P-3294

QOS AWARE SERVICE SELECTION IN SERVICE ORIENTED WIRELESS SENSOR NETWORKS

By

T. Subalaxmi

Reg. No: 0820108020

of

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna University, Coimbatore)

COIMBATORE – 641 006

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In partial fulfillment of the requirements for the award of the degree of

MASTER OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

MAY 2010

BONAFIDE CERTIFICATE

Certified that this project report titled "**Qos Aware Service Selection in Service Oriented Wireless Sensor Networks**" is the bonafide work of **Ms.T.SUBALAXMI (0820108020)** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

GUIDE

(Dr. L. S. JAYASHREE)

HEAD OF THE DEPARTMENT

(Ms. P. DEVAKI, M.E.)

The candidate with **University Register No. 0820108020** was examined by us in Project Viva-Voce examination held on 18/5/10

Internal Examiner

External Examiner

VELALAR COLLEGE OF ENGINEERING AND TECHNOLOGY

THINDAL, ERODE - 12.

NATIONAL CONFERENCE ON RECENT TRENDS IN INNOVATIVE TECHNOLOGIES

NCRTIT '10

Organized by ***Department of Computer Science and Engineering***

CERTIFICATE

This is to certify that T. SUBALAXMI, PG STUDENT **of** KUMARAGURU COLLEGE OF TECH, COIMBATORE **has presented a paper titled** SERVICE COMPOSITION IN SERVICE ORIENTED WIRELESS SENSOR NETWORKS **in the NATIONAL CONFERENCE ON RECENT TRENDS IN INNOVATIVE TECHNOLOGIES (NCRTIT '10) held on March 27, 2010.**

Co-ordinator

Convener

Principal i/c

Administrative Director

ABSTRACT

Wireless Sensor Networks (WSNs) present a challenging programming environment because of their limited resources, heterogeneity, and highly dynamic nature. Service Oriented Architecture has become a promising paradigm for application development in wireless sensor networks due to its benefits in cost, efficiency, agility, and adaptability. One of the main issues in service oriented WSN is to select appropriate component service for a service composition so as to fulfill the user request. For services providing similar functionality, Quality of Service (QoS) is the main factor to differentiate them. This leads to general optimization problem of how to select services for each task so that the overall QoS requirements of the composition are satisfied and optimal. This paper presents a method for selecting best service among a bunch of services with similar functions but with different QoS. Based on global QoS constraints set by users the service selection method selects appropriate service from each service group to form composite service.

ஆய்வுச்சுருக்கம்

கம்பியில்லா உணர்வி இனணயமானது பல்வேறு வகையான உணர்வி முடிச்சுகளையும் வரையறுக்கப்பட்ட ஆற்றலையும், அதிகபட்ச மாறும் தன்மையையும் கொண்டது. இச்சூழலில் பயன்பாட்டு நிரலாக்கம் என்பது மிகவும் சவாலான ஒன்றாகும். இதற்கான தீர்வு சேவை அடிப்படையிலான கட்டமைப்பாகும். சேவை அடிப்படையிலான கட்டமைப்பை சார்ந்த கம்பியில்லா உணர்வி இனணயத்தில் பயனாளர்களின் தேவையை பூர்த்தி செய்ய ஒன்றுக்கு மேற்பட்ட சேவையாளர்கள் உள்ளனர். இச்சேவையாளர்கள் செயல்திறன் அடிப்படையில் ஒன்றுபட்டும் தரத்தின் அடிப்படையில் வேறுபட்டும் உள்ளனர். மேலும் பயனாளர்களின் தேவையை பூர்த்தி செய்ய ஒன்றுக்கு மேற்பட்ட சேவை அவசியம். எனவே ஒன்றுக்கு மேற்பட்ட சேவையாளர்கள் ஒருங்கிணைக்கபட்டு பயனாளரின் தேவை பூர்த்தி செய்யபடுகிறது.

சேவை அடிப்படையிலான கட்டமைப்பை சார்ந்த கம்பியில்லா உணர்வி இனணயத்தில் மிக முக்கியமான பிரச்சனைகளில் ஒன்று சேவையாளர் தேர்வு. சேவையாளர் தேர்வானது பயனாளர்களின் தேவையை பூர்த்தி செய்யும் வகையுலும், ஒருங்கிணைந்த சேவை தரத்தை மேம்படுத்தும் வகையுலும் அமைய வேண்டும். இச்செயல் திட்டத்தில் சேவையாளர்களை தேர்வு செய்ய கான்வெக்ஸ் ஹல் முறை பயன்படுத்தப்படுகிறது. இம்முறை ஒன்றுக்கு மேற்பட்ட ஒரே செயல்திறனும் வெவ்வேறானா தரமும் கொண்ட சேவையாளர்களிருந்து ஒரு சேவையாளரை தேர்வு செய்யும் வழிவகையை விளக்குகிறது, மேலும் ஒருங்கிணைந்த சேவை தரத்தையும் மேம்படுத்துகிறது.

ACKNOWLEDGEMENT

First and foremost, I would like to thank the almighty for enabling me to complete this project.

I whole-heartedly wish to express my special thanks to our Chairman, Management, and Director for giving this great opportunity to pursue this course. I thank **Dr. S. Ramachandran,** Principal, for providing me with the necessary facilities and Infrastructure to work on this project.

I would like to take this opportunity to thank **Dr.S.Thangasamy,** R&D Dean, for his everlasting support and useful suggestions throughout this project.

I would like to extend my thanks to **Ms. P. Devaki, M.E.,** Head, Department of Computer Science and Engineering for her valuable suggestions.

I express my heartiest thanks to my Project Guide **Dr. L.S.Jayashree,** Professor and Head, Department of Information Technology for her valuable suggestions and support right from the beginning to perform my project work extremely well.

I thank our Project Coordinator **Ms. V.Vanitha M.E**, Assistant Professor, Department of Computer Science and Engineering, for her valuable advice and guidance throughout this project.

I would like to convey my honest thanks to all **Teaching and Non-Teaching** staff members of the department for their support.

I would to extend my heartfelt thanks to all my beloved friends and my family members for their moral support and encouragements.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

WSN	Wireless Sensor Networks
QoS	Quality of Services
SOA	Service Oriented Architecture
MANET	Mobile Ad-hoc Networks

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW OF WIRELESS SENSOR NETWORKS

Wireless Sensor network

Wireless Sensor Networks is a network of typically small, battery-powered, distributed autonomous wireless devices. A sensor node, also known as a 'mote', is a node in a wireless sensor network that is capable of performing some processing, gathering sensory information and communicating with other connected nodes in the network. In addition to one or more sensor nodes, each node in a sensor network is typically equipped with a radio transceiver or other wireless communications device, a small microcontroller, and an energy source, usually a battery [1] .

Components of sensor networks

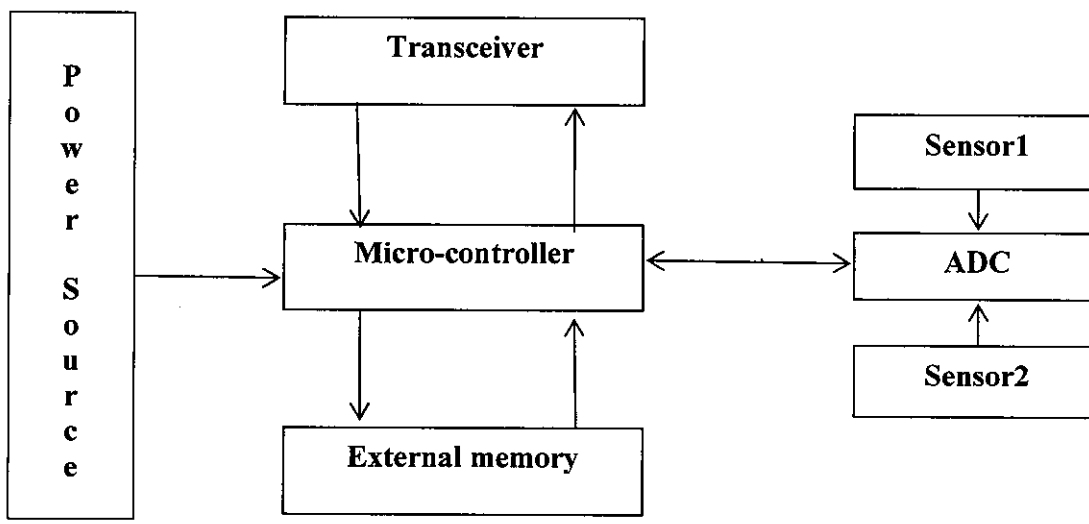


Figure 1.1 Components of Sensor Networks

- **Microcontroller**

Microcontroller performs tasks, processes data and controls the functionality of other components in the sensor node. Other alternatives that can be used as a controller are: General purpose desktop microprocessor, Digital signal processors, Field Programmable Gate Array and Application-specific integrated circuit. Microcontrollers are the most suitable choice for a sensor node.

- **Transceiver**

Sensor nodes make use of ISM band which gives free radio, huge spectrum allocation and global availability. The various choices of wireless transmission media are Radio frequency, Optical communication (Laser) and Infrared. Laser requires less energy, but needs line-of-sight for communication and also sensitive to atmospheric conditions. Infrared like laser, needs no antenna but is limited in its broadcasting capacity. Radio Frequency (RF) based communication is the most relevant that fits to most of the WSN applications. WSN's use the communication frequencies between about 433 MHz and 2.4 GHz. The functionality of both transmitter and receiver are combined into a single device known as transceivers are used in sensor nodes. Transceivers lack unique identifier. The operational states are Transmit, Receive, Idle and Sleep.

- **External Memory**

From an energy perspective, the most relevant kinds of memory are on-chip memory of a microcontroller and Flash memory - off-chip RAM is rarely if ever used. Flash memories are used due to its cost and storage capacity. Memory requirements are very much application dependent. Two categories of memory based on the purpose of storage a) User memory used for storing application related or personal data. b) Program memory used for programming the device. This memory also contains identification data of the device if any.

- **Power Source**

Power consumption in the sensor node is for the Sensing, Communication and Data Processing. More energy is required for data communication in sensor node. Energy expenditure is less for sensing and data processing. The energy cost of transmitting 1 Kb a distance of 100 m is approximately the same as that for the executing 3 million instructions by 100 million instructions per second/W processor. Power is stored either in Batteries or Capacitors. Batteries are the main source of power supply for sensor nodes.

- **Sensors**

Sensors are hardware devices that produce measurable response to a change in a physical condition like temperature and pressure. Sensors sense or measure physical data of the area to be monitored. The continual analog signal sensed by the sensors is digitized by an Analog-to-digital converter and sent to controllers for further processing. Characteristics and requirements of Sensor node should be small size, consume extremely low energy, operate in high volumetric densities, be autonomous and operate unattended, and be adaptive to the environment. As wireless sensor nodes are micro-electronic sensor device, can only be equipped with a limited power source of less than 0.5 Ah and 1.2 V.

- **ADC - Analog to Digital Converter**

The analog signals produced by the sensors are converted to digital signals by the analog to digital converter (ADC), and then fed into the processing unit.

1.2 APPLICATIONS OF SENSOR NETWORKS

Wireless Sensor Networks have a wide range of applications such as,

1. Military applications
    i. Monitoring friendly forces and equipment.
    ii. Battlefield surveillance.
    iii. Nuclear, biological and chemical attack detection.
2. Environmental Applications
    i. Forest fire detection.
    ii. Bio-complexity mapping of the environment.
    iii. Flood detection

3. Health applications

i. Tele-monitoring of human physiological data.

ii. Tracking and monitoring doctors and patients inside a hospital.

iii. Drug administration in hospitals.

4. Home application

i. Home automation.

ii. Smart environment

In order to enable reliable and efficient observation and initiate right actions, physical phenomenon features should be reliably detected/estimated from the collective information provided by sensor nodes. Moreover, instead of sending the raw data to the nodes responsible for the fusion, sensor nodes use their processing abilities to locally carry out simple computations and transmit only the required and partially processed data. Hence, these properties of WSN impose unique challenges for development of communication protocols in such architecture. The intrinsic properties of individual sensor nodes, pose additional challenges to the communication protocols in terms of energy consumption.

1.3 CHALLENGES OF WIRELESS SENSOR NETWORKS

A sensor network design is influenced by many factors, which include,

- **Energy efficiency/system lifetime**

As sensor nodes are battery-operated, protocols must be energy-efficient to maximize system life time. System life time can be measured such as the time until half of the nodes die or by application-directed metrics, such as when the network stops providing the application with the desired information about the phenomena.

- **Fault Tolerance**

Some sensor nodes may fail or be blocked due to lack of power, have physical damage or environmental interference. The failure of sensor nodes should not affect the overall task of the sensor network. This is the reliability or fault tolerance issue. Fault

tolerance is the ability to sustain sensor network functionalities without any interruption due to sensor node failures.

- **Scalability**

The number of sensor nodes deployed in studying a phenomenon may be in the order of hundreds or thousands. Depending on the application, the number may reach an extreme value of millions. The new schemes must be able to work with this number of nodes.

- **Production Costs**

Since the sensor networks consist of a large number of sensor nodes, the cost of a single node is very important to justify the overall cost of the networks. If the cost of the network is more expensive than deploying traditional sensors, then the sensor network is not cost-justified. As a result, the cost of each sensor node has to be kept low

.

- **Environment**

Sensor nodes are densely deployed either very close or directly inside the phenomenon to be observed. Therefore, they usually work unattended in remote geographic areas. They may be working in busy intersections, in the interior of large machinery, at the bottom of an ocean, inside a twister, on the surface of an ocean. They work under high pressure in the bottom of an ocean, in harsh environments such as debris or a battlefield, under extreme heat and cold such as in the nozzle of an aircraft engine or in arctic regions, and in an extremely noisy environment such as under intentional jamming.

- **Hardware Constraints**

A sensor node is made up of four basic components: a sensing unit, a processing unit, a transceiver unit and a power unit. Sensing units are usually composed of two subunits: sensors and analog to digital converters (ADCs). The analog signals produced by the sensors based on the observed phenomenon are converted to digital signals by the ADC, and then fed into the processing unit. The processing unit, which is generally

associated with a small storage unit, manages the procedures that enable the sensor node collaborate with the other nodes to carry out the assigned sensing tasks. A transceiver unit connects the node to the network. One of the most important components of a sensor node is the power unit. Power units may be supported by a power scavenging unit such as solar cells. There are also other subunits, which are application dependent. Most of the sensor network routing techniques and sensing tasks require the knowledge of location with high accuracy.

- **Sensor Network Topology**

Sheer numbers of inaccessible and unattended sensor nodes, which are prone to frequent failures, make topology maintenance a challenging task. Hundreds to several thousands of nodes are deployed throughout the sensor field.

a) Pre-Deployment Phase

Sensor nodes can be either thrown in mass or placed one by one in the sensor field. They can be deployed by dropping from a plane, delivering in an artillery shell, rocket or missile, throwing by a catapult , placing in factory, and placing one by one either by a human or a robot. Although the sheer number of sensors and their unattended deployment usually preclude placing them according to a carefully engineered deployment plan, the schemes for initial deployment must reduce the installation cost, eliminate the need for any pre-organization and preplanning, increase the flexibility of arrangement, and promote self-organization and fault tolerance.

b) Post-Deployment Phase

After deployment, topology changes are due to change in sensor nodes position, reach ability (due to jamming, noise, moving obstacles, etc.), available energy, malfunctioning, and task details. Sensor nodes may be statically deployed. However, device failure is a regular or common event due to energy depletion or destruction. It is also possible to have sensor networks with highly mobile nodes. Besides, sensor nodes and the network experience varying task dynamics, and they may be a target for

deliberate jamming. Therefore, sensor network topologies are prone to frequent changes after deployment.

c) Re-Deployment of Additional Nodes Phase

Additional sensor nodes can be re-deployed at any time to replace the malfunctioning nodes or due to changes in task dynamics. Addition of new nodes poses a need to re-organize the network. Coping with frequent topology changes in an ad hoc network that has myriads of nodes and very stringent power consumption constraints requires special routing protocols.

- **Transmission Media**

In a multi-hop sensor network, communicating nodes are linked by a wireless medium. These links can be formed by radio, infrared or optical media. To enable global operation of these networks, the chosen transmission medium must be available worldwide. One option for radio links is the use of *Industrial, Scientific and Medical* (ISM) bands, which offer license free communication in most countries.

- **Power Consumption**

The wireless sensor node, being a microelectronic device, can only be equipped with a limited power source. In some application scenarios, replenishment of power resources might be impossible. Sensor node lifetime, therefore, shows a strong dependence on battery lifetime. The malfunctioning of few nodes can cause significant topological changes and might require rerouting of packets and re-organization of the network. In sensor networks, power efficiency is an important performance metric, directly influencing the network lifetime. Application specific protocols can be designed by appropriately trading off other performance metrics such as delay and throughput with power efficiency. The main task of a sensor node in a sensor field is to detect events, perform quick local data processing, and then transmit the data. Power consumption can hence be divided into three domains: *sensing*, *communication*, and *data processing*.

1.4 OPEN ISSUES

Over the years, in order to meet the requirements of diverse applications on network QoS, significant effort has been made to provide end-to-end QoS support using various algorithms and mechanisms at different network protocol layers. Particularly, Internet QoS has been a focus of enormous research and development activities. Due to the many distinctive characteristics of WSNs, however, existing QoS mechanisms may not be applicable to WSNs. To achieve QoS support in WSN, the above challenges have to be addressed [2].

- **Service-Oriented Architecture**

Sensor nodes may have different and varying capabilities, be manufactured and operated by different vendors, and be accessed by multiple clients exercising different functionalities. A service-oriented architecture (SOA) is an architectural style encompassing a set of services for building complex systems of systems. It can be regarded as a model in which a system is decomposed into smaller, distinct units that are able to provide certain functionality. SOA offers flexibility in the design of WSN applications since it provides accepted standards for representing and packaging data, describing the functionality of services, and facilitating the search for available services which can be invoked to meet application requirements.

- **Qos-Aware Communication Protocols**

In order to efficiently support QoS in WSNs, communication protocols need to be designed with in mind the platform heterogeneity, specifically the heterogeneity between sensors that are involved in the communication. As an essential component of QoS, service differentiation should be supported by communication protocols. Obviously, the best-effort service offered by current wireless networking technologies such as Zigbee and Bluetooth cannot provide different QoS to different applications. Therefore, the communication protocols for WSNs should be designed to perceive the service requirement of each type of traffic so that it can be guaranteed a specific service level.

From a practice perspective, the best-effort service is likely to be the standard for the foreseeable future. It is therefore necessary for all new QoS mechanisms to be layered on top of the existing networks. Cross-layer design has proved to be effective in optimizing the network performance and hence may be incorporated in the development of QoS-aware communication protocols for WSNs.

- **Qos-Aware Power Management**

Energy conservation is a major concern in WSNs. The lifetime of sensor nodes is tightly restricted by the available battery energy. Since wireless communication is much more energy-expensive than sensing and computation, the transmission power of nodes has to be properly managed in a way that the energy consumption is minimized in order to prolong the lifetime of the whole network.

However, minimizing energy consumption and maximizing QoS are in most cases two conflicting requirements. For instance, reliability can be improved by increasing the number of maximum allowable retransmissions or using higher transmission power levels; however, more energy will be expended in both cases. Therefore, tradeoffs must be made between energy conservation and QoS optimization. The problem then becomes how to make these tradeoffs at runtime. Is it possible to find an integrated performance metric that accounts for both energy efficiency and QoS, and then optimize it, either online or offline?

QoS can be maximized through exploiting the different capabilities of sensors and actuators. In like manner, different transmission power levels may be assigned to the same node with respect to different types of traffic. In-network computation can be exploited to reduce the energy consumption of sensor since it reduces traffic load at the cost of slightly increased computation in each involved node. Still, the inherently non-deterministic and open nature of wireless channels poses great challenges for QoS-aware power management.

CHAPTER 2

LITERATURE REVIEW

2.1 SERVICE COMPOSITION IN WIRELESS SENSOR NETWORKS

Wireless sensor networks (WSNs) present a challenging programming environment because of their limited resources, heterogeneity, and highly dynamic nature. Service Oriented Architecture has become a promising paradigm for application development in wireless sensor networks due to its benefits in cost, efficiency, agility, and adaptability. In a service oriented WSN, a typical application requires several different services like data aggregation, data processing, decoding etc.., which are provided by service providers that are also sensors [3] . However, it might happen that there is no single service satisfying the requirement of the end user. So it is requires to form a composite service by combining one or more existing services.

Service composition is a key technology for building service oriented, loosely coupled integrated applications. Service Composition involves the development of customized services often by discovering, integrating, and executing existing services. This can be done in such a way that already existing services are combined into one or more new services that fit better to composite service.

For supporting the lifecycle of service compositions several aspects have to be addressed.

- Synthesis of service compositions deals with creation of service compositions which can happen both at design-time and run-time.

  Synthesis of service compositions is done in three different ways

  1. Automated Service Composition

     It is a family of approaches to service composition that aim at full or partial automation of the composition process, in order to enable handling

of higher levels of complexity. Automated Service Composition can be achieved using either Workflow-based or Planning techniques in order to create the Composition Schema.

2. Model-driven Service Composition

   Model-Driven Service Composition is a service composition that generates service orchestrations from a more general or abstract model.

3. Quality of Service-Aware Service Composition

   QoS-aware service composition is a form of service composition that is based on and attempts to improve the overall Quality of Service (QoS) of the composite service, such as execution time, reliability, availability or cost.

- After the creation of a service composition, verification techniques are needed for verifying the composition against certain properties
- After deployment of a service composition to the corresponding middleware, which for service orchestrations is typically a process engine in combination with a service bus, the composition is executed. At runtime, the composition can be adapted by for example rebinding other services, if a predefined service fails.
- Finally, monitoring of service compositions is performed either for run-time verification or to measure performance metrics of service compositions.

- **Service Discovery and Composition**

Service Discovery Protocol (SDP) is invoked to determine which nodes in the network provide which services. In a passive discovery protocol a provider advertises a service only when a request for that service has been received . This optimization minimizes the overall number of message transmissions, thus also minimizing power consumption.

The SDP maintains two service repositories

1. Local Service Repository (LSR) which catalogs the application services running locally,
2. Discovered Services Repository (DSR), which catalogs remote application services that have been discovered in the past.

When new service request arrives, the SDP locates a provider for that service by performing following steps. Note that the algorithm searches both service repositories in order to obtain a complete list of known providers.

- The service discovery algorithm receives as input a service ID, which if not present in either service repository, will prompt the SDP to broadcast a service request to other nodes in the network.
- The outgoing service discovery message contains the ID of the requested service and the node ID of the sender.
- Nodes providing the requested service will send a service discovery reply message, which includes information containing node vitals such as physical location and remaining power.
- The SDP caches the provider node ID in the DSR, and forwards the message to the Composer.
- It is the Composer's job to produce a set of services and service providers that satisfy the constraints specified in the service graph.
- These services are then bound and eventually invoked.

Because several instances of the same service could be residing on multiple nodes across the network, the Composer can expect multiple replies. As replies arrive, the Composer checks to see that any atomic service graph constraints are satisfied, and if so, the node information is stored (Compositional constraint satisfaction commences after all replies have been received. The

service composition is implemented as middleware. The general architecture for a middleware is shown in figure

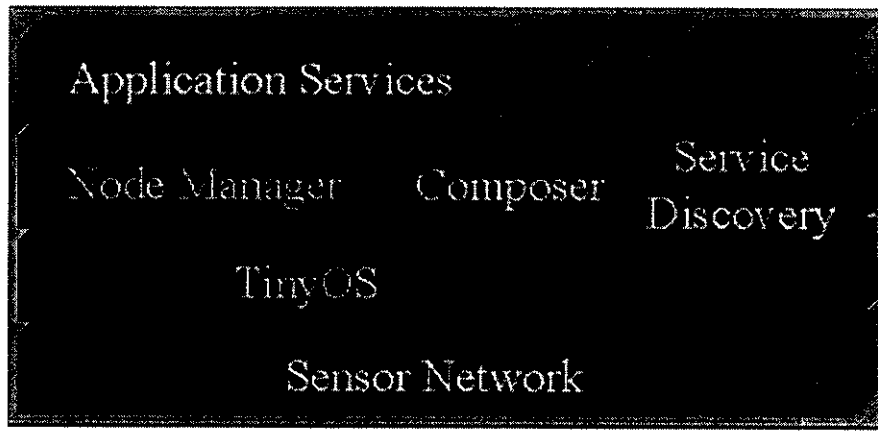


Figure 2.1 Middleware Architecture of WSN

2.2 EXISTING SYSTEM

Services in the service oriented sensor networks are resources capable of performing tasks that form a coherent functionality from the point of view of provider entities and requester entities. They have a well-defined interface which allows them to be described, published, discovered, and invoked over the network. Furthermore, services are modular and function autonomously, providing an excellent mechanism for application reconfiguration during runtime. Each service can have zero or more input ports and zero or more output ports. For example, the Localization service in our tracking example has four input ports, three for sensor readings and one for wind velocity, and one output port, on which the position estimate is placed. These service providers are sensor nodes in the WSN. Each provider registers itself with the register which is used as a repository of services by the service discovery. However the registry does not provide facility for registering QoS information of a service provider in the registry.

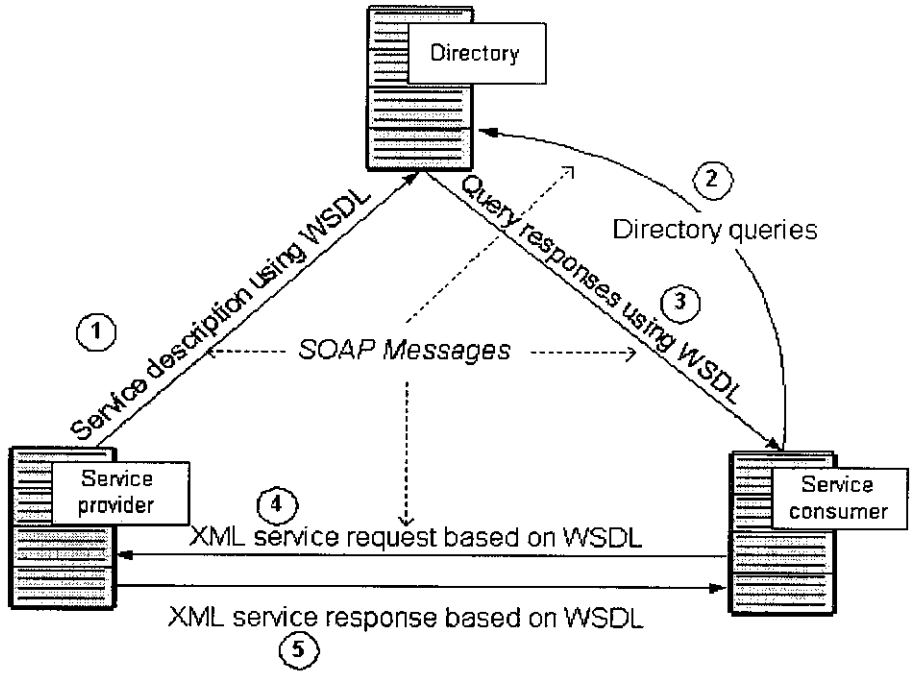


Figure 2.2 Service Oriented Architecture

Challenges

- More than one Service for a particular requirement.
- Registry does not contain QoS information of the service provider. Manually contact the service providers to obtain the QoS information .
- Service composition considers considers only single QoS constraint.
- Do not provides global optimization.

2.3 RELATED WORKS

Recently, QoS issues of WSN have obtained great interest in research community. However limited work has addressed service selection in the global. In most of research efforts presented in the literature have discussed the problem of service selection under single constraint with no global QoS parameter.

2.3.1 Qos-Guaranteed Path Selection Algorithm for Service Composition

In [4], authors proposed a depth first search heuristic algorithm *K-Closest Pruning (KCP)* to solve the problem of multi-constraint service path selection for Service Overlay Network in polynomial time. The main feature of this algorithm is that the path selected by this algorithm meets all the QoS requirements specified by the user/application. The main objective is to find a service-path, which is defined as sequence of nodes providing such that the request QoS constraints such as end-to-end loss, delay and available bandwidth are satisfied. A service path is said to satisfy a user QoS constraint only if the aggregated (end-to-end) node/link properties is better than the QoS constraint. This aggregation could be additive such as link delays or max-min such as service path capacity. If any link or node property violates the corresponding QoS requirement, then the service path containing such link or node would certainly fail to meet the QoS requirement. However in K-Closest Pruning (KCP) service composition is done with single constraint and it does not consider global QoS constraints.

2.3.2 Qos-Assured Service Composition in Managed Service Overlay Networks

In QUEST (Qos assUerd composEable Service infrasTructure) [5], service composition is performed by the service composer (SC) using a composed service provisioning protocol. The protocol is designed based on a network-centric client-service model. Instead of contacting service providers directly, the client contacts *SC* through a well-known address and specifies its desired services and QoS requirements. Then, *SC*,

as an intermediate agent, composes and instantiates a qualified service path in SON. The key algorithms used by SC include: (1) initial service composition, and *(2)* dynamic service composition. The initial service composition algorithm properly chooses and composes the service instances, based on their SLA contracts and current performances in order to best match the QoS constraints of the user. QUEST achieves not only QoS assurances but also load balancing in SON by comprehensively considering both QoS and resources of different service instances. Moreover, QUEST provides dynamic service composition, which is used during runtime when service outages or QoS violations occur. It also finds an alternative service path in SON, which can quickly recover the failed composed service delivery. QUEST addresses service composition with multiple constraints but still global QoS is not addressed.

2.3.3 A QoS Aware Service Composition Protocol in Mobile Ad Hoc Networks

In [6], authors propose a distributed QoS-based service composition protocol for MANETs. The protocol uses flooding-based source route techniques to broadcast service composition messages in search of the optimal service composition path which meets QoS constraints and minimizes resource consumption. As a flooding-based protocol may create a large number of control messages, some measures must be taken to reduce the control message overhead and eliminate the broadcast storm problem. However due to the energy constraints of WSN this approach may not be suitable for service composition in WSN.

CHAPTER 3

METHODLOGY

3.1 QOS - AWARE SERVICE COMPOSITION

QoS is increasingly important when composing services because a degrading QoS in one of the services can negatively affect the QoS of the overall composition. Protocols and mechanisms follow QoS aware service composition. QoS aware service composition system in WSN consists of following components [7].

- **Registry**

  In Service Oriented Wireless sensor Networks service providers are sensor nodes and sensor nodes register itself with the registry. Registry is the repository of registered services which contains service description as well as QoS information.

- **Translator and Planner**

  Translator and Planner components are used in composition schema development.

- **Selector**

  Selector is the component responsible for selecting appropriate service for each task in the service composition. Selector obtains QoS information of each service via Qos Manager and uses this for selection process.

- **QoS Manager**

  QoS Manager is responsible for updating latest QoS information of each service provider (by monitoring the registered service providers' mechanisms in run time or advertised by the service provider or fed back by the users) and saving updates to the registry.

- **Execution Engine**

  The generated selection plan is given to Execution engine for execution.

If the selecting service becomes unavailable during execution then a request is send back to selector for replanning which consumes more energy. Since energy consumption is the critical issue in WSN Selector uses sleeping schedule information of the sensor nodes for selection process. It is assumed that Registry component of the Dynamic service composition system contains availability information of sensor nodes.

3.2 MODULES

The QoS aware service composition in service oriented wireless sensor networks is divided in to two steps: composition schema planning and service selection. In the first step AI planning can be used to automatically derive the composition schema of a given composite service. In the second step, an appropriate service is selected from a group of service for each task in the composite service. However this work mainly concentrates on QoS aware service selection.

3.2.1 Service Registration

To better facilitate the service selection process, registry has both functional as well non-functional information such as QoS information. For each service providers six QoS parameters are defined. Service selection involves selecting a appropriate service from a group of candidate service with high values of QoS parameters which in turn results in better service composition.

The following parameters are defined.

- **Throughput:** The number of requests served in a given period.
- **Time:** Time taken to deliver a service. This is defined as follows

  Processing time + Transmission time.
- **Cost:** It is the cost of executing a service.
- **Reliability:** Probability that a service responded correctly with in a maximum expected time frame. It is calculated as

  $$\frac{\text{Number of successful execution}}{\text{Total number of executions}}$$

- **Availability:** Residual energy of the sensor node.
- **Security:** Related to the ability of a given service to provide suitable security mechanisms by considering the following three parameters.
    1. Encryption
    2. Authentication
    3. Access Control

3.2.2 QoS Aware Service Selection

One of the main issues in service oriented WSN is to select appropriate component service for a service composition so as to fulfill the user request. For services providing similar functionality, Quality of Service (QoS) is the main factor to differentiate them. This leads to general optimization problem of how to select services for each task so that the overall QoS requirements of the composition are satisfied and optimal. This work mainly addresses the issue of QoS-based service selection problem involved in service composition and proposes a model for service selection based on QoS. The service selection algorithm selects a best service among a bunch of services with similar functions but with different QoS for each task of a composite service based on global QoS constraints set by users.

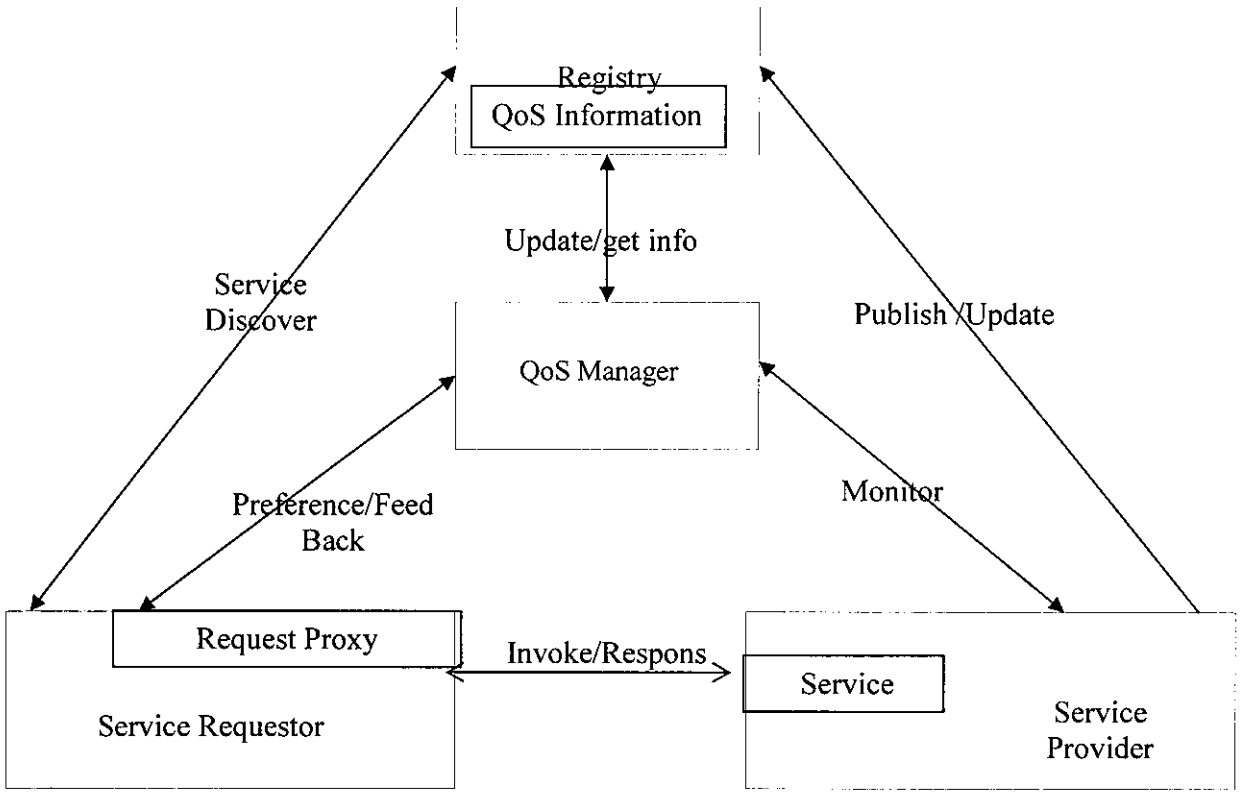


Figure 3.1 Architecture of QoS Aware Service Selection

3.2.3 Comparative Analysis

The results are compared with the existing methodology implementation.

3.3 PROBLEM DEFINITION

Let us consider a composite service containing *n* tasks, *t1, t2, ... , tn.* For each task *ti (l≤ i ≤n)*, there are *li* candidate services that can perform the task. Users may impose some constraints on the QoS parameters (e.g. execution time, cost, reliability). Let us assume that *m* QoS parameters are *(r1, r2..., rm)* taken for consideration. Then, the QoS-based service selection problem decides how to select one service for each involving

task from its corresponding candidate service group, so that the overall QoS of the constructed composite service can be maximized while the constraints set.

The mathematical model of a QoS optimization problem with multiple global constraints can be expressed as follows:

$$\text{Max} \left(\sum_{i=1}^{n} \sum_{j=1}^{li} x_{ij} v_{ij} \right)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{li} xi_j r_{ij}^{k} \leq R$$

Where $x_{ij} \in \{0, 1\}$ are picking variables satisfying $\forall i \in \{1,2,\ldots, n\}$, $\sum_{j=1}^{li} x_{ij} = 1$, v_{ij} is the QoS score of the j^{th} candidate service for the i^{th} task in the composition $r_{ij} = (r_{ij}^{1}, r_{ij}^{2}, r_{ij}^{3} \ldots r_{ij}^{3})$ represents the resources required by the j^{th} candidate service for the i^{th} task in the composition and $R= (R^1,R^2,R^3 \ldots R^m)$ are the upper limits of these resources set by user.

3.4 THE ALGORITHM

This work proposes an efficient heuristic algorithm to solve the QoS based service selection for composite services with multiple global constraints. The algorithm uses the concept of constructing the convex hull [8] [9] of related point for approximation. In this approach every actual (candidate) service can be mapped to a point in the two-dimensional space which has an x-coordinate indicating the resource constraints of the service and a y-coordinate indicating its QoS score. So each task in the composition corresponds to a set of points in the space. If there are *n* tasks involved in the composition model, there will be *n* sets of points, each containing l_i points respectively. The objective is to pick up exactly one point from each set of points so that the total resource consumption represented by the selected *n* points can meet the user's requirements while maximizing their overall QoS value. To achieve this, the convex hulls of those sets of points are respectively constructed through use of algorithms Graham-scan. *N* efficient

convex hull frontiers are computed, which are the segments connecting the highest point and the lowest one in the hull [7]. Then the gradients (i.e. $\Delta y / \Delta x$) of these segments can be used as heuristic information for point selection.

The algorithm is presented as follows:

Algorithm

Initialize the solution vector X= (x1,x2,…,xn) ;

transformer = initial_transformer();

For rep = 1 to 3 do

{

X* = X ; *//save current solution*

f =ftotal(X) ; *//save current total score*

For i = 1 to n do

{

transform the resource consumption vector of each candidate to single dimension using the vector *transformer*;

I_frontier = the resulted efficient convex hull frontier of the point set corresponding to all the candidate services similar to xi in (xresource,y-score) space by calling convex_hull_frontier method;

frontier_segments = frontier_segments + I_frontier;

}

Sort all the segments in frontier_segments in descending order according to the angle between the segment and the positive x-axis;

For each segment in frontier_segments do

{

p1, p2 = endpoints of the segment;

Satisfied(p1) = whether all constraints are satisfied after the service represented by p1 is selected;

if Satisfied(p1) = true then

{

Change the selection for the corresponding task from the current

```
                selected service to the one represented by p1;
                update X ;
            }

            Satisfied (p2) = whether all constraints are satisfied after the service
            represented by p2 is selected ;
            if Satisfied(p2) = true then
            {
                Change the selection for the corresponding task from the current
                selected service to the one represented by p2;
                update X ;
            }
        }
        If ftotal(X)<f
        then X =X* ; //if new solution is inferior to the saved one, restore it
        transformer = adjust_transformer (transformer) ;
}
Output the solution X
```

In the above algorithm, transformer is an m-dimensional vector used to transform the m constrained resources to one dimension, in other words, to give an overall cost for each candidate service. If transformer is $q = (q_1,q_2,q_3..q_m)$ and the constrained resource vector is $r = (r_1,r_2,r_3..r_m)$ the transformed resource vector will be $Rp=(r_1q_1,r_2.q_2....r_m.q_m)$. The formula for calculating single dimension will be

$$R^* = \sqrt{Rp_1^2 + Rp_2^2 ... Rp_m^2}$$

The formula used for initial transformer will be

$$q_k = rsum_k / R_k + 1 \quad (k=1,2,3...m)$$

Where R represents the total number of resources, rsum represents the vector summation of resource consumption vectors of every candidate service for every task and

k is the number of resource constraints. Where R represents the total number of resources, rsum represents the vector summation of resource consumption vectors of every candidate service for every task and k is the number of resource constraints.

3.5 SYSTEM SPECIFICATION

3.5.1 Hardware Requirement

Processor	: Pentium III
Clock speed	: 550MHz
Hard Disk	: 20GB
RAM	: 128MB
Cache Memory	: 512KB
Operating System	: Windows2000
Monitor	: Color Monitor
Keyboard	: 104Keys
Mouse	: 3Buttons

3.5.1 Software Requirement

IDE	: Eclipse with JDK 1.5
Server	: Tomcat server
Database	: MS Access

CHAPTER 4

RESULTS

It is assumed that the composite service contains three tasks. For each task there are twelve instances available in each candidate service group. Each service is represented in two-dimensional space where x-axis indicates single dimension resource constraints (throughput, time, and cost) and y-axis indicates single dimension QoS score (reliability, security, availability).

Table 4.1 shows candidate services for task1. Each service has the following QoS information: reliability (Q1), security (Q2), availability (Q3), throughput (Q4), time (Q5), and cost (Q6). Table 4.2 shows the candidate services for task1 after constructing convex hull. The efficient convex hull frontier segments are sorted in descending order based on the gradient (i.e. ΔQoS score / Δresource constraints) and is used as a heuristic information for service selection. The point with highest gradient has highest QoS score and least resource consumption.

	Q1	Q2	Q3	Q4	Q5	Q6
S1	0.8	2	0.3	98	21	30
S2	0.5	4	0.6	55	32	35
S3	0.45	3	1	67	24	22
S4	0.53	7	0.8	89	12	21
S5	0.75	7	0.2	67	12	12
S6	0.28	6	0.3	90	14	18
S7	0.5	5	0.6	47	21	30
S8	0.4	7	0.3	126	32	35
S9	0.3	4	0.9	117	24	22
S10	0.6	3	0.5	67	12	21
S11	0.7	2	1.2	125	12	12
S12	0.2	1.0	0.8	78	14	18

Table 4.1 QoS Attribute Values of Candidate services

	Q1	Q2	Q3	Q4	Q5	Q6
S7	0.5	5	0.6	47	21	30
S5	0.75	7	0.2	67	12	12
S8	0.4	7	0.3	126	32	35
S3	0.45	3	1	67	24	22
S11	0.7	2	1.2	125	12	12
S12	0.2	1.0	0.8	78	14	18
S1	0.8	2	0.3	98	21	30

Table 4. 2. QoS Information of Optimal Services - After constructing Convex hull

The results of QoS aware service selection is compared with branch and bound technique [8]. Table 3 and 4 shows the QoS information of both algorithms. The QoS aware service selection using convex hull yields better QoS.

	Q1	Q2	Q3	Q4	Q5	Q6	QoS
S7	0.5	5	0.6	77	21	22	8
S3	0.55	5	0.6	68	15	14	5
S6	0.25	6	0.8	45	12	21	5

Table 4.3. Service selection using Convex Hull approach

	Q1	Q2	Q3	Q4	Q5	Q6	QoS
S3	0.45	3	1	67	24	30	5
S6	0.35	4	0.7	57	25	15	4
S6	0.25	6	0.8	45	12	21	5

Table 4. 4. Service selection using Branch and Bound technique

- **Candidate services**

The registered services are plotted as points in 2-dimensional plane, whose x-axis incates resource constraints and y-axis indicate QoS score

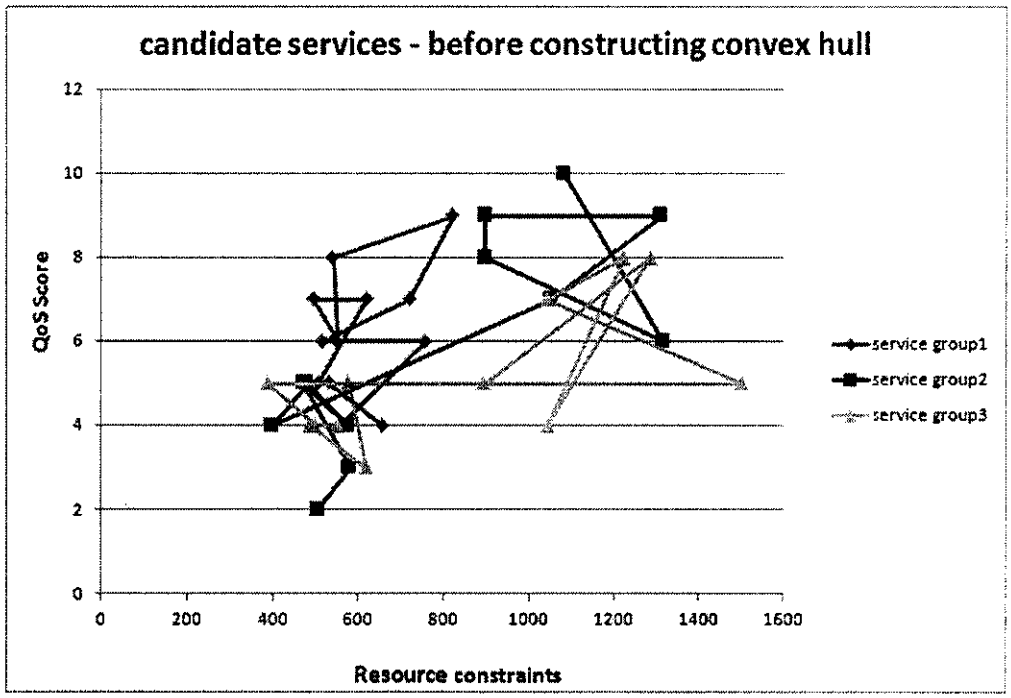


Figure 4.5 Candidate Services

- **Sorted Services**

The service selection algorithm produces set of convex hull frontier segments. The efficient convex hull frontier segments are sorted in descending order based on the gradient (i.e. ΔQoS score / Δresource constraints) and is used as a heuristic information for service selection. The point with highest gradient has highest QoS score and least resource consumption.

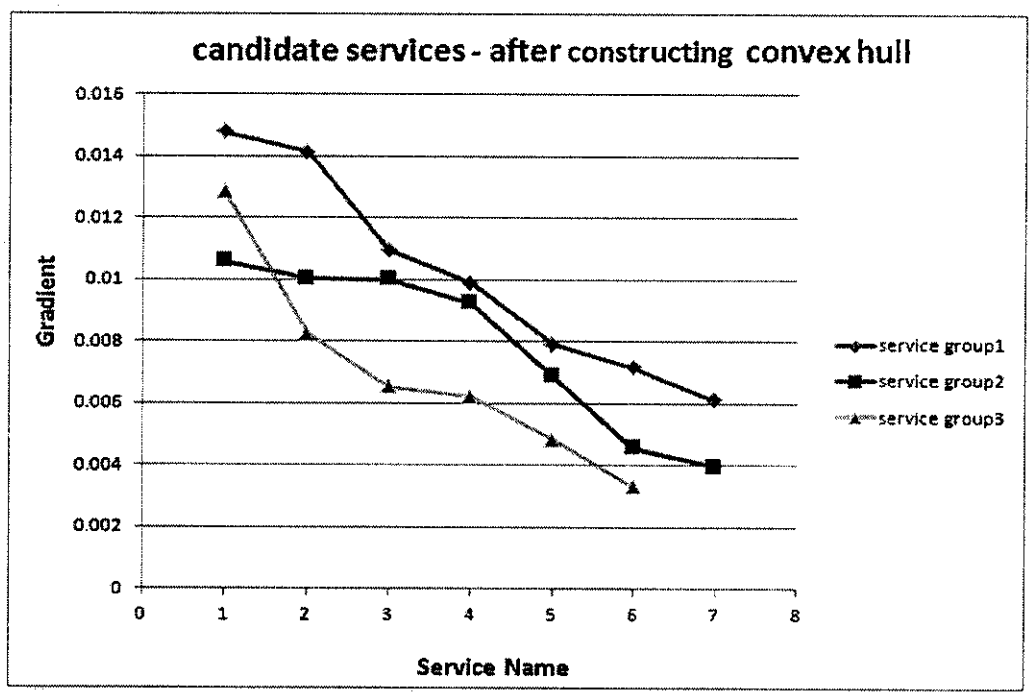


Figure 4.6 Sorted Services - After Constructing Convex Hull

- **Comparative Analysis**

The results of QoS aware service selection is compared with branch and bound technique [8] in terms of QoS and results shows that the QoS achieved by the proposed method is better than the existing one.

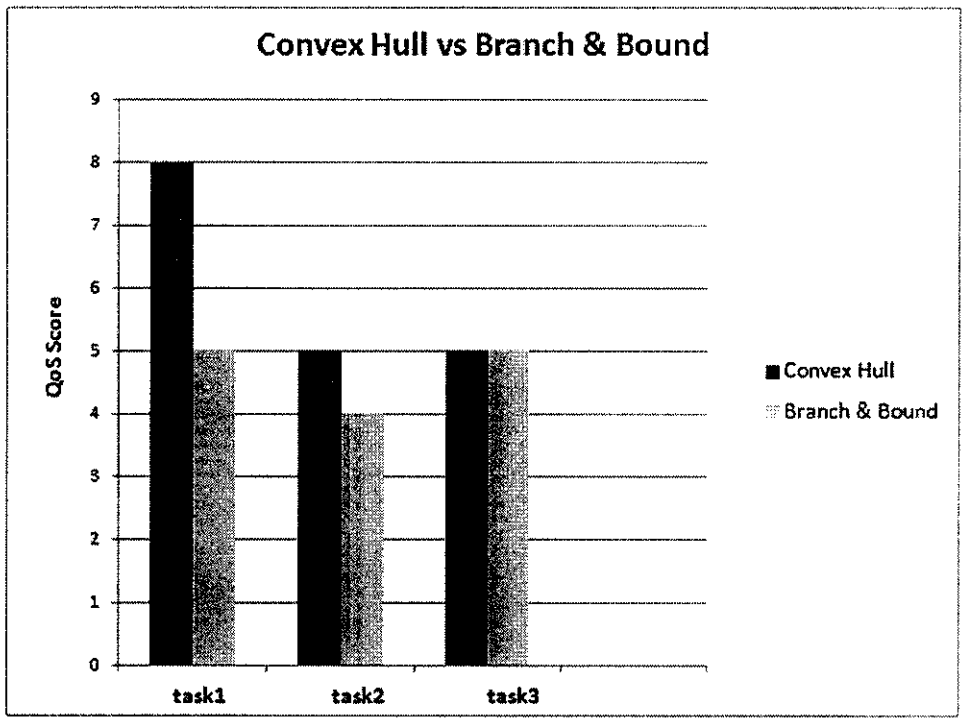


Figure 4.7 Comparisons of QoS of a Composite Service

CHAPTER 5

CONCLUSION AND FUTUTE OUTLOOK

5.1 CONCLUSION

Recently, QoS issues of WSN have obtained great interest in research community. This work mainly addresses the issue of QoS-based service selection problem involved in service composition. To better facilitate the service selection process, registry has both functional as well non-functional information such as QoS information. As we discussed above, the QoS-based service selection problem for composite services determines how to select one service for each involving task from its corresponding candidate service group, so that the overall QoS of the constructed composite service can be maximized while the constraints set by users are satisfied. The mathematical model is established for multiple global constraint service selection. The implementation constructs convex hull frontier segments for each service in the candidate service group. The frontier segment of each service group is sorted in descending order based on their gradients. During selection process the point with highest gradient is selected which yields better QoS with least resource consumption. Comparing to the existing technique the QoS of a composite service is increased to 40% when the proposed method is used.

5.2 FUTURE OUTLOOK

The proposed service selection method is implemented in the base station or gateway of a WSN as a centralized approach. However this may suffer from single point of failure if the base station is down and care must be taken to handle such situation. In future enhancements this can be implemented as a distributed approach to avoid such failures.

CHAPTER 6

APPENDIX

6.1 Source Code

 # Coding for service description

```
package com.QWSEM.registry;
import java.io.Serializable;
public class ServiceDescription implements Serializable
{
        private static final long serialVersionUID = 8801892407722922166L;
        private String serviceKey;
        private String serviceName;
        private String serviceDescription;
        private String serviceURL;
        private String qos1;
        private String qos2;
        private String qos3;
        private String qos4;
        private String qos5;
        private String qos6;

        /**
         * @return the serviceKey
         */
        public String getServiceKey() {
                return serviceKey;
        }
        /**
```

```
 * @param serviceKey
 *            the serviceKey to set
 */
public void setServiceKey(String serviceKey) {
        this.serviceKey = serviceKey;
}
/**
 * @return the serviceName
 */
public String getServiceName() {
        return serviceName;
}
/**
 * @param serviceName
 *            the serviceName to set
 */
public void setServiceName(String serviceName) {
        this.serviceName = serviceName;
}
/**
 * @return the serviceDescription
 */
public String getServiceDescription() {
        return serviceDescription;
}
/**
 * @param serviceDescription
 *            the serviceDescription to set
 */
public void setServiceDescription(String serviceDescription) {
        this.serviceDescription = serviceDescription;
```

```
}
/**
 * @return the serviceURL
 */
public String getServiceURL() {
        return serviceURL;
}
/**
 * @param serviceURL
 *          the serviceURL to set
 */
public void setServiceURL(String serviceURL) {
        this.serviceURL = serviceURL;
}
/**
 * @return the qos1
 */
public String getQos1() {
        return qos1;
}
/**
 * @param qos1
 *          the qos1 to set
 */
public void setQos1(String qos1) {
        this.qos1 = qos1;
}
/**
 * @return the qos2
 */
public String getQos2() {
```

```
        return qos2;
}
/**
 * @param qos2
 *          the qos2 to set
 */
public void setQos2(String qos2) {
        this.qos2 = qos2;
}
/**
 * @return the qos3
 */
public String getQos3() {
        return qos3;
}
/**
 * @param qos3
 *          the qos3 to set
 */
public void setQos3(String qos3) {
        this.qos3 = qos3;
}
/**
 * @return the qos4
 */
public String getQos4() {
        return qos4;
}
/**
 * @param qos4
 *          the qos4 to set
```

```
 */
public void setQos4(String qos4) {
        this.qos4 = qos4;
}

/**
 * @return the qos5
 */
public String getQos5() {
        return qos5;
}
/**
 * @param qos5
 *          the qos5 to set
 */
public void setQos5(String qos5) {
        this.qos5 = qos5;
}
/**
 * @return the qos6
 */
public String getQos6() {
        return qos6;
}
/**
 * @param qos6
 *          the qos6 to set
 */
public void setQos6(String qos6) {
        this.qos6 = qos6;
}
```

```
}
```

 # Coding for service selection

```
package compgeom.algorithms;
import compgeom.RLine2D;
import compgeom.RPoint2D;
import compgeom.RPolygon2D;
import compgeom.Rational;
import compgeom.util.Utils;
import java.util.*;
public class GrahamScan {
    private GrahamScan() {
  }


  public static List<RPoint2D> getConvexHull(int[] xs, int[] ys)
      throws IllegalArgumentException {
    return getConvexHull(new RPolygon2D(Utils.createRPoint2DList(xs, ys)));
  }
  public static List<RPoint2D> getConvexHull(List<RPoint2D> points)
      throws IllegalArgumentException {
    return getConvexHull(new RPolygon2D(points));
  }

  public static List<RPoint2D> getConvexHull(RPolygon2D polygon) {

    Set<RPoint2D> points = new HashSet<RPoint2D>(polygon.getPoints());
    List<RPoint2D> sorted = getSortedFromLowestY(points);
    removeCollinearPoints(sorted);

    Stack<RPoint2D> stack = new Stack<RPoint2D>();
```

```
    stack.push(sorted.get(0));
    stack.push(sorted.get(1));
    stack.push(sorted.get(2));

    for (int i = 3; i < sorted.size(); i++) {
      RPoint2D head = sorted.get(i);
      RPoint2D middle = stack.pop();
      RPoint2D tail = stack.pop();
      if (Utils.formsLeftTurn(tail, middle, head)) {
        stack.push(tail);
        stack.push(middle);
        stack.push(head);
      } else {
        stack.push(tail);
        i--;
      }
    }
    stack.push(sorted.get(0));
    return new ArrayList<RPoint2D>(stack);
  }


  private static List<RPoint2D> getSortedFromLowestY(Set<RPoint2D> points) {
    List<RPoint2D> sorted = new ArrayList<RPoint2D>(points);
    final RPoint2D lowestY = Utils.getPointWithLowestY(points);
    Collections.sort(sorted, new Comparator<RPoint2D>() {
      @Override
      public int compare(RPoint2D pA, RPoint2D pB) {
        if (pA.equals(lowestY)) return -1;
        if (pB.equals(lowestY)) return 1;
```

```
            Rational slopeA = new RLine2D(lowestY, pA).m;
            Rational slopeB = new RLine2D(lowestY, pB).m;

            if (slopeA.equals(slopeB)) {
               Rational distanceAToLowestY = pA.distanceXY(lowestY);
               Rational distanceBToLowestY = pB.distanceXY(lowestY);
               return distanceAToLowestY.compareTo(distanceBToLowestY);
            }

            if (slopeA.isPositive() && slopeB.isNegative()) {
               return -1;
            } else if (slopeA.isNegative() && slopeB.isPositive()) {
               return 1;
            } else {
               return slopeA.compareTo(slopeB);
            }
         }
      });
      return sorted;
   }

   private static void removeCollinearPoints(List<RPoint2D> sorted) {
      RPoint2D lowestY = sorted.get(0);
      for (int i = sorted.size() - 1; i > 1; i--) {
         Rational slopeB = new RLine2D(lowestY, sorted.get(i)).m;
         Rational slopeA = new RLine2D(lowestY, sorted.get(i - 1)).m;
         if (slopeB.equals(slopeA)) {
            sorted.remove(i - 1);
         }
      }
   }
```

```
}


package compgeom.algorithms;
import compgeom.*;
import compgeom.util.Utils;
import java.util.*;
public class RotatingCalipers
 {

   /* the convex hull of the polygon to get the bounding rectangles from */
   private static List<RPoint2D> convexHull;

   /* the array that holds the four calipers to be rotated around the convex hull */
   private static Caliper[] calipers;

   // some final numbers used in this class
   private final static Rational zero = Rational.ZERO;
   private final static Rational minOne = Rational.MINUS_ONE;
   private final static Rational inf = Rational.POSITIVE_INFINITY;

   private RotatingCalipers()
   {
   }

   public static List<RRectangle> getBoundingRectangles(int[] xs, int[] ys)
         throws IllegalArgumentException {
     return getBoundingRectangles(xs, ys, false);
   }
```

```
  public static List<RRectangle> getBoundingRectangles(int[] xs, int[] ys, boolean
convex)
        throws IllegalArgumentException {
    return getBoundingRectangles(Utils.createRPoint2DList(xs, ys), convex);
  }


  public static List<RRectangle> getBoundingRectangles(List<RPoint2D> points)
        throws IllegalArgumentException {
    return getBoundingRectangles(points, false);
  }


  public static List<RRectangle> getBoundingRectangles(List<RPoint2D> points,
boolean convex)
        throws IllegalArgumentException {
    return getBoundingRectangles(new RPolygon2D(points), convex);
  }

  public static List<RRectangle> getBoundingRectangles(RPolygon2D polygon) {
    return getBoundingRectangles(polygon, false);
  }

  public static List<RRectangle> getBoundingRectangles(RPolygon2D polygon,
boolean convex) {
    convexHull = convex ? polygon.getPoints() : polygon.getConvexHull();
    Collections.reverse(convexHull);
    LinkedHashSet<RRectangle> rectangles = new LinkedHashSet<RRectangle>();
    initCalipers();

    // Keep looping while unique rectangles are added to the set.
```

```
        while (rectangles.add(new RRectangle(calipers[0].line, calipers[1].line,
                calipers[2].line, calipers[3].line))) {

            // Get the next smallest positive tangent from the calipers.
            Rational tangent = getSmallestPositiveTangent();


            if (tangent == null || tangent.isNegative() || rectangles.size() == convexHull.size())
{
                break;
            }

            // Rotate all calipers.
            rotateCalipersBy(tangent);
        }
        return new ArrayList<RRectangle>(rectangles);
    }

    public static RRectangle getMinimumBoundingRectangle(int[] xs, int[] ys)
            throws IllegalArgumentException {
        return getMinimumBoundingRectangle(xs, ys, false);
    }


    public static RRectangle getMinimumBoundingRectangle(int[] xs, int[] ys, boolean
convex)
            throws IllegalArgumentException {
        return getMinimumBoundingRectangle(Utils.createRPoint2DList(xs, ys), convex);
    }

    public static RRectangle getMinimumBoundingRectangle(List<RPoint2D> points)
```

```
        throws IllegalArgumentException {
    return getMinimumBoundingRectangle(points, false);
  }

  public static RRectangle getMinimumBoundingRectangle(List<RPoint2D> points,
boolean convex)
        throws IllegalArgumentException {
    return getMinimumBoundingRectangle(new RPolygon2D(points), convex);
  }


  public static RRectangle getMinimumBoundingRectangle(RPolygon2D polygon) {
    return getMinimumBoundingRectangle(polygon, false);
  }

  public static RRectangle getMinimumBoundingRectangle(RPolygon2D polygon,
boolean convex) {
    List<RRectangle> boxes = getBoundingRectangles(polygon, convex);
    RRectangle minimum = null;
    for (RRectangle rectangle : boxes) {
      if (minimum == null || rectangle.area() < minimum.area()) {
        minimum = rectangle;
      }
    }
    return minimum;
  }

    private static Rational getSmallestPositiveTangent() {
    Rational smallestTangent = null;
    for (Caliper c : calipers) {
      Rational temp = c.getTangentNextCaliper();
```

```
        if (temp.isLessThanEq(Rational.ZERO)) continue;
        if (smallestTangent == null || temp.isLessThan(smallestTangent)) {
            smallestTangent = temp;
        }
    }
    return smallestTangent;
}


private static void initCalipers() {
    RPoint2D pI, pJ, pK, pL;
    int iI, iJ, iK, iL;
    pI = pJ = pK = pL = convexHull.get(0);
    iI = iJ = iK = iL = 0;
    for (int i = 1; i < convexHull.size(); i++) {
        RPoint2D p = convexHull.get(i);

        if (p.x.isLessThan(pI.x) || (p.x.equals(pI.x) && p.y.isMoreThan(pI.y))) { // i
            pI = p;
            iI = i;
        }
        if (p.y.isMoreThan(pJ.y) || (p.y.equals(pJ.y) && p.x.isMoreThan(pJ.x))) { // ii
            pJ = p;
            iJ = i;
        }
        if (p.x.isMoreThan(pK.x) || (p.x.equals(pK.x) && p.y.isLessThan(pK.y))) { // iii
            pK = p;
            iK = i;
        }
        if (p.y.isLessThan(pL.y) || (p.y.equals(pL.y) && p.x.isLessThan(pL.x))) { // iv
            pL = p;
```

```
            iL = i;
        }
    }
    calipers = new Caliper[]{
            new Caliper(new RPoint2D(pI.x, pI.y.minusOne()), pI, iI),
            new Caliper(new RPoint2D(pJ.x.minusOne(), pJ.y), pJ, iJ),
            new Caliper(new RPoint2D(pK.x, pK.y.plusOne()), pK, iK),
            new Caliper(new RPoint2D(pL.x.plusOne(), pL.y), pL, iL)
    };
}


private static void rotateCalipersBy(Rational minTangent) {
    Rational slopeIK = null;
    Rational slopeJL = null;
    Stack<Integer> todo = new Stack<Integer>();

    // First handle all calipers that have the smallest 'tangent', can be more than one!
    for (int i = 0; i < calipers.length; i++) {
        if (minTangent.equals(calipers[i].getTangentNextCaliper())) {
            calipers[i] = calipers[i].nextCaliper();
            if (i == 0 || i == 2) { // I or K
                slopeIK = calipers[i].getSlope();
            } else {              // J or L
                slopeJL = calipers[i].getSlope();
            }
        } else {
            todo.push(i);
        }
    }
```

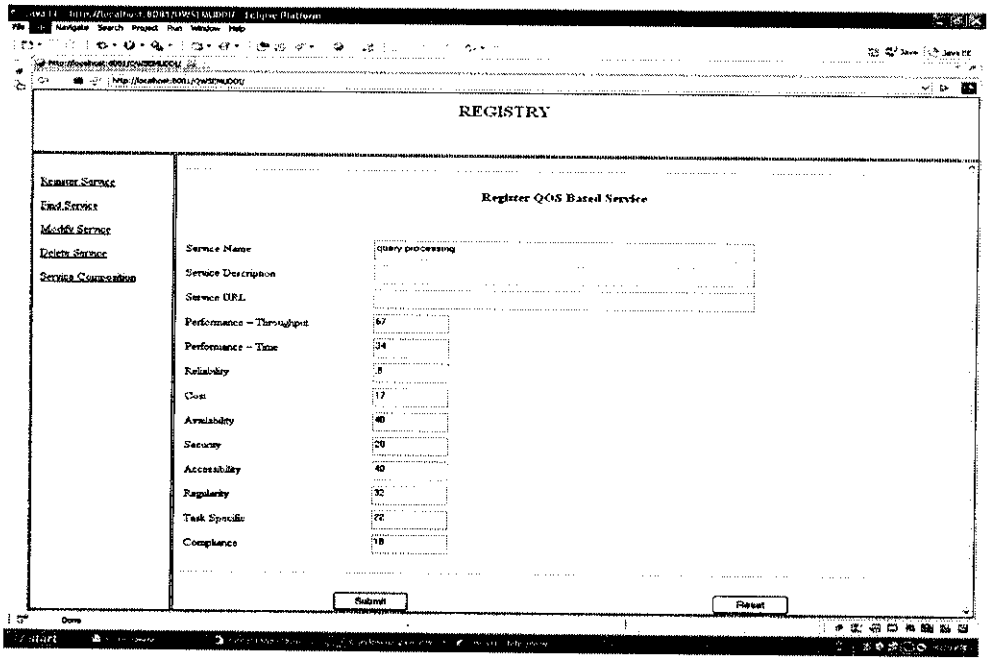
```
    // Set the slopes that have not yet been set. One is definitely assigned, but
    // it is possible both have been set already.
    if (slopeIK == null) slopeIK = zero.equals(slopeJL) ? inf : minOne.divide(slopeJL);
    if (slopeJL == null) slopeJL = zero.equals(slopeIK) ? inf : minOne.divide(slopeIK);

    // Handle the calipers that have not yet been rotated (could even be none!).
    while (!todo.isEmpty()) {
      int index = todo.pop();
      if (index == 0 || index == 2) { // I or K
        calipers[index] = calipers[index].nextCaliper(slopeIK);
      } else {                        // J or L
        calipers[index] = calipers[index].nextCaliper(slopeJL);
      }
    }
  }


  private static class Caliper
{
    RPoint2D turnPoint;
    RLine2D line;
     private int index;
    Caliper(RPoint2D p1, RPoint2D p2, int idx) {
      turnPoint = p2;
      line = new RLine2D(p1, p2);
      setIndex(idx);
    }

    private Caliper(Rational slope, RPoint2D p, int idx) {
      turnPoint = p;
      line = new RLine2D(slope, turnPoint);
```

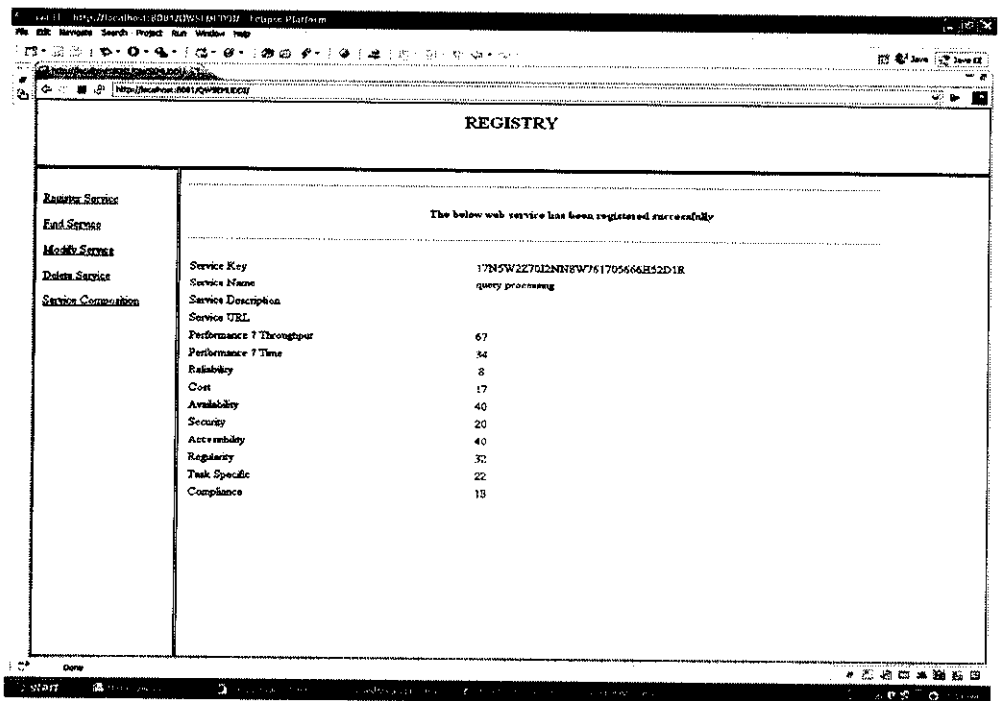
```
    setIndex(idx);
  }

  Rational getSlope() {
    return line.m;
  }

  Rational getTangentNextCaliper() {
    return line.tangent(this.nextCaliper().line);
  }

  private RPoint2D nextHead() {
    return convexHull.get(index + 1);
  }

  Caliper nextCaliper() {
    return new Caliper(turnPoint, nextHead(), index + 1);
  }

  Caliper nextCaliper(Rational slope) {
    return new Caliper(slope, turnPoint, index);
  }

  private void setIndex(int idx) {
    // Note that the last point in the convex hull is the same as the first,
    // that's why the last element, or the index of that element, is skipped
    index = idx == convexHull.size() - 1 ? 0 : idx;
  }

  @Override
  public String toString() {
```

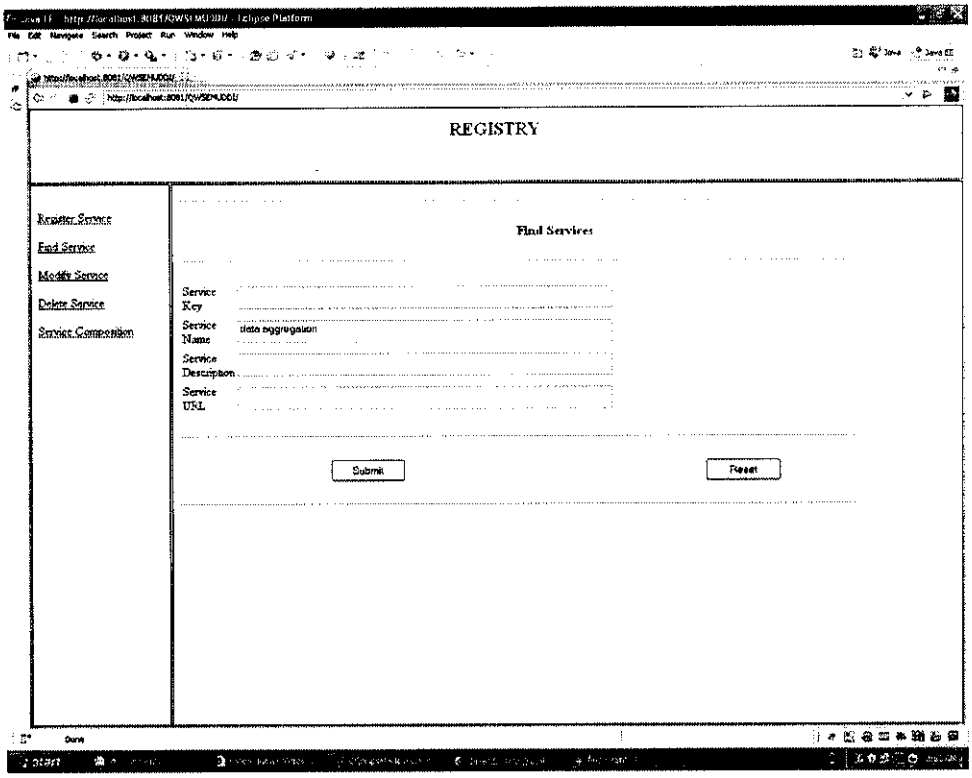
```
            return String.format("Caliper[line=%s, rotation-point=%s]", line, turnPoint);
        }
    }
}


package compgeom.demos;

import compgeom.RPoint2D;
import compgeom.RPolygon2D;
import compgeom.RRectangle;
import compgeom.algorithms.GrahamScan;
import compgeom.algorithms.RotatingCalipers;
import compgeom.util.Utils;

import java.util.List;
public class Main
{

  public static void main(String[] args)
 {

    // create a list of points
    List<RPoint2D> points1 = Utils.createRPoint2DList(
    new int[]{656,533,506,620,495,551,541,822,                    720,517,758,558},
     new int[]{4,5,5,7,7,6,8,9,7,6,6,4});

        List<RPoint2D> points2 = Utils.createRPoint2DList ( new                    int[]
                              {505,581,472,576,481,400,1052,1310,898,897,1315,1082},
       new int[]{2,3,4,5,4,5,7,9,9,8,6,10});
```

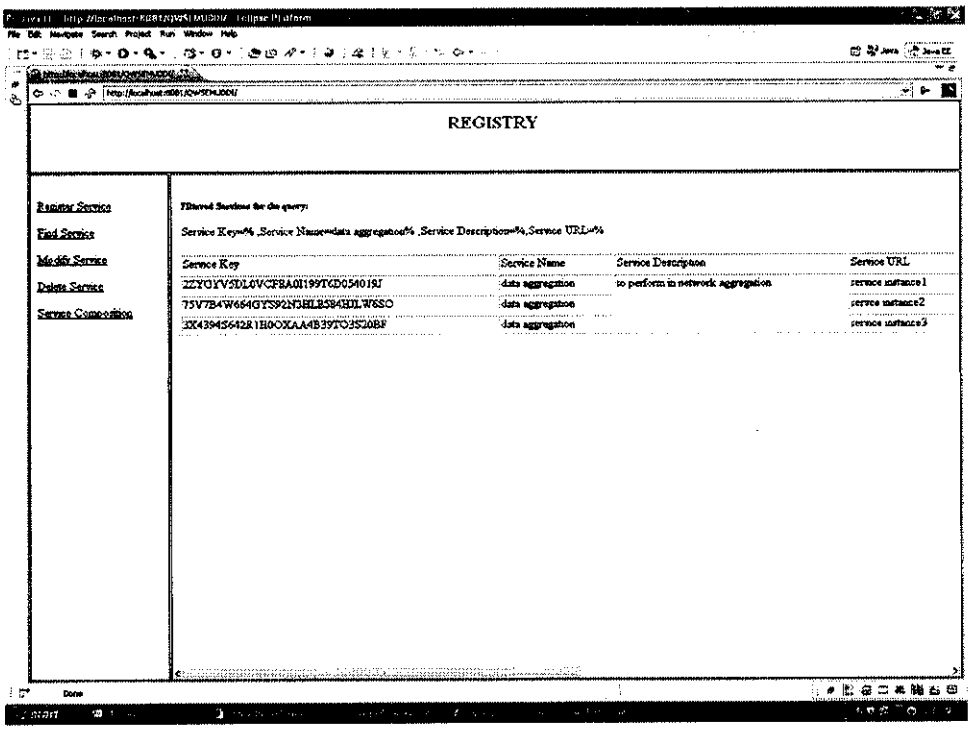
```
    List<RPoint2D> points3 = Utils.createRPoint2DList(
    new int[]{578,620,485,553,501,389,1505,1042,1223,1044,                1286,896},

     new int[]{5,3,4,4,5,6,7,8,4,8,5});

    // find the minimum bounding rectangle of the list of points
      RRectangle rectangle1 =
RotatingCalipers.getMinimumBoundingRectangle(points1);
      RRectangle rectangle2 =
RotatingCalipers.getMinimumBoundingRectangle(points2);
      RRectangle rectangle3 =
RotatingCalipers.getMinimumBoundingRectangle(points3);
      //System.out.println(rectangle);

      // get the convex hull of the list of points
      List<RPoint2D> convexHull1 = GrahamScan.getConvexHull(points1);
      System.out.println("\nconvexHull1 = " + convexHull1);

      List<RPoint2D> convexHull2 = GrahamScan.getConvexHull(points2);
      System.out.println("\nconvexHull2 = " + convexHull2);

      List<RPoint2D> convexHull3 = GrahamScan.getConvexHull(points3);
      System.out.println("\nconvexHull3 = " + convexHull3);
   }
}
```

6.2 SNAPSHOTS






REGISTRY
Register Service
Find Service
Modify Service
Delete Service
Service Composition
Find Services
Service Key
Service Name
data aggregation
Service Description
Service URL
Submit
Reset


REGISTRY
Register Service
Find Service
Modify Service
Delete Service
Service Composition
Service Key%,Service Name=data aggregation%,Service Description=%,Service URL=%
Service Key
Service Name
Service Description
Service URL
2ZYOYV5DL0VCFBA0I199TGD05401SJ
data aggregation
to perform in network aggregation
service instance1
75V7B4W664GTS92N3HLB584SDLWSSO
data aggregation
service instance2
3X43945642R1H0OXAA4B39TO2S20BF
data aggregation
service instance3

REFERENCES

[1] Akyildiz et al., "Wireless Sensor Networks: A Survey," Computer Networks, vol. 38, No. 4, Pages 393–422, 2002.

[2] A. Rezgui and M. Eltoweissy, "Service-Oriented Sensor-Actuator Networks," in IEEE Communications, Volume 45, No. 12, pages 92-100, December 2007.

[3] Xiumin Wang, Jianping Wang, Zeyu Zheng, Yinlong Xu, Mei Yang, " Service Composition in Service Oriented Wireless sensor Networks with Persistent Queries", Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference 2009.

[4] Manish Jain, Puneet Sharma, Sujata Banrjee , "QoS-Guranteed Path Selection Algorithm for Service Composition", in IEEE communications, FEB 2006.

[5] X. Gu, K. Nahrstedt, R. Chang, C. Ward, "QoS-assured service composition in managed service overlay networks," in ICDCS 2003, pp.19-22, May 2003.

[6] HAN Song-qiao, ZHANG Shen-sheng, ZHAN Yong, CAO Jian, "A QoS Aware Service Composition Protocol in Mobile Ad Hoc Networks", 2008.

[7] W. Zang, Y. Yang, S. Tang, L. Fang, "QoS-driven Service Selection Optimization Model and algorithms for Composite Web Services," 31st Annual International Computer Software and Applications Conference IEEE 2007.

[8] M. M. Akbar, M. S. Rahman, M. Kaykobad, et al. "Solving the multidimensional multiple-choice Knapsack Problem by constructing convex hulls," Computers and Operations Research, 2006, vol. 33, pp.1259-1273, 2006.

[9] Addelkader Sbihi, " A best first search exact algorithm for the multiple-choice multidimensional knapsack problem," Journal of combinatorial optimization, Springer Link, Volume 13, No. 4, pages 337-351, May 2007.