the other hand, Service discovery protocols such as INS/Twine, INS and Jxta, are adopted for large and mobile network. INS/Twine is built on top of chord where resolvers hold a set of a keys and values while INRs form an application-level overlay network. Jxta employs a super-peer model to build a Wide Area Network (WAN) with a rendezvous peers connected to each other in an overlay network. In the later three system, special nodes used to store services/resources known as resolver, INR and rendezvous respectively.

SDP	Service Identity	Service description	Service Handler*	Expiry Time
Jini	Globally Unique ID	List of attr. and val.	Proxy stub	Soft State
UPnP	Globally Unique ID	XML based on UPnP template language	-URL -Description (XML)	-
SLP	URL/URI	Service template	URL	Soft State
Salutation	Globally Unique ID	Service ID records	Service ID	Hard State
INS	Locally Unique ID + IP:Port	Name specifiers, consisting of attr. and val.	Network Address	Soft State
INS/ Twine	Locally Unique ID + IP:Port	Hierarchies of attr. and val. XML Based	Network Address	Hybrid
Jxta	Globally Unique ID	XML Based messages	-URL -Description (XML)	Soft State

Table 3.2.: Comparison between different Service Discovery Protocols and Jxta, cont.
*Service Handler is a reference to the service outcome from lookup process.

Services, which are stored in directory, have different properties in terms of Service Identity, Service Description, Service Handle and Expiry Time. Table 3.2 compares between these aspects of the available service. All Service Discovery Protocols use either globally unique ID or locally unique ID with the full service address (IP address and port number) except SLP, which uses Uniform Resource Locator (URL). SLP, Jini, Salutation and UPnP store attribute-value pairs as lists and they do not relate attributes with each other, while INS and Twine store the attribute-value pairs as a tree hierarchy. Jxta, INS/Twine and UPnP represent data in XML format. Each discovery protocol uses a distinct service handle.

Finally, None of traditional Service Discovery Protocols are really suited for wireless sensor networks environment because they are either based on collecting information into central directories, which impose single points of failure or on maintaining a network wide multicast tree that is used for periodic service advertisements [65]. Jxta is a compromise solution between both approaches since it employs the super-peer architecture.

3.3. Summary

This chapter was divided into two sections. The first section has presented the sensor information management and sensor network aspects. Proceeding from the middleware aspects, we highlighted the queries processing in WSN. Also data and query routing algorithms are presented. In the second section, Service Discovery Protocols are described. Several examples are also studied and compared to Jxta in order to give an overview on resource locating that are currently is use.

Database approaches was worthy studied in many literatures. They have a simple interface and users can issue SQL-like query without writing a new program. However, these approaches are suitable for set of ubiquitous devices with high capabilities and resources, such as motes, to efficiently handle the injected query. Other sensors, such as Particles and μParts, have limited resources and employing these approaches directly are not useful. Next chapter presents a new architecture for these limited-resources sensors.

Chapter 4

Conceptual Design

The aim of this thesis is to design and implement a decentralized Peer to Peer system for sensor network that exposes sensors. This design would serve as a basis for an ultimate goal, implementation on embedded Linux systems where resources are restricted.

The main advantages of building a Peer to Peer application on existing Wireless Sensor Network (WSN) are significantly important. First, it hides the complexity of accessing different sensor nodes. For example, users are not interested how Particles or μParts access the USB Bridge or vice verse ([66]). Second, this system offers many services to users. Sensor node discovery and obtaining sensor data are good examples for services that are offered. The last advantage for such an application is that many distributed Wireless Sensor Networks (WSNs) can be absolutely discovered and observed.

This chapter provides an overview of the main concepts and salient features of a new system named as Jxta Middleware for Sensor Network (JMSN), which was developed during this project. In the first section, the system environment is described. Section 4.2 provides a detailed analysis of the problem. Section 4.3 outlines the architecture of the proposed system and the basic system blocks. The last section explains the structural components of the system in details.

4.1. System Overview

Sensor nodes, such as μPart from Teco [66], exchange data through a USB-bridge within small area. User, who has an attached USB bridge to his PC, is able to access sensed data.

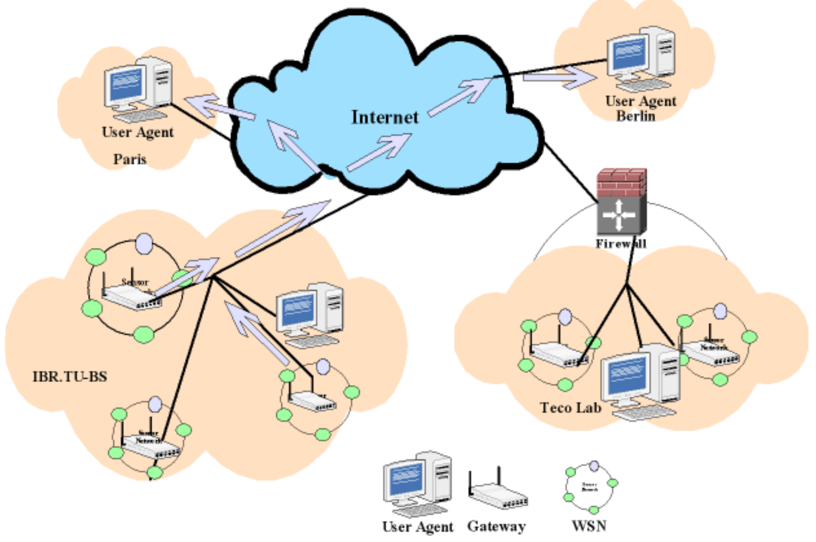


Figure 4.1.: A scenario of communication between Sensor Gateway and User Agent.

Figure 4.1 depicts the main components of the proposed system, sensor gateway and user agents. Sensor gateway gathers sensor data from nearby sensors. Any user agent can receive these sensed data after issuing a request. For example, in Figure 4.1, two users located in Berlin and Paris receive the sensed data from sensor gateway located in Braunschweig not only from the same gateway, but also from different gateways located in different rooms.

Sensor gateway in Figure 4.1 consists of three parts, i.e. sensor nodes, a USB-Bridge and a gateway. The sensor nodes itself, μPart [66], form the wireless sensor network. The USB-Bridge 1/90 ([66]) is attached to any device such as PC or router, which is connected to Internet.

4.2. Analysis

This section analyzes “Peer to Peer Overlay Network for Sensor Network” using Use Case Model to identify the system requirements. The Use Case Model “*defines the behavior of the system and its interaction with the involved actors*” [67]. The use case model consists of Use Cases. Each Use Case describes the functionality to be built in the proposed system.

4.2.1. Defining the Use Case Model

4.2.1.1. Use Case 1

Figure 4.2 illustrates how user can interact with the proposed system. The user can join a special group, which is established to gather all interested users sensor data. In order to access a gateway, a searching process is required to discover any gateway. After discovering any one, user can view all details of available services that are provided by this gateway. Even more, user can issue a SQL-like query to retrieve a special set of sensed data.

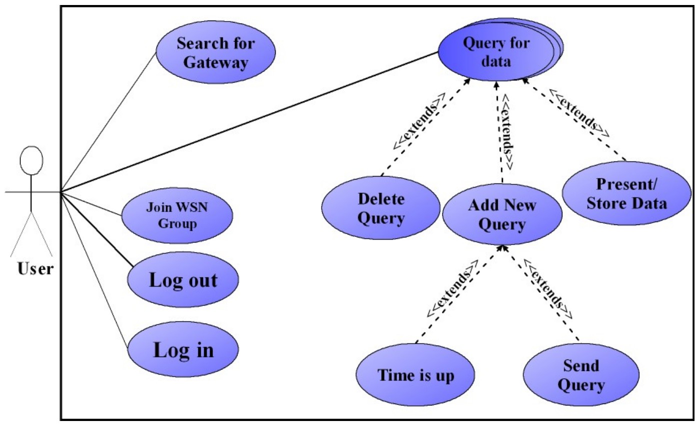


Figure 4.2.: Use Case 1: Common interaction of user.

4.2.1.2. Use Case 2

Sensor gateway acts as broker or mediator between sensor nodes and user agents since it manages the flow of sensor data from sensor nodes to user agents. Figure 4.3 illustrates how the gateway would interact with both of them. Sensor gateway should be able to discover advertisements and join a predefine group in order to meet users. The gateway records all surrounded sensors and publishes related advertisements. From user agent view, the gateway should be able to record any received filter query, remove a filter and apply these filters on received sensed data before directing it to the caller.

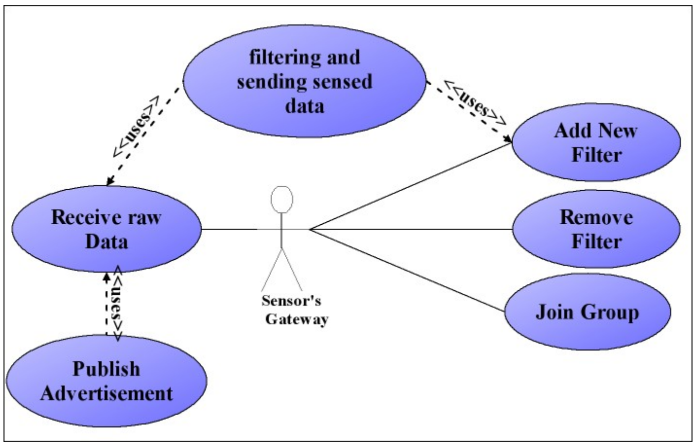


Figure 4.3.: Use Case 2: Common interaction of Sensor Gateway.

4.2.1.3. Use Case Model

Figure 4.4 depicts the Use Case model. The Use Case model combines all use cases into one case digram, in order to give an overall visual picture of what the system looks like. It illustrates how a user can interact with the system as well as how a sensor gateway interacts with the proposed system and the environment.

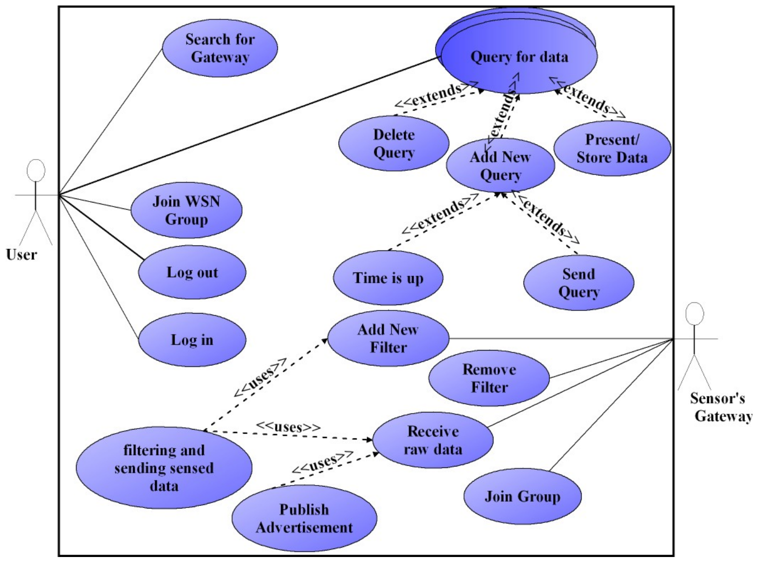


Figure 4.4.: The Use Case Model of the proposed System.

4.2.2. Requirements

In the previous section, use cases and the use case model for the proposed system are presented to identify the properties, which are important in the system. The findings from applying the use case model lead to properties that are considered as requirements for the proposed architecture. These requirements define a service, which the system should provide to end users and the constraints under which the system must operate [67]. They are classified into functional and non-functional requirements. Functional requirements describe the behavior of the system that should offer, while non-functional requirements specify properties or constraints on the system.

The main functional requirements or the application requirements are summarized in the following list,

1. Both sensor gateway and user agent should join the same group.
2. Both sensor gateways and user agents should log in with unique ID.
3. Sensor gateway should be able to publish advertisements that composed of information about its location and a meta-data on surrounded sensors and their capabilities in order to inform users about them.
4. The user agent should be able to discover gateway advertisements and show them in front of user.
5. The user agents must have the capability to express user requests in an easy way such as SQL language.
6. The user agent should be able to forward user requests to sensor gateway.
7. The user should be able to stop query processing at any time.
8. The sensor gateway must filter the sensed data before returning them to user agent according to user specification.
9. The user agent should be able to present or store the query result.

From the above functional requirements list, service discovery and data filtering are the most important services because it is obvious that not all data in the sensor network are worthwhile to user. For example, a user may want to know the temperature value at specific time or when it exceeds some limits in his house but not all raw data. Moreover, the application should facilitate searching and discovering all sensor gateway.

The non-functional requirements specify properties or constraints on the system. They are also known as technical requirements. In the following, the non-functional requirements are listed.

1. Sensor gateway should be implemented in C, since the ultimate goal aims to run the system on embedded Linux system, where resources are scarce to run heavy applications.

2. Interoperability. Users on different devices ranged from Personal Digital Assistants (PDAs) and cellular phones to Personal Computer (PCs) should be able to run the new system.

3. Usability. The proposed system should be easy to learn and use.

4. Scalability. The system must be scalable in terms of user agents, sensor gateway and sensor nodes.

5. Availability. The user should be able to search and discover gateways in the predefine group and connect to them without failure.

4.3. System Design

4.3.1. System Architecture Overview

The new system is suggested to be built atop Jxta, not only to gain many characteristics of Peer to Peer systems, but also many non-functional requirements can be achieved through Jxta. It is obvious that Jxta is a generic overlay network consisting of a set of protocols that are language, platform and network independent. Scalability and interoperability are also insured by Jxta (Section 2.7.6). Figure 4.5 illustrates the architecture of the new system. This system carries Jxta Middleware for Sensor Network (JMSN) as a name because it hides the sensor nodes from the user by means of Jxta and all communication is carried out by Jxta Layer.

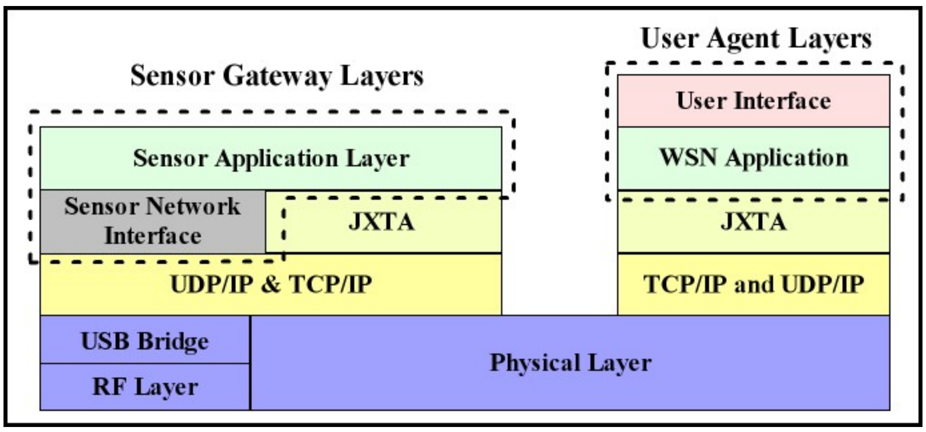


Figure 4.5.: **System Architecture.**

4.3.1.1. Sensor Gateway Layers

Sensor Application Layer. The manager of all interactions between user and sensor nodes is the Sensor Application Layer. It manages sensor data that are received from sensor nodes and all requests that are received by Jxta Layer. Moreover, it has the functionality to filter any sensor data and send out the results to the user agent.

Sensor Network Interface Layer. Sensor Network Interface Layer reads UDP packets from the sensor network, which encapsulates a tuple of data read by the USB Bridge. This layer is able to extract data from this packet and hands it to Sensor Application layer. USB Bridge and RF Layer are two sub-layers that are provided from Telecooperation Office (TecO) [66] to interact with μPart sensors.

4.3.1.2. User Agent Layers

User Agent Layer consists of two sub Layers, i.e. User Interface Layer and WSNApplication Layer. The purpose of User Interface layer is to interact with user through GUI. WSNApplication layer interacts with a gateway through message exchanging. It takes user data from the above layer, encapsulate them in a message and then forward it to the gateway. It also extracts all data from any received messages and hands them to the user Interface layer.

4.3.1.3. Jxta Layer

Jxta Layer governs the connection between the gateway and user agents, and the overlay network. This layer ensures the interoperability between the gateway and user agents. The Sensor Application Layer and WSN Application Layer interact with Jxta Layer through clear interface. For example, Jxta Layer enables Sensor Application Layer to announce advertisements and forward sensor data to user agent.

4.4. Block Diagram

The previous architecture is realized with a block diagram (Figure 4.6), which shows the interaction between system components. The right part of Figure 4.6 are mapped to Sensor Gateway Layers. Sensor Application Layer and Sensor Network Interface Layer (Figure 4.5) are represented by filter and sensor manager components. Gateway organizer, advertisements publisher and message handler represents an interface to Jxta Layer to handle peer operations. From the user agent's view, the graphical user interface represents the first layer of User Agent Layers. WSN Application Layer is represented by query analyzer and result collector components. User organizer, advertisements discovery and WSNMessanger are interfaces for Jxta Layer.

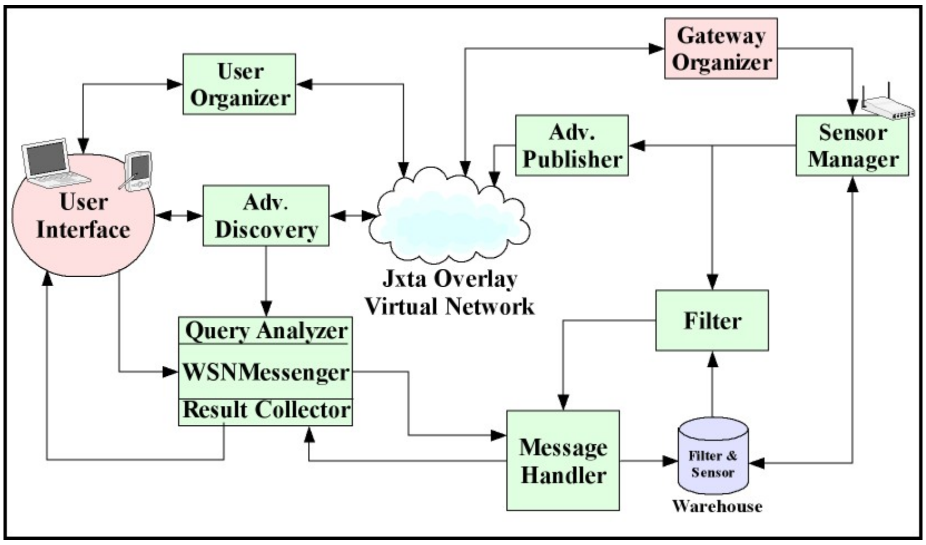


Figure 4.6.: JMSN: System Block Diagram.

Peers in JMSN. JMSN design imposes two different types of Jxta peers, i.e. Rendezvous peers and Edge peers. In Section 2.7.6, Rendezvous peer was defined as a powerful peer that processes search queries from other peers since they are super nodes while the edge peer forms the most peers in the overlay network that has a transient state of joining and leaving the network. Sensor gateway and user agent are considered as edge peers. The reason for that is, sensor gateway would be implemented on devices that have scarce resources. User agents must be edge peers because if they disappear, all resources and advertisements that they have will also disappear from Jxta overlay network unless they are replicated.

Types of Communication in JMSN. Figure 4.6 depicts also the communication process between user agent and sensor gateway. Here there are only two ways for communication between edge peers — direct and indirect communication. In the direct communication, user agent and sensor gateway exchange XML-based messages via *Jxta Pipes* without passing onto rendezvous peers. On the other hand, the indirect communication requires the interaction from both of them with the Jxta overlay network via *publishing and discovering advertisements*. For the purpose of illustration, the direct communication is shown outside of Jxta overlay virtual network.

4.5. Description of Components

4.5.1. User and Gateway Organizer

Organizer components are responsible for initializing JMSN and performing all peer operations. This ensures that all edge peers join the same group. Organizer starts with bootstrapping, an important step in Jxta. "*Bootstrapping specifies how users and services establish contact with the discovery system*" [68]. It helps both to join a peer to peer network with prior knowledge or pre-configuration on the network.

4.5.2. Sensor Manager

Sensor Manager is an interface between Jxta Layer and Sensor Network Interface Layer (Figure 4.5). It receives sensor packets and stores meta-data about surrounding sensors in a special warehouse. As shown in Figure 4.6, **Sensor Manager** activates **Advertisements Publisher** component in two cases, when new sensor is discovered or when data in the warehouse become too old. It activates also the filter component to process the received packet.

4.5.3. Advertisements Publisher

The gateway announces sensors to interested users through publishing advertisements. Publishing advertisements is a process, in which sensor gateway publishes its location and a meta-data about surrounding sensors and their capabilities.

4.5.4. Advertisements Discovery

Discovery process is a process to find an advertisement for sensor gateway. These advertisements include a set of properties that describe the sensor gateway and a set of attributes that are provided by each sensor node. The set of properties that describes the sensor gateway includes the name of gateway (*GatewayName*), the location of gateway (*GatewayLocation*) and the identification address of a sensor (*SensorID(s)*). The other set includes types of data that the specified sensor can measure i.e. temperature, light . . . etc. and their units.

Advertisements are important because they help users to establish a connection with the gateway. Furthermore, without these advertisements, user does not aware of any services provided by the sensor gateway.

4.5.5. Filtering Data

Filtering is defined as a way to suppress sensor sensor's tuples from randomly propagation and hand them in graceful manner to all interested users. It should able to answer "*to which user should I send this sensed data*". Practically, sensor gateway receives many tuples from distinct sensor nodes that are not important to all users. Filter model forwards only a subset of these tuples. Therefore, we consider it as a main part of sensor gateway.

4.5.5.1. Filtering Mechanism

Filter considers sensor data tuple as a raw of data, which is composed of set of values that are required for Filter mechanism. These tuples are discrete data since they are generated by sensors at discrete time, and they are not stored in a data warehouse.

Filter mechanism is based on attributes, which are described by key-value-operator. Key indicates the semantics of the attribute [69](temperature, light, . . . etc.). Operator describes how attributes will match when two attributes are compared. Typical attributes are sensor reading, meta-data and internal state. Sensor reading includes, temperature, voltage, light, movement and acceleration. Meta-data encompasses general information about sensor node such as sensor ID. Internal state is any information that is not related to sensor reading, such as timestamps.

4.5.5.2. Filter Data Model

Since users send their requests in different format than sensor data, the design of Filter imposes a model for data that should be handled. Data Model is an abstract model that describes how data is represented and used [67]. Listing 4.1 describes the filter data model.

In this model, *creatorName* represents a user, who creates the filter and who should receive the results. *QeuryID* is an attribute that distinguishes between user's query. Each filter has a *key*, which specifies the type of filter, e.g. is it a temperature or light filter. Moreover, It requires also two values that indicate the arguments of operation. These values are necessary for filter operation *op* because this operation determines the type of comparison between them and the sensor's values. Finally, *duration* specifies the life time of the filter process.

Listing 4.1: Filter Data Model

```
        creatorName /*Specify a user, who creates the filter*/
        qeuryID /*Specify the ID of user query */
        key /*Type of the type of filter, Temperature*/
        v1  /* First operator*/
        v2  /* Second operator*/
        op  /*Operation*/
        duration /*Specify the life time*/
        next /*pointer to the next element*/
```

4.5.5.3. Filtering Algorithm

Algorithm 1 presents the filtering mechanism that processes any received tuple. This algorithm has two inputs — the set of all filters and the received tuple. The output is sending this tuple for each user, if it matches his condition.

This algorithm is controlled by a while loop, which goes through all filters. Based on each filter key, a *compare()* procedure is called with the corresponding sensor value, filter operation and two operators. For successful operations, *send()* procedure is called to send out this tuple and user is not notified for unsuccessful operations.

```
input  :  F is a pointer to non-empty list of Filters
input  :  t is a Tuple of sensor data
Result: for each successful comparison, send t to F.creatorName
1  /*check for end of list*/
2  while $F.next \neq null$ do
3      if  $F.key$ = 'Any' then
4          send(F.creatorName,F.QeuryID ,t)
5      end
6      if  $F.key$ = 'Temperature' then
7          if  compare(t.Temperature,F.op,F.v1,F.v2) = True then
           send(F.creatorName,F.QeuryID ,t)
8      end
9      if  $F.key$ = 'Light' then
10         if compare(t.Light,F.op,F.v1,F.v2) = True then
           send(F.creatorName,F.QeuryID ,t)
11     end
12     if  $F.key$ = 'Voltage' then
13         if  compare(t.Voltage ,F.op,F.v1,F.v2) = True then
           send(F.creatorName,F.QeuryID ,t)
14     end
15     if  $F.key$ = 'Movement' then
16         if  compare(t.Movement ,F.op,F.v1,F.v2) = True then
           send(F.creatorName,F.QeuryID ,t)
17     end
18     F $\leftarrow$ F.next /*get the next element*/
19 end
```

Algorithm 1: **Filtering Algorithm**

4.5.5.4. Filtering process cycle

Filtering process passes through five states, i.e. creation, deactivation, activation, send and finally ends with the filter deletion. Figure 4.7 depicts these states. Filter

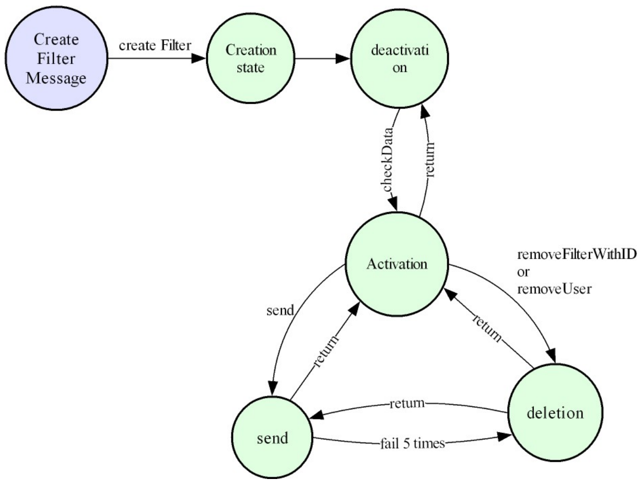


Figure 4.7.: Filter States.

process cycle is initiated by filter creation. User query is the only source for creating new filter component. Any QueryMessage received by **Message Handler** is saved. QueryMessage includes fields that are required for filter processing. These fields are explained in Query Data Model, Section 4.5.6.1.

In the deactivation state, **Filter** is waiting for an activation signal. When any sensor tuple is detected, **Filter** is activated. In the activation state, filter processes this tuple, which can activate the send state.

Filtering process is terminated by filter deletion. There are two sources to remove the filter from the warehouse, either user issues a remove message or filter time is ended. In the last method, the duration field is compared each time against the time.

4.5.6. WSNMessenger

WSNMessenger is a user agent component that specialized to carry out all communication necessary with the sensor gateway. It is dedicated to relay user queries to gateways and to collect results that are received. Communication with sensor gateway is performed over *Jxta Pipe*, in which XML messages are exchanged.

4.5.6.1. Query Analyzer

JMSN is characterized by a Query Analyzer, which is used to process user declarative query. It allows user to write a SQL-like query to retrieve sensor data and encapsulates it in a message, which is sent to the gateway.

Query Syntax. JMSN employs a simple query format that has clear semantics. Listing 4.2 illustrates the syntax of user query. In this syntax, *SELECT* clause specifies attributes in which user is interested in. The *FROM* clause specifies the name of the sensor gateway. The *WHERE* clause filters sensor records by a predicate. The *DURATION* clause, which is illustrated in the next listing, is a period of time, which specifies how long the query should be run before terminated and removed from the filter list of both agents.

Listing 4.2: Query Syntax

```
SELECT {attributes}
FROM {Sensor\_Gateway S}
WHERE {predicate}
DURATION {time interval in seconds}
```

Query Data Model. After parsing the user query, query is stored for later process. Query data model contains selectPart, fromPart and wherePart which correspond to *SELECT*, *FROM* and *WHERE* of user query respectively. The wherePart can be further separated into *key*, *operation* and *values*. The *key* is identical to a name of attribute. The *operation* is a comparison function e.g. = represents an equal operation. Finally, *values* represent arguments of the operation function.

Messages. There are mainly three messages that user can sent, i.e. RemoveMessageWithID , RemoveMessageWithUserName and QueryMessage. RemoveMessageWithID is sent out to terminate the filter processing while RemoveMessageWithUserName terminates all user's query at the destination gateway. Both messages have the structure, which is illustrated by Listing 4.3.

Listing 4.3: Remove Messages Format

```
        creatorName             /*Specify a user name*/
        qeuryID                 /*Specify the ID of user query */
        messageType             /* Type of message either RemoveMessageWithID*/
```

```
                                   /* or RemoveMessageWithUserName*/
```

Query Message includes the same fields as the filter data model except *messageType*, which represents the type of message. Listing 4.3 illustrates the query data structure.

Listing 4.4: Query Message Format

```
        creatorName     /*Specify a user name*/
        qeuryID                 /*Specify the ID of user query */
        messageType             /* Type of message*/
        key                     /*Type of the type of filter, e.g. Temperature*/
        OperandValue\_1 /* First operator*/
        OperandValue\_2 /* Second operator*/
        Operation               /*Operation*/
        duration                /*Specify the life time*/
```

Distributed Query Process. From user's view, all gateways are considered as distributed tables without any relation among them. The common characteristics among these tables are that they possess the same structure and data model. User learns about sensor gateways from advertisements discovery process. Therefore, query can't handled locally. In this design, query is distributively processed between the gateway and the user agent.

Query is distributively processed as follows:

1. The gateway name is extracted from the user query (specified in *FROM*) in order to establish a connection with the gateway.
2. The *WHERE* clause is encapsulated in a Query XML-based message.
3. Then, the Query message is sent out to the gateway.
4. After that, the received results are displayed according to *SELECT*.
5. Finally, query is terminated either by user or by gateway.

4.5.6.2. Result Collector

Result Collector is asynchronous section of WSNMessenger because it receives results in a discrete time. It presents any received results in response to user query by carrying out the projection part of query, defined by *SELECT*. This section can be used to save results on external file.

4.5.7. Message Handler

Message Handler is an interface component between Jxta Layer and Sensor Application Layer. It manages the connection between sensor gateway and all connected agents through exchanging XML-based messages over *Jxta Pipe*. This model is divided into two parts i.e. incoming- and outgoingMessageHandler.

The first one handles all incoming messages from user agents. According to the user message, it adds new filter entry or remove a specific filter entry. The second part is specialized to handle all messages that should be sent. It encloses any result of **Filter** component in a message and sent it out to user.

4.6. Summary

In this chapter, a detailed analysis of the application requirements are presented. The proposed Architecture and the basic components of its, which were built on top of Jxta and Bridge libraries, are also explained.

Discovery process and filtering process are the most important processes in the system. **Sensor Manager** handles the received sensor data and publishes them. User can discover advertisements through **Advertisements discovery** block. **WSN-Messenger** introduces new Query interface that allows user to write SQL-Query. Finally, If any filter entry was stored in the warehouse, **Filter** is activated for each new sensor data in order to send out any tuple that matches filter-condition. Chapter 5 discusses the implementation of the proposed system components.

Chapter 5

Implementation

The prototype for a sensor network over Peer to Peer, which is name as Jxta Middelware for Sensor Network (JMSN), has been developed during this project, and consists of two parts:

Sensor Gateway implements the basic functionality of the sensor agent, which controls interaction between sensor nodes and user agents.

User Agent prototype implements a user interface for JMSN, which is able to discover advertisements and access any available sensor network. It is enhanced with a simple query analyzer to handle users requests.

This chapter describes the implementation of JMSN, which was proposed in Chapter 4. Only the most important implementation decisions and rationale are introduced. The first section describes the implementation of the sensor gateway. Section 5.2 gives full details on the implementation issues of user agent.

5.1. Sensor Gateway Application

5.1.1. Technologies Used

The implementation of the sensor gateway is based on Jxta-C (version 2.5); an open source cross-platform C implementation of the Jxta platform. Jxta-C requires the following open source libraries,

- Apache Portable Runtime (APR) provides a library of routines that simplify writing a portable program (write once and compile it anywhere).
- Apache Portable Runtime Utility (APR-Util) adds package to APR containing non-core useful features.
- SQLite3 provides a view of database.
- LibXml2 provides a library of routines that support XML
- OpenSSL provides an implementation of the SSL and TLS protocols.
- zlib is a format for data that has been compressed using deflate-style compression.

5.1.2. Implementation of Sensor Gateway

At the most abstract level, the sensor gateway consists of five components i.e. Gateway Organizer, Advertisements Publisher, Sensor Manager, Filter and Message Handler. These components are implemented in JMSN in order to define the most fundamental methods for accessing the system.

A **Application_organizer.c** implements the Gateway Organizer. A **SensorManager.c** is an implementation of the Sensor Manager, which includes many functions such as *sensorApplication()* to control the application processes and *addNewSensor()* to manage sensor data. A **Publish.c** is the Advertisements Publisher and includes one function, which publishes a WSNAdvertisement (**WSNAdvertisement.c**). A **MessageHandler_Outgoing.c** and a **MessageHandler_Incoming.c** implement the Message Handler.

Finally, the **sensorLib** package contains an implementation of the Filter component and other helper functions. A **Filter.c** includes *addNewFilter()* and *removeFilter()* functions to handle all filter elements and *checkData()* to perform the filter operation. A **User.c** handles all connected users. A **Parser.c** provides an interface to parse all μPart packets.

5.1.2.1. Gateway Organizer

The gateway organizer is responsible for gateway initialization. It is a significant process in JMSN, because it connects the sensor gateway to the Jxta overlay network. Initialization is carried out in two phases, manual and dynamic phases. The manual phase is done only once by editing a *PlatformConfig* file. Running any Jxta application creates the *PlatformConfig*, which is a generic configuration in XML format for any Jxta peer. Within the PlatformConfig file, a *seeds* tag should indicate to at least one rendezvous peer (IP Address) in order to fast locate the rendezvous peers.

When the gateway is launched, the dynamic phase is always performed. In this phase, bootstrapping is achieved with the help of **PlatformConfig**. Then, control is given to **application_organizer.c** in order to perform peer operations. It includes functions such as *find_any()* to discover any Peergroup advertisements and *join()* to join a predefined group named as **WSNGroup**. After that, a new input pipe is created to handle all incoming messages from users. Finally, **Sensor Manger** gains the application control.

WSNGroup. **WSNGroup** represents a place where sensor gateway and user agent can meet each other and discover advertisements. Although it is possible to create dynamic groups, it is preferred to create a static group with a known ID and a name because it restricts the scope of advertisement searching. creating many groups can produce islands of groups, which enforces the user application to join all of them to be able to discover their advertisements.

Naming the sensor gateway is an important issue, because a suitable name defiantly improves searching process. For instance, suppose two sensor gateways are placed in two computer labs. They are given Lab145 and Lab146 as an identifier name. When users discover these names via advertisements discovery process, they may assume that these labs are adjacent. Furthermore, having a unique name is another important issues. What happens if two sensor gateways have the same names? JMSN does not impose any restriction on naming or naming conveniences and it is the responsibility of the sensor gateway operator. It is suggested to use hierarchy names as Internet naming to facilitate user searching process e.g. lab145.dus.ibr.cs.tu-bs.de and lab146.ubi.mit.edu.

5.1.2.2. Sensor Manager

Sensor Information Management is the core part in JMSN because it implements four components of sensor gateway (Figure 4.6). It implements Sensor Manger, Advertisement Publisher, Filter and Message Handler components. Figure 5.1 depicts the structural units of Sensor Information Manager.

SensorManager class represents the main class of Sensor Information Manager. It receives sensor data as a *p_packet* with help of *libparticle* library. Figure 5.2 depicts sequence diagram of detecting a new tuple. **SensorManager** calls *parse()* to extract sensor data. In case of detecting a new sensor, **SensorManager** stores it in a Sensors_List via calling *addNewSensor()* and publishes it via calling *create_WSNAdv()*. The importance of the Sensors_List is to keep a consistent view of nearby sensors, in which old sensors information must be deleted if not refreshed within a specific time (by default it is 240 seconds). Listing 5.1 shows the structure of the Sensors_List.

Sensor Manager is also responsible for activating **filter process** for each received tuple, which is discussed in Section 5.1.2.5.

Listing 5.1: sensors structure code

```
struct sensors{
        char nodeID;    /*Sensor ID*/
        int locX;               /*The X Location of the gateway*/
        int locY;               /*The Y Location of the gateway*/
        int locZ;               /*The Z Location of the gateway*/
        int time;               /*The time left to refresh*/
        sensors *next;  /*a pointer to the next sensor*/
};
```

5.1.2.3. Advertisements Publisher

Publish class is responsible for announcing sensor information through advertisements. It has only one function, *create_WSNAdv()*.

One important issue in JMSN is, how sensor gateway should publish their sensed sensor information. It implies that, should the sensor gateway publish each tuples of sensed sensor data on Internet or not?

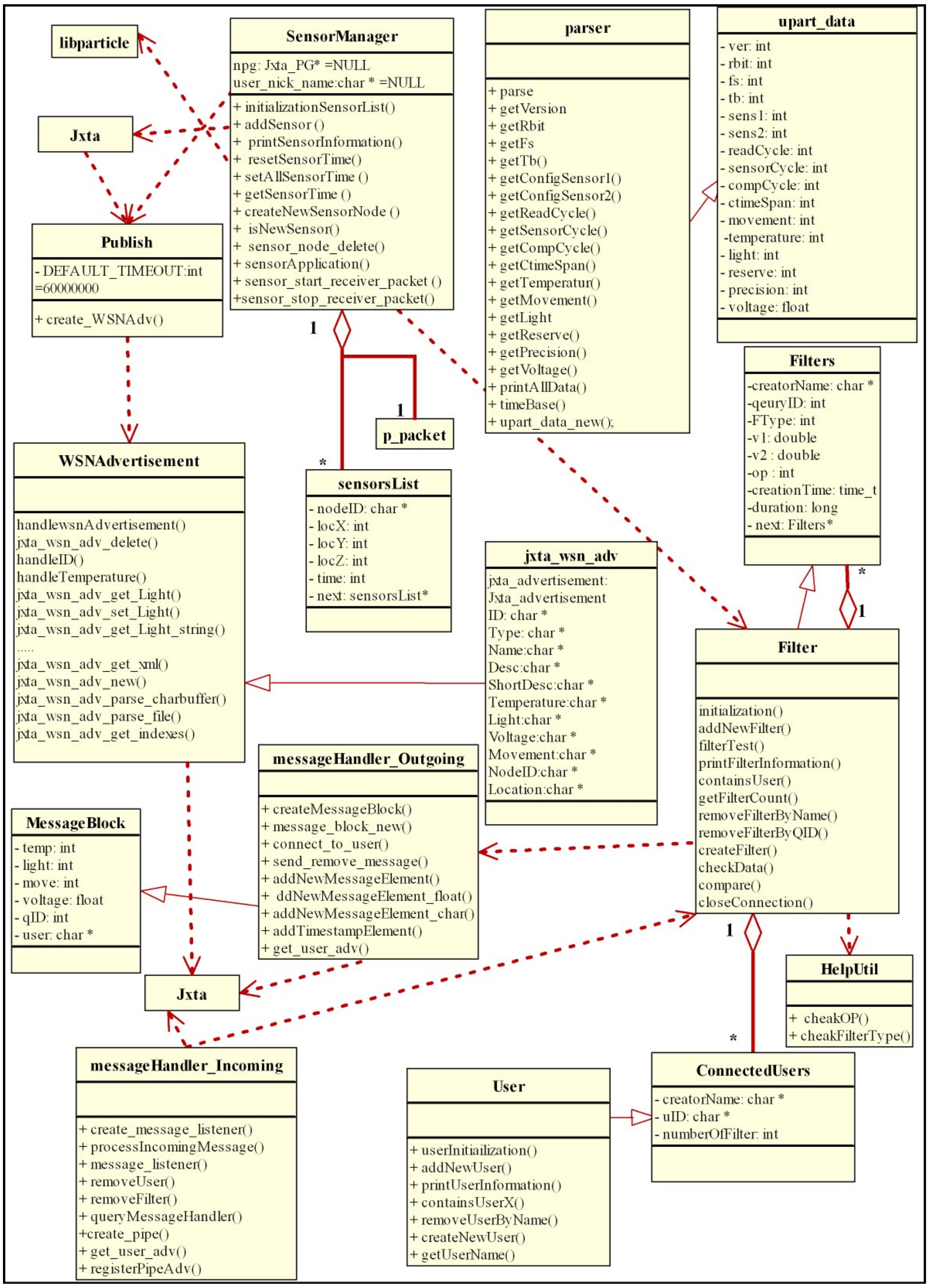


Figure 5.1.: **Structural Units of Sensor Information Manager.**

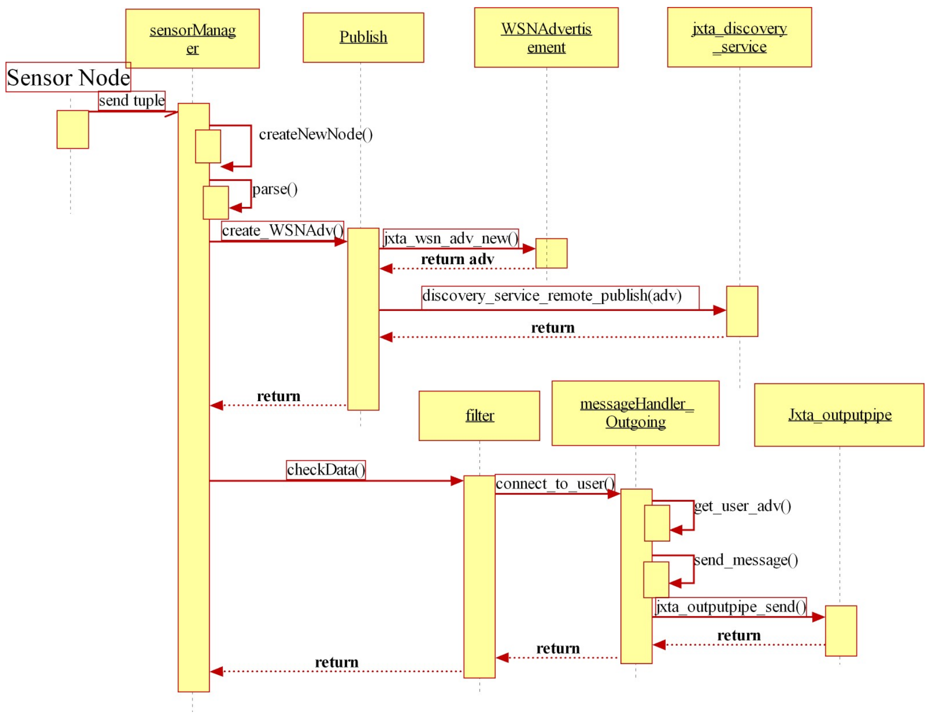


Figure 5.2.: **Sequence Diagram for Receiving a tuple from Sensor Network.**

Advertisement Method. There are three ways that can be implemented over Jxta overlay to deliver sensed sensor data. They are identical to data delivery model, i.e. broadcast, multicast and unicast models. In the broadcasting or flooding model, advertisements that contain the sensor data, are transmitted by the gateway to Jxta overlay network, in which they are indexed each time in different rendezvous peers. In this way, Jxta overlay network suffers from a bulk of advertisements and keeps all user agents in waking state by forcing them to send a periodic searching query, which does not guarantee that all advertisements are read.

The multicast model employs propagate pipes which are UDP based pipes. The shortcoming in this model is that all registered users receive sensed data whether they are interested in or not. As a consequence, data filtering can't be implemented.

The unicast model employs point to point pipes, which are TCP based reliable pipes. JMSN implements the unicast model to ensure that all packets reach the destination since it is TCP based.

Advertisements. **WSNAdvertisement.c** implements a gateway advertisement, which is used to carry information on the gateway. **Publish.c** creates a WSNAdvertisement, (Listing 5.2), which is based on sensor data that are given as arguments to *create_WSNAdv()*. This advertisement contains the name of the gateway (included in <*Name*>), that should all user agents forward their queries to. From the gateway's view, it is a name for an input pipe. From user agent's view, it represents a table of sensors data. Moreover, it contains information on what the sensor can sense, e.g. temperature, light, ...etc., and it includes other attributes such as units.

Listing 5.2: WSNAdvertisement XML code

```
<?xml version=\"1.0\"?>
<!DOCTYPE jxta:WSNAdvertisement>
<jxta:WSNAdvertisement xmlns:jxta=\"http://jxta.org\">
    <ID>
        urn:jxta:uuid-59616261646162614E504720503250338B2EEA5F3D3E4A1
        BBFB68517F6AF055A04
    </ID>
    <Type>  1                                        </Type>
    <Name>WSNAdvertisement.lab145.ibr.cs.tu-bs.de       </Name>
    <Desc>
        If you are Interested in sensor reading, just use my ID to
        reach me in order to obtain a dynamic set of them.
    </Desc>
    <ShortDescription>
        Wireless Sensor Adv. Created by alzaq Gateway.
    </ShortDescription>
    <Temperature unit=‘‘Celsius’’>     True            </Temperature>
    <Light unit=‘‘candle’’> True                        </Light>
    <Voltage unit=‘‘volts’’>    True                    </Voltage>
    <Movement>  True                                    </Movement>
    <Location> (1,2,3)                                  </Location>
    <NodeID>    1.2.3.4.0.4.0.59                        </NodeID>
</jxta:WSNAdvertisement>
```

5.1.2.4. Message Handler

Message Handler is implemented in JMSN by **MessageHandler_Outgoing.c** and **MessageHandler_Incoming.c**. **MessageHandler_Outgoing.c** manages all sending messages. Table 5.1 presents these messages. *Connect_to_user()*, which calls *send()* or *send_remove_message()* are the main functions to send messages.

MessageHandler_Incoming.c manages all received messages from a user agent. It creates a PipeAdvertisement through calling *registerPipeAdv()* and publishes it. In addition, it constructs a message listener that handles all received messages through calling *Create_message_listener()*. Table 5.1 shows three different user-messages that can be handled by *ProcessIncomingMessage()*. (PipeAdvertisement, which is an advertisement for a recourse of type pipe, is an essential part to establish a connection with the pipe's creator either the sensor gateway or the user agent).

Message Type	Meaning	Source
SENDING_QUERY	Query message	User Agent
REMOVE_FILTER_BY_ID	Remove a filter indicated by ID	User Agent
REMOVE_FILTER_BY_USERID	Remove all user queries	User Agent
QUERY_RESULT	A resulted message from testing purpose	Sensor Gateway
TIME_IS_UP	Execution is finished	Sensor Gateway
SERVER_IS_CLOSED	Server will close	Sensor Gateway

Table 5.1.: Types of messages that are exchanged between a sensor gateway and a user agent

5.1.2.5. Data Filter

After receiving **SENDING_QUERY** message from a user agent, *processIncomingMessage()* creates a new filter entry and appends it to the filter list via calling *createFilterWithDuration()*. The sequence diagram for handling this message is shown in Figure 5.3.

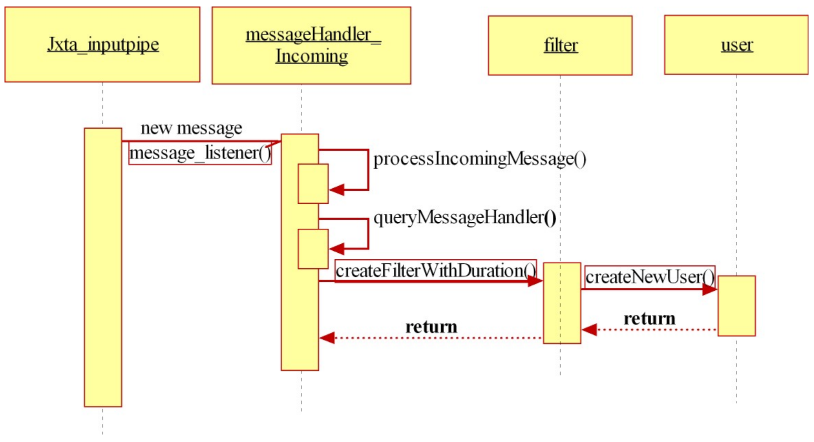


Figure 5.3.: **Sequence Diagram for handling SENDING_QUERY message.**

Listing 5.3: Filter structure code

```
struct filter_struct{
        char *creatorName; /* The user name, who creates the filter*/
        int qeuryID;/* User Query ID*/
        int FType;      /* Type of the filter, Temperature or Light or ...*/
        double  v1; /* First operator*/
        double v2;      /* Second operator*/
        int op;         /* Operation*/
        time_t creationTime;
        long duration;/* in seconds*/
        filter_struct *next;
};
```

Listing 5.3 shows the structure of a filter element. Each filter element has a *creater-Name*, which represents the the name of the user, and a *qeuryID*, which specify an identification for the user query. It has also a filter type (*FType*), an operator (*op*) and operands of the filter (*v1* and *v2*). The type of the filter and the operator are indicted by an integer value instead of string to reduce the size of occupying memory as shown in Listing 5.4 and Listing 5.5.

Listing 5.4: Types of Filter

```
#define NO_FILTER_TYPE  0
#define TEMPERATURE_TYPE 1
#define LIGHT_TYPE 2
#define MOVEMENT_TYPE 3
#define ACCELRATION_TYPE 4
#define VOLTAGE_TYPE 5
```

Listing 5.5: Query Operation

```
/* No Operator */
#define EQ_ANY 0        //No condition operator that accept any value
// Singleton Operators
#define EQ  1   //Equal operator
#define NEQ  -1 //Not equal operator
#define GT  2   //Greater than operator
#define GEQ  -2 //Greater than or equal operator
#define LT  3   //Less than operator
#define LEQ  -3 //Less than or equal operator
#define IS  4   //IS operator
// Logical Operators
#define NOT 10    // Logical NOT operator
#define OR  11    // Logical OR operator
#define AND 12    // Logical AND operator
#define XOR 13    // Logical XOR operator
// Range Operators
#define BW   5    // Between (Inclusive)
```

```
#define NBW  -5  // Not Between (Inclusive)
#define BWX  6  // Between (Exclusive)
#define NBWX -6 // Not Between (Exclusive)
```

The first operand and the second operand are used in comparison against the sensed value. The filter type, the operator and the first operand are always required for filtering process and only the range operators require the second operand such as the between operation.

Function *sensorApplication()* in **SensorManager** (Figure 5.2) activates the filter process by calling *checkData()*. This function implements Algorithm 1. For each success operation, a new message is sent out to user agent. Upon any failure to send message or user PipeAdvertisement does not exist, the number of iterations is increased till five before deleting the filter. In each activation, the filter process tries to send the result message only one time and after five trials, it assumes that the user agent does not exist. This is the only way to identify that the user leaves without removing its filters.

5.2. User Agent Application

5.2.1. Technologies Used

The user agent is built on top of Jxta (version 2.4). In contrast to the sensor gateway, the user agent is developed in Java to take the advantage of platform independent. It requires Java 2, Standard Edition 1.4 runtime. JBuilder 2006, an integrated development environment for Java, has been used to assist the development.

5.2.2. Implementation of User Agent

In addition to a graphical user interface, user agent implements the user organizer, the advertisements discovery, the query analyzer, the result collector and the WS-NMessanger components (Figure 4.6). The implementation provides the user with all methods to interact with a sensor gateway through a set of classes. The **User-GUI** class implements the graphical user interface. The **WSNGroupManager** implements the user organizer component, while **JxtaClientServices** class is implemented to handle the advertisement discovery process. **Filter**, **Qmessage** and

QueryAnalyizer are the main classes that implement the query analyzer component. The current design of the result collector is included in **WSNMessanger** class, which also is an interface for Jxta messaging.

5.2.2.1. WSNGroupManager class

Initialization is performed in two phases as in the sensor gateway i.e. manual phase and dynamic phase. Since they should join the same group, class **WSNGroupManager** is in charge of joining group.

5.2.2.2. QueryAnalyizer class

A crucial part of JMSN is the **Query Analyizer** component because it parses a user query into its main parts, validates the syntax and processes the execution in a distributed manner. JMSN employs a simple query processor, which is described by a Backus-Naur Form (BNF) grammar (Appendix A presents the BNF for JMSN Query Language). The **QueryAnalyizer** class implements a simple parser that was based on the BNF grammar. However, users are not allowed to write a complex query. Only query with SELECT, FROM, WHERE and DURATION are allowed. Moreover, WHERE clause enforce a simple condition to be used e.g. (Temperature ¿ 22) .

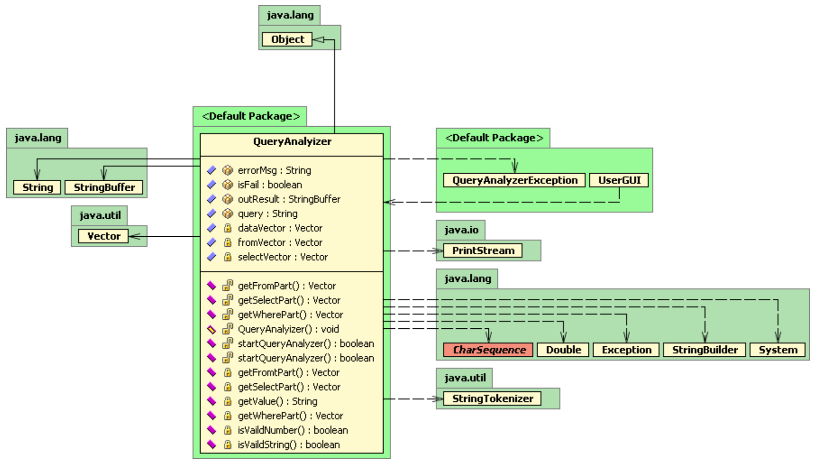


Figure 5.4.: **Detailed Class Diagram for QueryAnalyizer.**

The class diagram for **QueryAnalyizer** is presented in Figure 5.4 which contains a *startQueryAnalyzer()* to parse the user query. The result is encapsulated in **QMes-**

sage object (Figure 5.5). A **QueryAnalyzerException** is thrown in the case of miss syntax.

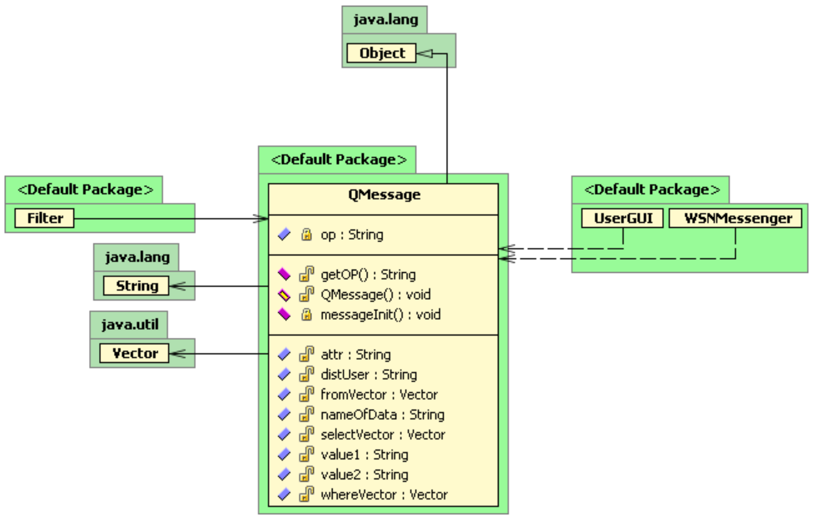


Figure 5.5.: Detailed Class Diagram for QMessage.

5.2.2.3. WSNMessenger class

The output of paring a query is an object that is handed to the **WSNMessenger** object. Before sending the query, this class discovers a *PipeAdvertisement* for the destination, and encapsulates the WHERE clause in a Jxta message. Then it forwards the message to the destination, whose name is specified within the FROM clause. Furthermore, this class presents any received results via calling *presentQueryResult()*, which performs a projection to all parts that doesn't match the SELECT clause.

The **WSNMessenger** class is not dedicated for handling queries, but it is the communication part of the user agent. It has the responsibility for handling all sent and received messages that are exchanged between the user agent and the sensor gateway. These messages are shown in Table 5.1. The class diagram of **WSNMessenger** with all the public, private and protected methods and some of the variables is shown in Figure 5.6

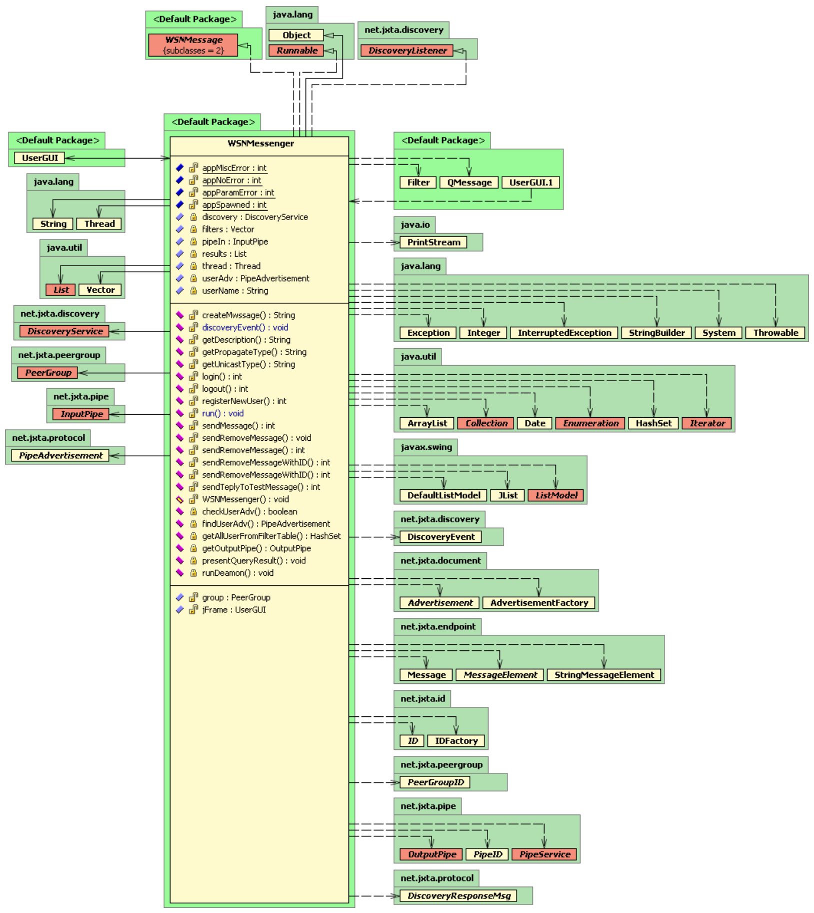


Figure 5.6.: Detailed Class Diagram for WSNMessenger.

5.2.3. User Agent Processes

The Discovery Process for WSNAdvertisements. Figure 5.7 depicts the sequence diagram of the discovery process. This process is initiated when a user issues a find service (from the user graphical interface), which searches for **WSNAdvertisements** in all connected Rendezvous Peers. After that, they asynchronously responds with the advertisements that they have. Finally, the contents of the **WSNAdvertisements** are displayed in front of the user.

Connecting Process. To send any query to a discovered sensor gateway, a user should connect to the system in order to create a unique input pipe and a listener for all received messages. It insures that the user connects with a unique name through concatenating the current time (in milliseconds), the IP address of his machine and the user name. Figure 5.8 depicts the sequence diagram of connecting process.

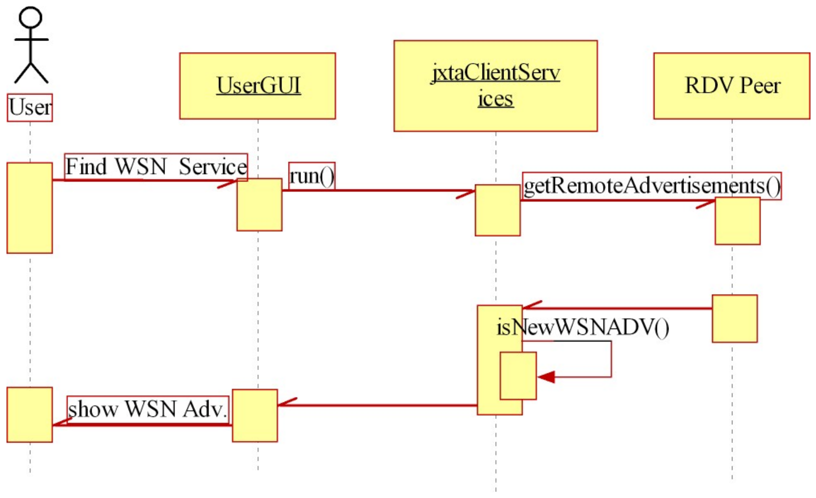


Figure 5.7.: **Sequence Diagram for WSNAdvertisements Discovery Process.**

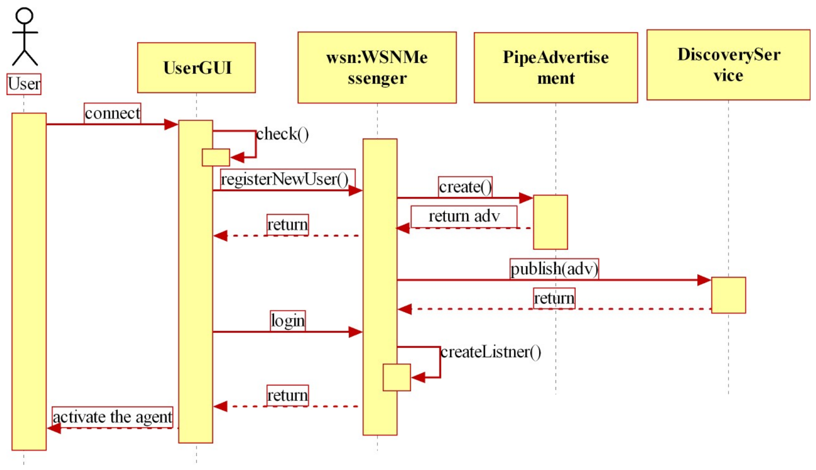


Figure 5.8.: **Sequence Diagram for Connecting Process.**

Sending Query Process. Figure 5.9 shows the sequence diagram for sending query. It illustrates that a query must be checked two times. The **QueryAnalyzer** class checks the syntax of the query through parsing it into the main three parts (SELECT, FROM and WHERE clauses). Before sending the query, the **WSNMessenger** class checks the query again against suitable attributes that match the given attributes in the destination gateway's advertisements. The processing will stop if any mistake occurs, otherwise the query will send to the destination gateway over an *OutputPipe*.

Sending Remove Message Process. To cease the query processing, a user can delete his query. The user agent takes the right QueryID and sends it to the gateway inside a **REMOVE_FILTER_BY_ID** message. This is illustrated in Figure 5.10.

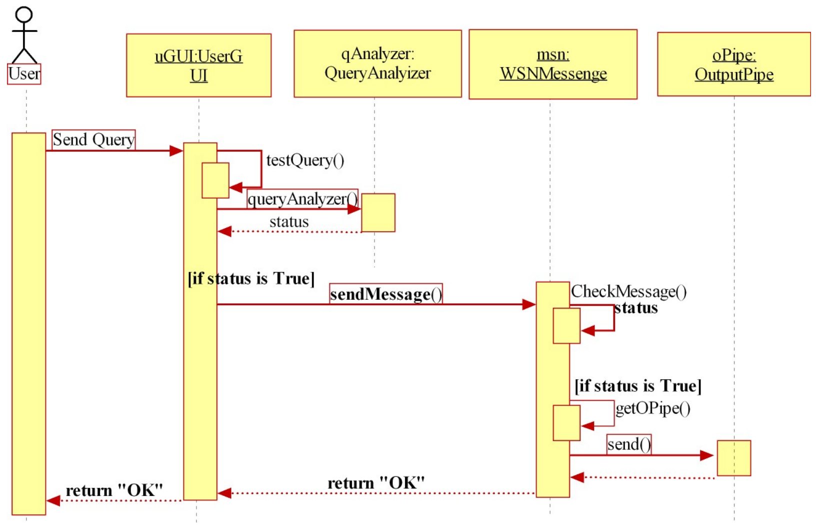


Figure 5.9.: **Sequence Diagram for Sending Query.**

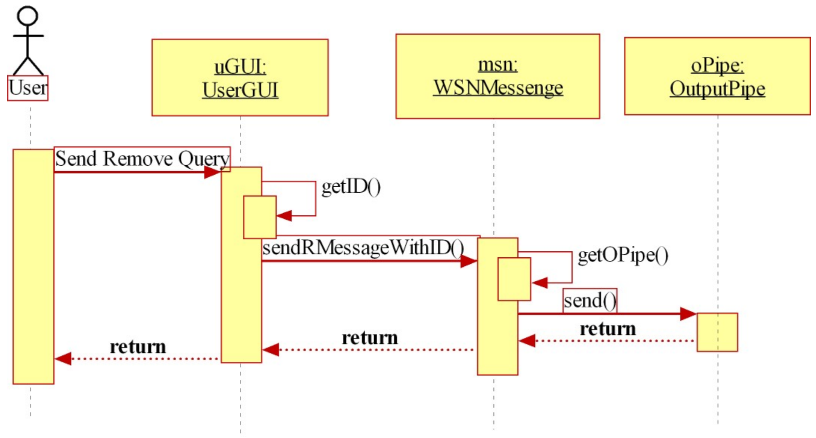


Figure 5.10.: **Sequence Diagram for Ceaseing Query Processing.**

Handling Received Message Process. Table 5.1 shows that a user agents can receive three different messages from a sensor gateway through an **InputPipe** class. In Figure 5.11, all these messages are tested against their types and handled with different methods.

Disconnect Process. Connection is closed only when the user want to disconnect from the system, which means that user can't send further queries or receive results. The sequence diagram of disconnect process is shown in Figure 5.12. This process ceases all query's processing by sending *SendRemoveMessage()* to all gateways. This message contains REMOVE_FILTER_BY_USERID as a message type. Then, a *logout()* is called to close the inputPipe and the message listener. Finally, the user logs out from the system.

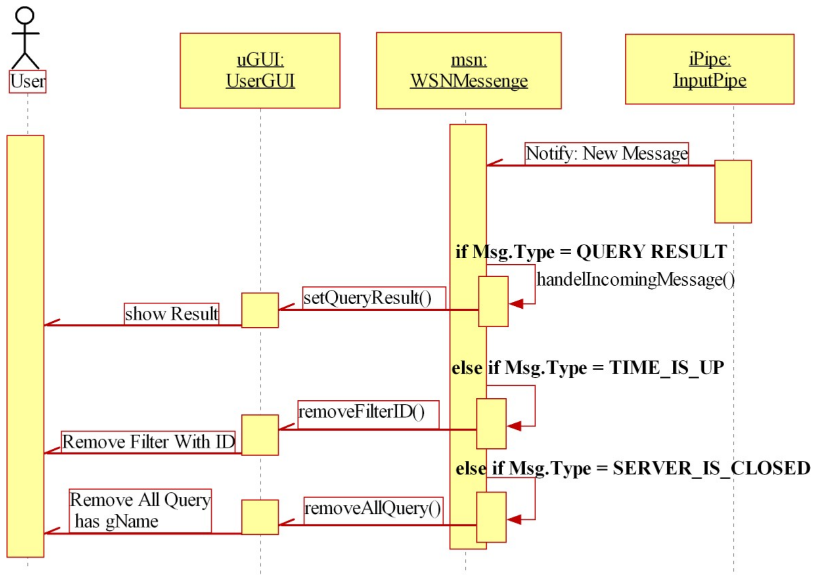


Figure 5.11.: Sequence Diagram for Handling Received Messages.

5.2.4. User Agent Interface (GUI)

A few screen shots of the running system are shown in Figure 5.13. The main panel on the left side allows a user to connect/disconnect, find services, send query. The lower right corner presents the result of discovering process, which aggregates all gateways and their advertisements in a tree form as shown in the left figure. A User writes his query in the query text area. Any query, which is successfully sent, are shown in the upper right corner as filter with a unique ID. In the middle right panel, the results of query processing are displayed.

5.3. Summary

This chapter has introduced the implementation part of JMSN. While sensor gateway has been developed in C, the user agent has been developed in Java. Some implementation issues were discussed. In the next chapter, an evaluation of JMSN is presented to give an impression on overall performance.

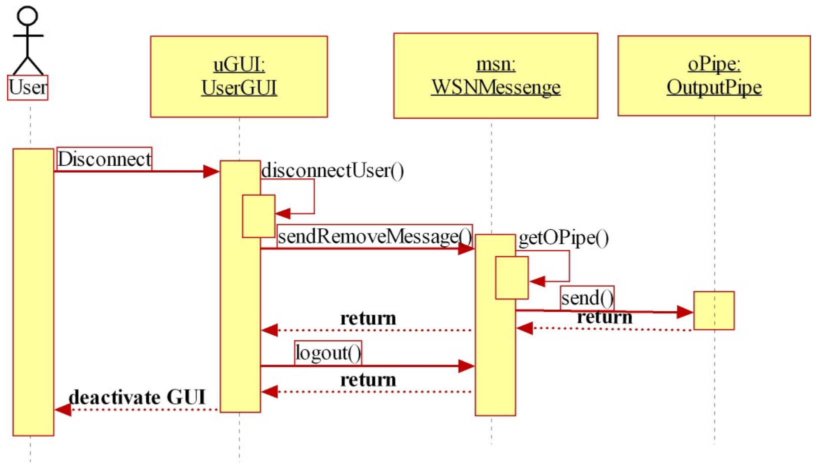
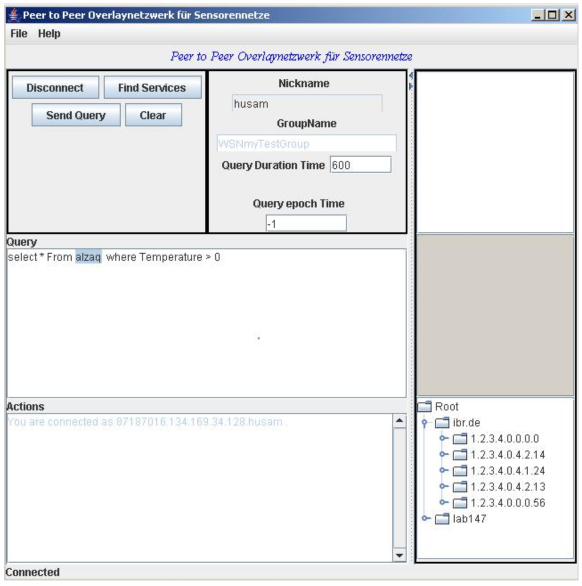


Figure 5.12.: Sequence Diagram for Disconnecting.



Nickname: husam
GroupName: WSNmyTestGroup
Query Duration Time: 600
Query epoch Time: -1

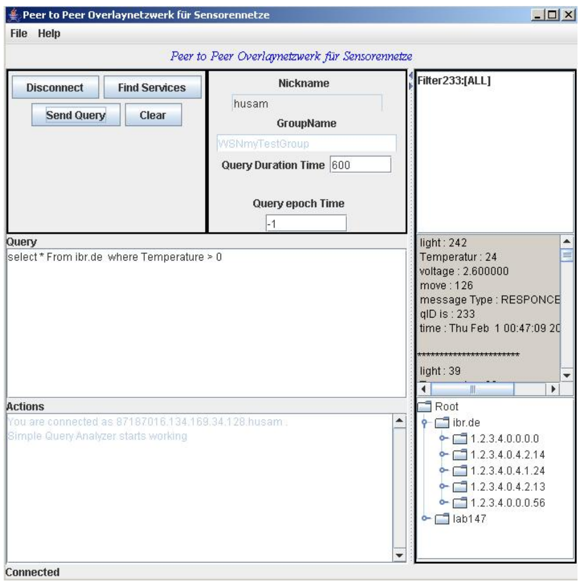
Query

```
select * From alzaq  where Temperature > 0
```

Actions

```
You are connected as 87187016.134.169.34.128.husam .
```

Connected



Nickname: husam
GroupName: WSNmyTestGroup
Query Duration Time: 600
Query epoch Time: -1

Query

```
select * From ibr.de  where Temperature > 0
```

```
light : 242
Temperatur : 24
voltage : 2.600000
move : 126
message Type : RESPONCE
qID is : 233
time : Thu Feb  1 00:47:09 20

**********************
light : 39
```

Actions

```
You are connected as 87187016.134.169.34.128.husam .
Simple Query Analyzer starts working
```

Connected

(a) Discovering Node

(b) Presenting the result of Query Processing

Figure 5.13.: JMSN: Screen Shots.

Chapter 6

Results

In this chapter, we present the evaluation and the preliminary empirical results for the performance of JMSN. We have performed experiments to evaluate the system on normal PCs as well as embedded devices by carrying on a set of experimental tests. They have been devoted to determine the reliability degree of the system in terms of continuous run to investigate stability against memory leakages, to examine Jxta pipes and messages overhead and to identify discovery time for sensor resources.

The first section presents the results of functionality and implementation tests. The second section describes test specifications that are performed on PCs. These tests are, effect of discovery process (Section 6.2.2), effect of message size and overhead of filtering process on querying and filtering process (Section 6.2.3 and Section 6.2.4) and the impact of memory overhead on the system performance (Section 6.2.5). Section 6.3 explains the behavior of JMSN when it run on embedded devices.

6.1. Functionality and implementation testing

The functional testing should verify the application design against the functional requirements that are specified in Chapter 4 and validate the functional design against the user's need [67]. It should detect possible internal incompatibility and errors in the application design. As a consequence, the design can be improved. On the other hand, implementation testing can be performed by comparing the implementation with the application design, and by testing the non-functional requirements, so the implementation can be further improved [70].

6.1.1. Test specification and results

A test plan for the main functionality of JMSN was performed with test cases and the desired results. This test is common for both sensor gateway and user agent because of the integration functionality between them. For instance, if the user issues a query, the sensor gateway will filter the data and return the results. Another example is publishing and discovering advertisements.

The testing was divided in five parts according to the main functionality of the JMSN.
1. Login.
2. Join.
3. Publish and Discover Advertisements.
4. Process queries.
5. Handle messages.

From user agent interface, the following test cases that are indicated in Table 6.1 are performed.

No.	Test Case	Desired result	Result
Test 1.1	Log in	Connect to the system and test input pipe creation	OK
Test 2.1	Join predefine group	Search for and join WSNGroup	OK
Test 3.1	Discovering WSNAdvertisements	Search for any WSN Advertisements in WSNGroup	OK
Test 3.2	Presenting Advertisement	View all WSN Advertisements in a tree grouped by gateway name	OK
Test 4.1	Issuing Query without where clause	Test filter operation with type EQ_ANY	OK
Test 4.2	Issuing Query with where clause	Test filter operation with all filter operation	OK
Test 4.3	Presenting Query Result	View Query Results	OK
Test 5.1	Remove Query By ID	Stop query processing	Not OK
Test 5.2	Remove Query By user Name	Disconnect from the system and notify all gateways	OK

Table 6.1.: Test specifications and results for main functionality of JMSN.

Only Test 5.1 was not passed since the user agent hangs after issuing remove message.

User agent suspends and can't give any response. This is an implementation problem due to failure in creating OutputPipe.

6.2. Sensor Gateway: Testing on PCs

6.2.1. Test-bed Equipment

The test has been performed at the Workstation pool of IBR, where twelve machines of identical hardware are used. They are equipped with Intel Pentium(r) D CPU 3.2GHz, 2.00 GB of RAM and fast Ethernet card. Half of them run Windows XP and the other run Debian GNU/Linux distribution.

6.2.2. WSN Advertisements Discovery Performance

JMSN directs its sensor advertisements to Rendezvous (RDV) peers, which form the Jxta overlay network. User should discover these advertisements in a reasonable time. Average discovery time measures the time that is required by the user agent to discover the gateways sensor advertisement.

The test setup was made up of one sensor gateway and several user agents, which were running on Windows and Linux environment. Both the sensor gateway and user agents have connected to random number of rendezvous peers. The tests have been performed 75 times and the results were averaged.

To draw a comparison, Figure 6.1 shows the resource discovery time in JMSN and Intentional Naming System (INS) (Section 3.2.2.3) as a function of n, the number of rendezvous peers or Intentional Naming Resolver (INR) in the second case. The discovery time is given in seconds.

While the impact of increasing number of nodes is slightly small on the case of INS, which is less than 100 ms, the average discovery time is increased in JMSN. By maintaining one rendezvous peer, the average discovery time is less than 500 ms and it will be doubled four times in the case of ten connected rendezvous peers within the overlay network.

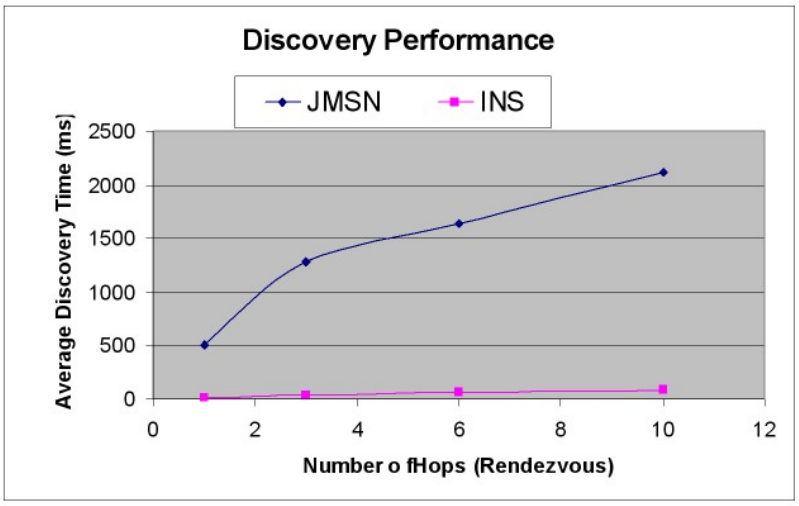


Figure 6.1.: **Average Discovery Time of Finding Network Recourses.**

6.2.3. Messages and Pipes Overhead

Distributed Query Processing depends on the performance of filter results that is composed of a fixed size message (approximately 100 Bytes). Since the most important way to directly exchange data between sensor gateway and user agents is Jxta Pipe, it is significant to study their performance. The basic metrics used to evaluate them are message Round-Trip Time (RTT) [71]. Parsing time is also another important factor that reflects the overhead of parsing user messages.

The test was initiated by a sensor gateway (running on Debian platform), which sent a test message to each connected user, then they returned the same message. The tests were repeated 100 times with different message size, in order to identify the average RTT, (application) throughput and the average parsing time. RTT is a measure of the time it takes for a fixed size message to travel from the gateway to the user agent and return it back again to the gateway. Throughput is the number of bits transmitted in a given time interval through a system. The following equation illustrates how to calculate the (application) throughput in terms of average RTT.

$$Throughput = \frac{2 * Message\_Size * 8}{RTT}$$

Where Message_Size is in bytes and RTT in seconds. The previous equation was multiplied by factor of two because RTT is two way time and it shows that the throughput inversely proportional to RTT. Parsing time is defined as the time requires to parse all messages elements.

Figure 6.2 depicts the impact of message size on the RTT in two scenarios, in which RTT is given in milliseconds. It shows that the RTT is partially changed by increasing

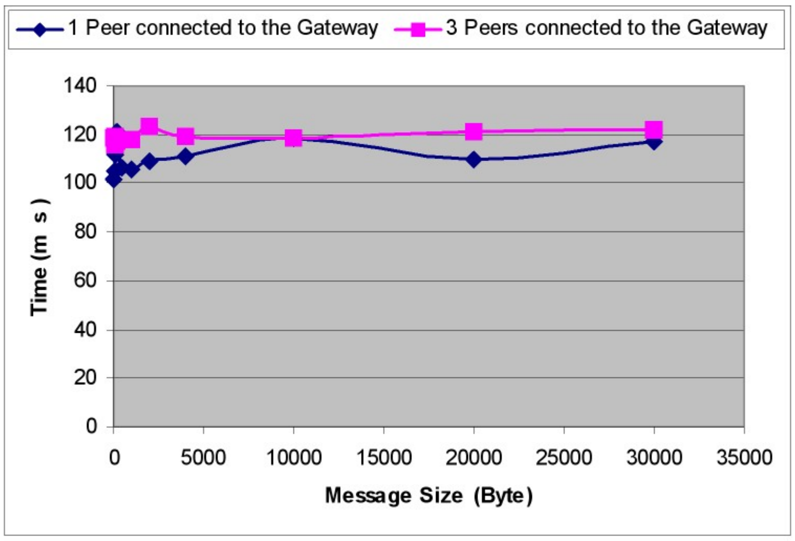


Figure 6.2.: RTT VS Message Size.

the message size. It also illustrates the increasing number of users will add small overhead on the average RTT. Since throughput is inversely proportional to RTT for constant message size, the throughput grows with increasing the message size.

Each time the gateway receives a message, JMSN parses it to exclude the basic elements of it. Figure 6.3 depicts the effect of message size on parsing time with different connected users.

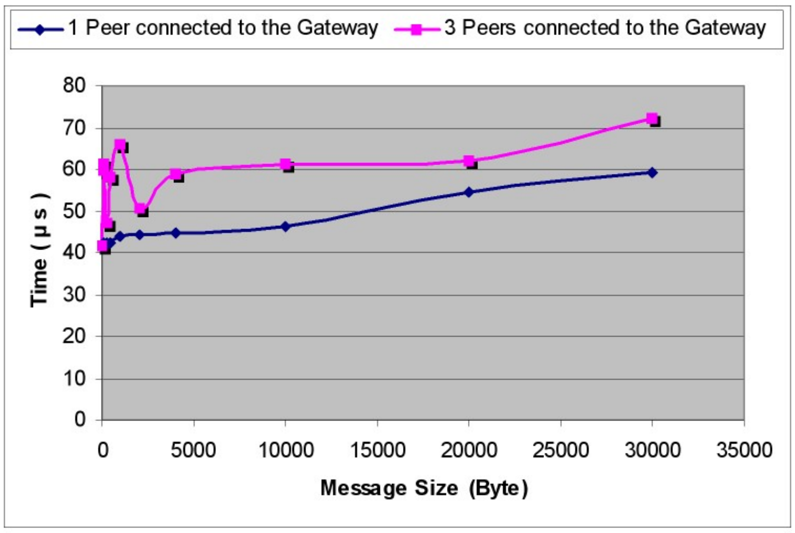


Figure 6.3.: Message Parsing Time.

In all shown scenarios, the parsing time slightly increases with the growing message size. It is ranged from 42 μs to less than 73 μs. With three connected users, the parsing time is partially changed.

6.2.4. The Overall Filtering Overhead

We are interested in the overall filtering time because it gives an indication on the performance of the system with increasing number of filters. For that purpose, a primary test is performed, in which one user agent sent out a query message to the sensor gateway. Then the time that was needed to process all filters per tuple was recorded. The test was performed with 1,3,6 and 10 filters for one user and the average time of 100 operations was plotted.

With constant message size (approximately 100 Bytes) Figure 6.4 illustrates the overhead of filter operation and sending message. It shows that the time is indeed linear in n with a slope of 210.4 ms/filter operation.

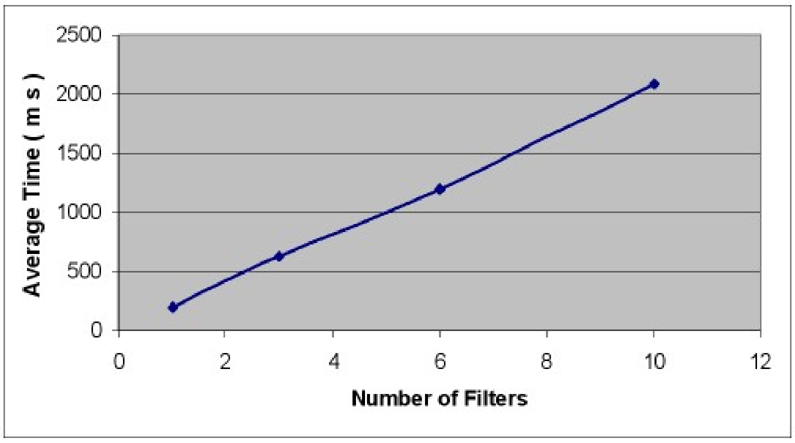


Figure 6.4.: Filter Overhead on JMSN.

The overhead of each filter process includes the time to process the filter, searching for user PipeAdvertisement, establishing a connection and finally sending the message. It appears that searching for user PipeAdvertisement is the most costly operation because a search query for the user will be sent out to the rendezvous peers in the case where no such one was found in the local database of the gateway.

6.2.5. Memory Overhead

In Table 6.2 a summary of the tested conditions and the achieved results is shown for two cases with different stack size. The results are in megabyte. The first part

Running Scenario	approx. memory Stack size 8 MB	usage (MB) Stack size 2 MB
Gateway after discovering RDV	54,4	18,4
Gateway with 3 discovers sensors	62,8	22,7
Gateway with 1 filter	70,8	23,7
Gateway with 3 filters	79,0	29,8
Gateway with 5 filters	95,3	30,2
Gateway with 10 filters	137,6	32,5
Gateway with 15 filters (from 10 connected users)	139	32,7

Table 6.2.: Summary of memory consumption tests.

of Table 6.2 shows that the memory overhead of running JMSN over Jxta-C after bootstrapping and creating an input pipe is relatively high in both cases. The second part reveals that with the increasing number of connected users and filters, the consumed memory size is also increased. Because of large stack size (8 MB), the consumed memory with 10 connections reaches approximately 139 MB, whereas it is approximately 33 MB for 2 MB stack size.

This memory footprint is mainly due to the Apache Runtime (APR), which is embedded in the Jxta-C implementation in responsible for creating and managing threads.

6.3. Sensor Gateway: Testing on Embedded System

The sensor gateway application was also tested on both ASUS WL-500G Deluxe V and WL-500gP. Both of them are equipped with OpenWRT Whiterussian RC5 [72] and WBridge, a Linux distribution for embedded devices. Appendix B shows how to install JMSN on Asus router.

The test has started well; but after a while an unexpected failure has occurred. The problem is located as a segmentation fault breaks down the application. All Jxta-C tests were also checked out but they were failed with same problem. The reasons behind this are unknown. The same problem also appeared in the gateway that is running on Debian when the internal database is locked. In other words, if the gateway fails to create this database or access it, the application will fail. However, this problem was probably due to the memory overhead, as we identified with the help of *ddd* debugger, eight threads are usually created during the bootstrapping phase.

6.4. summary

The first part of this chapter has presented the test specifications and the test results to check out the functionality and implementation of the current implementation of JMSN. The second parts has described test specifications to test the performance of JMSN.

We observed that the discovery time in JMSN was small, although it is higher than INS due to a fixed wait state required by JMAN. Communication over LAN, different platforms (Java and C) and different environments (windows and Debian), all introduce an overhead that had an influence on RTT. Moreover, this also had an effect on the filtering process, which is also affected by discovering pipe advertisements. The large impact on JMSN is due to memory consumption.

Chapter 7

Summary and Future Work

In this thesis we have presented the JMSN architecture for discovering sensor gateway and executing a declarative queries over sensor networks. Only a small subset of sensor nodes need to participate in answering the user query without flooding the query. Many applications in the field of ubiquitous and pervasive computing can take the advantage of JMSN specially control and observing systems.

7.1. Conclusions

This thesis studied how service discovery could be implemented in favor of sensor network based on Peer to Peer paradigm. Peer to Peer Systems provide a decentralized approach that does not require any costly infrastructure. Jxta was selected among other systems since it is a generic overlay network that connects devices on the network. Jxta is massively scalable since it employs the super-peer concept. Jxta implements a loosely-coupled consistent DHT rendezvous walker approach to overcome inconsistency problem of the DHT within the dynamic rendezvous network in order to efficiently discover service.

Our preliminary implementation of the specialization techniques illustrated that JMSN combines a peer service discovery with a distributed query processing. The current implementation enables users to discover WSNAdvertisements that carry information about a sensor network to allow them to forward their query to the right sensor gateway over Jxta Point-to-Point Pipe.

JMSN is similar to database approaches; however, the main difference between JMSN and these approaches is that query processing is neither accomplished by sensor nodes nor programmed in sensor nodes. Processing is divided between users and a gateway. The gateway is responsible to filter sensor data that sensors regularly send them. Results will be sent out to user if they meet his condition. Then, user completes the processing by presenting or storing the results based on query attributes.

JMSN design does not come without cost. Our preliminary results showed that the current implementation of Jxta-C consumes memory, which is not efficient for embedded system with limited memory. It also showed that use of Jxta Point-to-Point Pipes is inefficient for carrying messages. This means, Jxta-C is not bad, rather its urgent needs to improve its memory pool.

7.2. Future Works

A number of issues have not been discussed further in this thesis. This work opens a plethora of new research directions at the boundary of Peer to Peer Systems, database systems and wireless network. First, we would like to improve the functionality of JMSN. It is necessary to enhance the system with complex query that can benefit from Peer to Peer System in searching. This includes searching by partial names and attributes such as Location. Aggregate function such as Average, Max, Count ...etc., should be implemented in order to cope with all statistical operations.

Second, the simple design of our filter algorithm can be improved in three directions. In the first one, similar sensor data should not be sent out. They can be aggregated and compressed before returning them to the user. In the second direction filter processing can be optimized by gathering similar filter and by using anticipated methods that are used for example in the artificial intelligent. Instead of using TCP based pipe, UDP based pipe (propagate pipe) can be used as a third way to improve the overall system performance.

Third, as a Jxta Micro Peer, we would like to extend the current version of Jxta-C implementation to cope with the limited memory devices. The basic functionality of Jxta with the fundamental services are only required. This option is not an alternative solution, but it is an option for these kinds of devices.

Summing up, this work provides a solution to existing problems motivated by using the Peer to Peer paradigm in ubiquitous and pervasive computing systems. Our results have presented that current implementation of JMSN over Jxta-C wouldn't be accepted as a middleware for embedded system without improving it.

Bibliography

[1] Dejan S Milojicic and Vana Kalogeraki and Rajan Lukose and Kiran Nagaraja and Jim Pruyne and Bruno Richard and Sami Rollins and Zhichen Xu . Peer-to-Peer Computing. HP Laboratories Palo Alto; 2002.

[2] Oram A. Peer-To-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly; 2001.

[3] IPOQUE;. Available from: `http://www.ipoque.com/en/p2p_filter.html`

[4] BitTorrent;. Available from: `http://www.bittorrent.com/`

[5] G3 Torrent;. Available from: `http://g3torrent.sourceforge.net/`

[6] Shareaza;. Available from: `http://www.shareaza.com/`

[7] Kulbak Y, Bickson D. The emule protocol specification; 2005. Available from: `citeseer.ist.psu.edu/kulbak05emule.html`

[8] Hughes D, Coulson G, Walkerdine J. Free Riding on Gnutella Revisited: The Bell Tolls? IEEE Distributed Systems Online 2005;6(6):1.

[9] Hauswirth M, Dustdar S. Peer-to-Peer: Grundlagen und Architektur. Datenbank-Spektrum 2005;5(13):5–13.

[10] Cohen E, Shenker S. Replication strategies in unstructured peer-to-peer networks; 2002. In The ACM SIGCOMM'02 Conference, August 2002. Available from: `citeseer.ist.psu.edu/cohen02replication.html`

[11] Bo CW. Peer-to-Peer Overlay Networks: A Survey; 2003. Available from: `citeseer.ist.psu.edu/706822.html`

[12] Stoica I, Morris R, Karger D, Kaashoek MF, Balakrishnan H. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: Proceedings of the ACM SIGCOMM '01 Conference. San Diego, California; 2001. 149–160.

[13] Ratnasamy S, Francis P, Handley M, Karp R, Shenker S. A Scalable Content Addressable Network. In: Proceedings of ACM SIGCOMM; 2001. .

[14] Maymounkov P, Mazieres D. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Proc. 1st International Workshop on Peer-to-Peer Systems; 2002. `http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf`.

[15] Freenet Website.;. Available from: `http://freenetproject.org/`

[16] Napster;. Available from: `http://www.napster.com/`

[17] Softwax;. Available from: `http://www.softwax.com/`

[18] KaZaa Website.;. Available from: `http://www.kazaa.com/us/index.htm`

[19] JXTA;. Available from: `http://www.jxta.org/`

[20] Peer-to-Peer: Content Distribution.;. Available from: `http://www.intel.com/it/pdf/peer-peer-content-distribution.pdf`

[21] Tsoumakos D, Roussopoulos N. Analysis and comparison of P2P search methods. In: InfoScale '06: Proceedings of the 1st international conference on Scalable information systems. New York, NY, USA: ACM Press; 2006. P. 25.

[22] Tang C, Xu Z, Dwarkadas S. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. New York, NY, USA: ACM Press; 2003. 175–186.

[23] Traversat B, Abdelaziz M, Pouyoul E. A Loosely-Consistent DHT Rendezvous Walker. Sun Microsystems, Inc; 2003.

[24] Barkai D. A New Computing Model for the Internet.; 2005. Available from: `http://www.intel.com/intelpress/chapter-peer.pdf`

[25] BOINCstats - BOINC Statistics.;. Available from: `http://www.boincstats.com/stats/project_graph.php?pr=sah`

[26] Skype;. Available from: `http://www.skype.com/`

[27] Mauthe A, Hutchison D. Peer-to-Peer Computing: Systems, Concepts and Characteristics. Praxis in der Informationsverarbeitung & Kommunikation (PIK), K G Sauer Verlag, Special Issue on Peer-to-Peer 2003;26(03/03).

[28] Steinmetz R, Wehrle K. Peer-to-Peer Systems and Applications. Lecture Notes in Computer Science. Springer; 2005.

[29] Ledlie J, Taylor J, Serban L, Seltzer M. Self-organization in peer-to-peer systems; 2002. Available from: `citeseer.ist.psu.edu/article/ledlie02selforganization.html`

[30] SETI@HOME;. Available from: `http://setiathome.ssl.berkeley.edu/`

[31] Groove Virtual Office;. Available from: `http://www.groove.net/home/index.cfm`

[32] Microsystems S. JXTA v2.3.x: Java Programmer's Guide. Sun Microsystems, Inc; 2005.

[33] Brookshier D, Govoni D, Krishnan N, Soto JC. Enterprise Modeling and Computing with UML. Sams Publishing; 2002.

[34] Chord;. Available from: `http://pdos.csail.mit.edu/chord/`

[35] Subramanian R, Goodman B. Idea Group Publisher, Hershey, PA, USA; 2005.

[36] Helal S. Standards for Service Discovery and Delivery. IEEE Pervasive Computing 2002;1(3):95–100.

[37] Feng J, Koushanfar F, Potkonjak M. System Architecture for Sensor Network Issues, Alternatives, and Directions. In: IEEE International Conference on Computer Design: VLSI in Computers and Processors(ICDD'02); 2002. .

[38] Tilak S, Abu-Ghazaleh NB, Heinzelman W. A taxonomy of wireless micro-sensor network models. SIGMOBILE Mob Comput Commun Rev 2002;6(2):28–36.

[39] Römer K, Kasten O, Mattern F. Middleware challenges for wireless sensor networks. SIGMOBILE Mob Comput Commun Rev 2002;6(4):59–61.

[40] Römer K. Programming Paradigms and Middleware for Sensor Networks. GI/ITG Fachgespräch Sensornetze, Karlsruhe 2004;.

[41] Shenker S, Ratnasamy S, Karp B, Govindan R, Estrin D. Data-centric storage in sensornets. ACM SIGCOMM Computer Communication Review 2003;33(1): 137–142.

[42] Heidemann J, Silva F, Estrin D. Matching Data Dissemination Algorithms to Application Requirements. In: Proc. 1st Intl. Conf. on Embedded Networked Sensor Systems (SenSys). Los Angeles, CA: ACM; 2003. 218–230.

[43] Madden S, Franklin MJ, Hellerstein JM, Hong W. TAG: A Tiny Aggregation Service for ad hoc Sensor Networks. In: OSDI; 2002. .

[44] Yu Y, Govindan R, Estrin D. Geographical and Energy Aware Routing: A recursive data dissemination protocol for wireless sensor networks. University of California at Los Angeles; 2001.

[45] Zhao F, Guibas L. Wireless Sensor Networks – An Information Processing Approach. Amsterdam: Elsevier / Morgan-Kaufman; 2004.

[46] Yao Y, Gehrke J. The cougar approach to in-network query processing in sensor networks. SIGMOD Rec 2002;31(3):9–18.

[47] TinyDB;. Available from: `http://telegraph.cs.berkeley.edu/tinydb`

[48] Hill J, Culler D. A wireless embedded sensor architecture for system-level optimization; 2001. Available from: `citeseer.ist.psu.edu/hill01wireless.html`

[49] Yao Y, Gehrke J. Query Processing for Sensor Networks. In: First Biennial Conference on Innovative Data Systems Research(CIDR 2003); 2003. .

[50] Gehrke J, Madden S. Query Processing in Sensor Networks. IEEE PERVASIVE Computing 2004;.

[51] C Intanagonwiwat RG, Estrin D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: Proceedings of the sixth annual international conference on Mobile computing and networking. Boston, MA USA; 2000. 56–67.

[52] Sun Microsystems. Jini Community Resources: Jini Technology Architectural Overview; 1999.

[53] Sundramoorthy V, Hartel PH, Scholten J. On Consistency Maintenance In Service Discovery. In: 20th IEEE Int. Parallel & Distributed Processing Symp. (IPDPS). Rhodos Island, Greece: IEEE Computer Society Press, Los Alamitos, California; 2006. .

[54] Dabrowski C, Mills K. Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach; 2001. Available from: `citeseer.ist.psu.edu/dabrowski01analyzing.html`

[55] Bettstetter C, Renner C. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. In: EUNICE 2000, Sixth EUNICE Open European Summer School. Twente, Netherlands; 2000. .

[56] Lee C, Helal S. Protocols for Service Discovery in Dynamic and Mobile Networks. In: International Journal of Computer Research.; 2002. .

[57] Service Discovery in the Future for Mobile Commerce; 2003. Available from: `http://www.acm.org/crossroads/xrds7-2/service.html`

[58] Guttman E, Perkins C, Veizades J, Day M. Service Location Protocol, Version 2. RFC 2608 (Proposed Standard); 1999. Updated by RFC 3224. Available from: `http://www.ietf.org/rfc/rfc2608.txt`

[59] Guttman E. Current Solutions for Web Service Composition. IEEE Internet Computing 1999;3(4):71–80.

[60] Adjie-Winoto W, Schwartz E, Balakrishnan H, Lilley J. The design and implementation of an intentional naming system. In: SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles. New York, NY, USA: ACM Press; 1999. 186–201.

[61] Balazinska M, Balakrishnan H, Karger D. INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery. In: International Conference on Pervasive Computing. Zurich, Switzerland; 2002. .

[62] Twine: Scalable Intentional Resource Discovery for Pervasive Computing Environments; 2003. Available from: `http://nms.csail.mit.edu/projects/twine/`

[63] Marin-Perianu R, Scholten J, Havinga PJM. CODE: Description Language for Wireless Collaborating Objects. Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands; 2005.

[64] Microsoft Co. Understanding Universal Plug and Play: White Paper; 2000.

[65] Frank C, Handziski V, Karl H. Service Discovery in Wireless Sensor Networks; 2004.

[66] Teco, Particle Wib Site;. Available from: `http://particle.teco.edu/`

[67] Baekgaard L. Enterprise Modeling and Computing with UML. Idea Group; 2006. In press.

[68] Ahmed R, Boutaba R, Cuervo F, Iraqi Y,Li T, Limam N, Xiao J,Ziembicki J. Service Discovery Protocols: A Comparative Study. In: Proceeding of the IFIP/IEEE International Symposium on Integrated Network Management (IM'2005); 2005. To appear.

[69] Silva F, Heidemann J, Govindan R. Network Routing Application Programmer's Interface; 2002. Available from: `citeseer.ist.psu.edu/silva02network.html`

[70] Lyngstad BPS. Network technologies for Java-enabled, mobile devices. Master thesis; 2001.

[71] Halepovic E, Deters R. The Costs of Using JXTA. In: P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing. Washington, DC, USA: IEEE Computer Society; 2003. P. 160.

[72] OpenWRT;. Available from: `http://www.openwrt.org`

Appendix A

BNF for JMSN Query

Input ::= select_statement

select_statement ::= select_expression

select_expression ::= <SELECT> <ALL> | select_list <FROM> from_location_list (<WHERE> cond_exp)? <DURATION> time_interval

time_interval ::= <INTEGER_LITERAL>

from_location_list ::= <IDENTIFIER>

select_list ::= attribute (<COMMA> attribute)*

attribute ::= <IDENTIFIER>

cond_exp ::= cond_factor

cond_factor ::= (<NOT>)? cond_test

cond_test ::= cond_primary

cond_primary ::= simple_cond

simple_cond ::= between_cond | comparison_cond

comparison_cond ::= row_constructor comparison_operator row_constructor

between_cond ::= row_constructor (<NOT>)? <BETWEEN> row_constructor

<AND> row_constructor | row_constructor (<NOT>)? <BETWEENX> row_constructor <AND> row_constructor

row_constructor ::= expression

comparison_operator ::= <LESSEQUAL> | <GREATEREQUAL> | <NOTEQUAL> | <EQUAL> | <LESS> | <GREATER>

expression ::= <INTEGER_LITERAL> | <FLOATING_POINT_LITERAL>

Appendix B

Installing Jxta on Asus Router

To install JMSN on Asus Router that is equipped with OpenWRT (RC 5) the following standard libraries are necessary,

1. libopenssl: (ipkg install libopenssl_0.9.8d-1_mipsel.ipk)

2. libsqlite3: (ipkg install libsqlite_3.3.8-1_mipsel.ipk)

3. libexpat: (ipkg install libexpat_2.0.0-1_mipsel.ipk)

4. libpthread: (ipkg install libpthread_0.9.27-1_mipsel.ipk)

5. zlib: (ipkg install zlib_1.2.3-3_mipsel.ipk)

Besides the previous libraries, the following libraries are required,

1. libxml2: (ipkg install libxml2_2.6.20-1_mipsel.ipk)

2. libapr: (ipkg install libapr_1.2.7-1_mipsel.ipk)

3. apr-util: (ipkg install apr-util_1.2.7-1_mipsel.ipk)

Now it is possible to install JMSN. This package, jmsn_1.1-beta-1_mipsel.ipk, includes Jxta libraries as well as JMSN
(ipkg install jmsn_1.1-beta-1_mipsel.ipk)

It includes all test files that are shipped with jxta-c besides our prototype JMSN, the sensor Application. There are also four tutorials can be tested to make sure that Jxta-C works probably.

To run the application just type sensorApplication (or sensorApplication [name AsusUser-Name]) If the application success, it will create a file called Platformconfig and a hidden folder named as .cm to store a SQLite database.

Notes

- To ensure the router exposes the sensor network, use sensorManager as a test tool. You can use the following package sensorTest_1-1_mipsel.ipk (ipkg install sensorTest_1-1_mipsel.ipk) to install it. It lists are sensors in the environments.

- Asus WL-500g Deluxe has approximately 1.6M of 3.2M of file system available and an external USB stick is required. Asus WL-500gP has enough file system, more than 5MB of 7.2 MB is available.