Building Wireless Sensor Networks Application Using Sun SPOTs

Asadullah Shaikh[1,2], Naveed Anjum[3], Fahad Samad[2], and Asadullah Shah[1]

[1] Kulliyyah of Information and Communication Technology,
International Islamic University, Malaysia
asadullah.shaikh@live.iium.edu.my, asadullah@kict.iium.edu.my
[2] Institute of Business and Technology, Pakistan
{asadullah,fahad.samad}@biztekian.com
[3] University of Southern Denmark, Denmark
yoursanjum@gmail.com

Abstract. Wireless Sensor and Mobile ad-hoc Networks are characterized by their highly dynamic, multi-hop, and infrastructure-less nature. In this dynamic environment, different nodes offering different services may enter and leave the network at any time. Efficient and timely service discovery is a requirement for good utilization of shared resources in this kind of network. Service discovery is not a new problem. Many academic and industrial researchers have proposed numerous solutions (protocol/middleware) of service discovery for both wired and wireless networks. The wireless sensor networks have several characteristics that set them apart from the traditional wired networks. Hence, classical service discovery protocols in sensor networks are not applicable. This paper presents a service oriented architecture for autonomous wireless sensor networks that provide various services at different levels.

1 Introduction

During the past decade, the speed and reliability of communication over wireless network has increased rapidly. Due to this increase, distributed computing is now seen as an adaptable and economical way to enhance computing power and share information. Recent advances in both the hardware and the software have allowed for the emergence of purely distributed computing, in which each entity can be either client or server. One area of great interest in distributed systems is wireless ad-hoc network that allows collaboration in real time. Wireless ad-hoc networks are formed by a set of hosts that communicate with each other over a wireless channel. Each node has the ability to communicate directly with another node (or several of them) in its physical neighborhood. They may operate in a self-organized and decentralized manner. This allows to develop a dynamic network infrastructure which makes these networks extremely

B.S. Chowdhry et al. (Eds.): IMTIC 2012, CCIS 281, pp. 466–477, 2012.

cost effective. Ad-hoc wireless networks can be used for many applications such as the sensor-based system and the systems that are dynamically deployed for sudden incidents. Wireless sensor networks consisting of individual sensor nodes distributed over a given area are used to monitor some physical phenomenon (temperature, humidity) in the environment. The importance of such networks is increasing rapidly with advances in technology that result in smaller, cheaper, and power-efficient devices. As a result, wireless sensor networks (WSN) have become increasingly common in everyday applications. A fixed network structure was required in the past for computers to collaborate. However due to advances in wireless computing, network structures can now exercise significant flexibility. Still, the limited capability in terms of memory capacity and processor speed imposes difficulties upon wireless networking. Wireless sensor networks and its applications have many uses in everyday human life ranging from health care and logistics, through agriculture, forestry, civil and construction engineering, to surveillance and military applications. The sensors applications are growing ever larger in size and complexity. As sensor technology improves and the demand for new applications grows, most of the currently deployed sensor network systems will need to evolve over time. Currently, most of the sensor network applications are being developed in the low level languages such as nesC and C at the system level. This traditional low level approach cannot support to develop larger, scalable, and flexible sensor applications.Higher level software methodologies are needed to meet the scalability, flexibility, and portability requirements of modern sensor network systems. The computer networks are basically about using services; multiple users (clients) regardless of their location may use a single service. Hence, the networks connect the users and services. In distributed systems, clients use static service locations to discover services before using them. Moreover, manually configured network addresses pointing to services are sufficient. Wireless Sensor and Mobile ad-hoc Networks are characterized by their highly dynamic, multi-hop, and infrastructure-less nature. In this dynamic environment, different nodes, offering different services, may enter and leave the network at any time. Efficient and timely service discovery [1, 2, 3] is a requirement for good utilization of shared resources in these kind of networks.

The remainder of this paper focuses on a service oriented architecture for autonomous wireless sensor networks that provide various services at different levels. Section 2 describes the problem addressed in this paper. Section 3 focuses on the architectural design along with proposed solution while Section 4 explores the implementation environment. In Section 5, related work is presented. Finally, Section 6 provides the conclusion.

1.1 Problem Statement

Service discovery is not a new problem. Many academic and industrial researchers have proposed numerous solutions (protocol/middleware) of service discovery for both wired and wireless networks. The traditional view of service discovery within local area network is different from the scenarios in wireless ad-hoc networks. The wireless ad-hoc networks have several characteristics, as described

above, that set them apart from the traditional wired networks. Due to these differences, the classical service discovery protocols are not directly applicable to wireless networks.

1.2 Aims and Objectives

The aim of this paper is to present a service oriented architecture [4] which will handle autonomous sensors networks in which all entities of a network collaborate with each other through a suitable protocol. Furthermore, we design and implement a service discovery prototype for wireless sensor networks which allows sensors to take part in the service oriented environment where all nodes at work offer different services at different levels (different attributes). The design goal can be divided into following:

- Request/Reply base service discovery architecture which will handle autonomous sensors network in which entities/nodes (Sun SPOT [5]) will leave or join at any time.
- A protocol which allows Plug and Play sensor node (Sensor Integration)
- A scalable and dynamic wireless sensor network architecture which allows autonomous system administration of sensor nodes and easy sensor data flow management with nodes (Discovery/Routing).
- An architecture in which sensor nodes act as clients or service provider (services) or both.

2 Problem Evaluation

Nowadays there are many uses for wireless sensor networks in almost every field. As the sensor devices become cheaper and more powerful, we can now expect the number of possible applications for wireless sensor networks to grow. The wireless sensor systems need to evolve over time to support new needs and functionality due to changes in requirements or improvements in sensor technology. Throughout all these changes, the sensor network must maintain an acceptable level of operation, since in some cases human lives may depend on it. Wireless sensor networks must be reliable, scalable, and flexible to change. It is interesting that still the sensor applications are mainly written at a low level, which means at or near the system layer, where providing a scalable, flexible framework is extremely difficult. Low-level application design tends to overlook proper modularization, making development of complex application code very difficult. These types of applications also tend to bind the system functionality to the current set of resources. When resources or requirements change, engineers must spend extra time, man-power, and money to keep the system up-to-date. Given the wide spread use of traditional networking technologies, it may be surprising how challenging even simple tasks become in the domain of the wireless sensors networks. Nowadays, an application level developer expects certain services to be provided by the operating system or by network protocols. However, due to

the resource constraints of sensor nodes neither of these can be taken for guaranteed. Problems arise mainly from developing applications comparatively closer to the hardware on one side and the need for distributed algorithms in wireless sensor networks on the other. Implementing the service discovery architecture as an abstraction layer is widely accepted as a solution to this problem. The key to success is to provide the application level developer with a programming abstraction that empowers him to directly formulate the crucial parts of his application in a way that naturally maps to the wireless sensor network platform. In general, the service discovery in wireless ad-hoc environment is a difficult task. Nevertheless, it is a crucial feature for the usability of wireless ad-hoc networks. The service discovery allows devices to automatically locate network services with their attributes and to advertise their own capabilities to the rest of the network. To enable service discovery within wireless ad-hoc networks we face many challenges like enabling the resource-constrained wireless devices to discover services dynamically and enabling the service discovery in large wireless ad-hoc networks. In the simplest case, the service discovery mechanism needs to work without any configuration, management or administration. There are different scenarios where the service discovery protocol is being used in the wireless network environment. We would like to include a discussion on a few of these scenarios.

- Scenario 1: Imagine that you find yourself in a taxi cab without your wallet. Fortunately, you have a JINI technology enabled mobile phone, and your mobile service provider uses JINI technology to deliver network-based services tailored to your community. On your phone screen, you see a service for the City Taxi Cab Company, so you download the electronic payment application to authorize payment of your taxi cab fare. The cab company's payment system instantly recognizes the transaction and sends a receipt to the printer in the taxi. You take the receipt and you are on your way.
- Scenario 2: Consider an intelligent, on-line overhead projector with a library client. After identification to the system, the user may select a set of electronically stored charts or other document(s) for viewing. Rather than bringing foils to a meeting, the user accesses them through the LAN server in the library.
- Scenario 3: Consider an insurance salesman who visits a client's office. He wants to brief new products and their options to the client which are stored in his Windows CE Handheld PC. Since his Handheld PC has wireless network and supports UPnP, it automatically discovers and uses an Ethernet connected printer there without any network configuration and setup. He can print whatever in his Handheld PC or from computers in his main office and promote the new products.

Scenario 1 is a JINI demo scenario developed by Sun Microsystems and scenario 2 is a Salutation scenario by IBM. The last one is a UPnP scenario by Microsoft. At a glance, they seem to talk about the same stories: mobile devices, zero-configuration [6] and impromptu community enabled by service discovery protocols, and cooperation of the proximity network. Even though they work

in the same way, these three service discovery protocols have different origins, underlying technologies, flavors, and audiences. Since they see the problem at different angles and take different approaches to it. It is a fact that not all the service discovery architectures (protocols) provide the optimal solution. They all have some advantages and disadvantages. Especially for the ad hoc wireless infrastructure, it is very hard to come with the optimal solution due to the dynamic nature of the nodes. There are some challenges and requirements which hinder the development of service discovery architecture.

3 Architectural Design

We have concluded some capabilities of Sun SPOT on which we have developed our protocol. Every Sun SPOT can act as a mesh router which means that it forwards (relay) the packets toward the other Sun SPOT(s), which it receive. The Sun SPOT(s) use the AODV algorithm to determine the best route when the communications between them are routed over more than one hop. To maintain the route, every Sun SPOT use routing policies which can be enabled through the RoutingPolicyManager. The Sun SPOT(s) can also have the capability to broadcast the datagram packet. So we have approached our architecture design with higher-level programming methodologies on application layer with support from network layer's (AODV protocol) capabilities of Sun SPOT. We sum up with a directory-less architecture which is a suitable architecture for the resources constrained devices such as sensor nodes. To enable reliable service allocation, we apply the JINI lease mechanism in our architecture according to which a service is requested for a time period and then granted for negotiated period between the service user and the provider. In the following section, we propose our service discovery architecture.

3.1 Proposed Solution

We have proposed a directory-less architecture which is based on the pull-model (query the environment when service is required) service discovery approach. In our architecture, each participating node has the capabilities to store its own local services, deliver these services to other nodes, query the network for available services offered by others, and use the services it discovers in the network. The discovery process (we will call it service lookup process) is simple as the client node (requester) broadcasts the request to search for the service and waits a reply from any service provider. If a provider node (service provider) is found, then the provider issues a reply back to the client. The reply includes the id of the service provider, which is used for subsequent communication between the client and the provider. In case the service provider is not found, no reply is received by the client. The underlying network layer is responsible for routing the information within the network. The nodes in the network automatically transmit any received packets so that it is relayed to the entire network, including the nodes that are out of range of the original requester. Note that there are maximum

preset numbers of nodes which are traversed (time to live-TTL). When a node receives a request from the client during the service lookup phase, it checks the availability of the requested service within itself. Depending on the availability of service, the node issues a reply to the client. After a successful service lookup phase, further communication between the client and the provider is initiated by the client. It sends the lease and service binding request to the provider for the requested service. The lease request contains the provider's id so the node which receives the lease request matches the provider id with its own id and takes the necessary action in the form of lease reply. The provider sends the lease reply along with the service binding to the client. After the successful service binding phase, the client will have the service until the lease time expires. When the lease expires, the current communication session will end. Because the nodes are mobile, it is also likely that many link breakages along a route occur during the lifetime of that route. The underlying network protocol is also responsible to maintain a route for as long as the communication is active.

4 Implementation Environment

The prototype implementation is done in Java language. There are two reasons to select Java. First, we have used the Sun SPOT as sensor device and it is a Java embedded platform, more specifically Connected Limited Device Configuration (CDLC) of Java Micro Edition (J2ME) platform with the MIDP-1.0 (Mobile Information Device Profile) profile. Second, the Java language provides the feature like modularity, reusability, extendibility, and robustness for developing applications. We have used the NetBeans IDE as the development environment, since the NetBeans IDE is the default environment for developing the applications for the Sun SPOTs. Indeed, the choice of IDE is arbitrary as long as it supports ANT scripts. We utilize UML notations to explain the implementation of the architecture. The necessary implementation is discussed further in this section. All the experiments are done using the Sun SPOTs and having them all in the same room, placed a few feet away from each other. The behavior could be different in different environments. The implementation consists of the following Java packages:

- The XML parser org.xmlpull.v1 and org.kxml2.io packages contain the XML parser [...] which we used to read the XML files.
- The service org.sunspotworld.service package contains the classes which model services used in the system.
- The service framework org.sunspotworld package implementations of the main service discovery architecture and it functionality.

As we have used the pre-developed XML parser, we will not provide the detailed implementation of it.

4.1 Services

Service is the abstract base class for all the services. It defines the basic functionality of the service. Any service which needs to be activated must inherit from

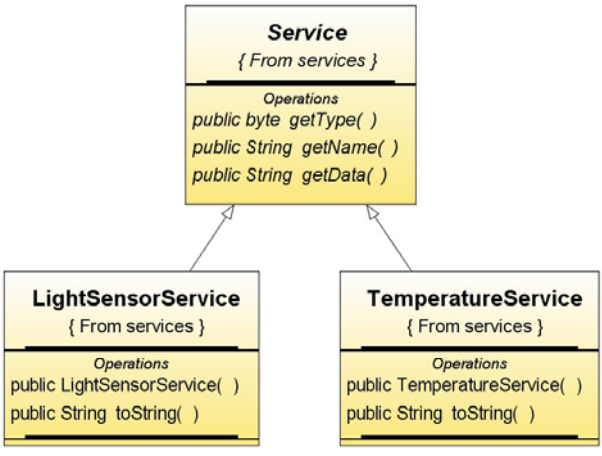

Fig. 1. UML Class Diagram of Service

this class. As shown in the UML class diagram in figure 1, the LightSensorService and TemperatureService classes are inherited from Service. These classes implement the actual functionality of the service e.g. activation and collection of data etc.

4.2 Framework Overview

Here is a brief overview of all the classes which implement the Service framework:

- StartServiceMain is the Midlet class and it starts the application.
- SpotParticipant models the spot itself.
- ServiceReader reads the service types defined in the XML file.
- ServiceLoader takes the service type codes and instantiates Service objects for each of them.
- ServiceRequest is a wrapper around the actual Service defining some additional info (e.g. lease time, interval).
- Packet wraps the data which we are sending between the spots. It is the message (packet).
- PacketTypes is an interface which just defines the different types of packets. We have five types of packet defined as:
  - LOOKUP_PACKET is sent by a client when a new service is required.
  - LOOKUP_REPLY_PACKET is sent by the service provider as a reply of LOOK_UP_PACKET packet.
  - LEASE_REQUEST_PACKET is sent by the client as a reply of LOOKUP_REPLY_PACKET asking for the service data and lease time for that service.
  - LEASE_REPLY_PACKET is sent by the service provider with service data and for a particular lease time.

  - LEASE_EXPIRED_PACKET sends by the service provider when the lease time is expired.
- PacketSender is responsible for the sending of packets.
- PacketReciever is responsible for the receiving of packets.
- Persistent is an interface which is implemented by every class which is sent between the spots (in our case the class Packet) and has methods to persist and load an object.
- PacketHandler performs the necessary action based upon the message (packet) that spot has received. It is responsible for processing the messages and acting upon them (e.g. sending a reply). It contains two inner classes:
  - ServiceHandler starts a separate thread where it sends data to the spot which has requested the service. It terminates when the lease time expires.
  - TimeoutTimer is used to determine if a connection to a spot has been lost.

4.3 Working Procedure

When we start the application, it runs both at the service provider and the service requester which means that a Sun SPOT can both offer and request services. The application has two concurrent behaviors as Service Provider and Service Requester. The UML class diagram in figure 2 shows the structure of the classes.

Service Provider. The class SpotParticipant has the objects PacketReciever and PacketSender. It uses the ServiceRequest and Service classes to implement the list of either the requested services or the offered services. The PacketReciver and PacketSender classes use to communicate with other Sun SPOT(s). These classes communicate by sending and receiving data wrapped in Packet object. Packets are handled in the SpotParticipant class using the PacketHandler class.

Service Requester (Client). As a service provider, the SpotParticipant models the Sun SPOT node as the service provider. After this, it reads the XML file for the offered services, initiate the services, and load them into internal list of offered services. The ServiceLoader class is responsible to load the services. It has a loadservice method which instantiates the service objects. Now service provider is ready and waits for any client for requested services.

Communication Protocol. This is a common functionality for the service requester (client) and the service provider. PacketSender and PacketReciever are responsible for interaction within the communication protocol. PacketReciever runs in a separate thread and is constantly listening for new messages. The PacketSender always broadcasts the packet along with the particular port while the PacketReciever uses the server connection to receive the packet. When a packet is received, the PacketReciever checks the receiver id of the packet and compares it to the id of the SpotParticipant which owns the receiver. If they are identical, it passes the packet to the SpotParticipant, otherwise it is dropped. If the sender id is an empty string (i.e. a receiver is not specified) the packet was meant as a broadcast message and it will be accepted by all the receivers.

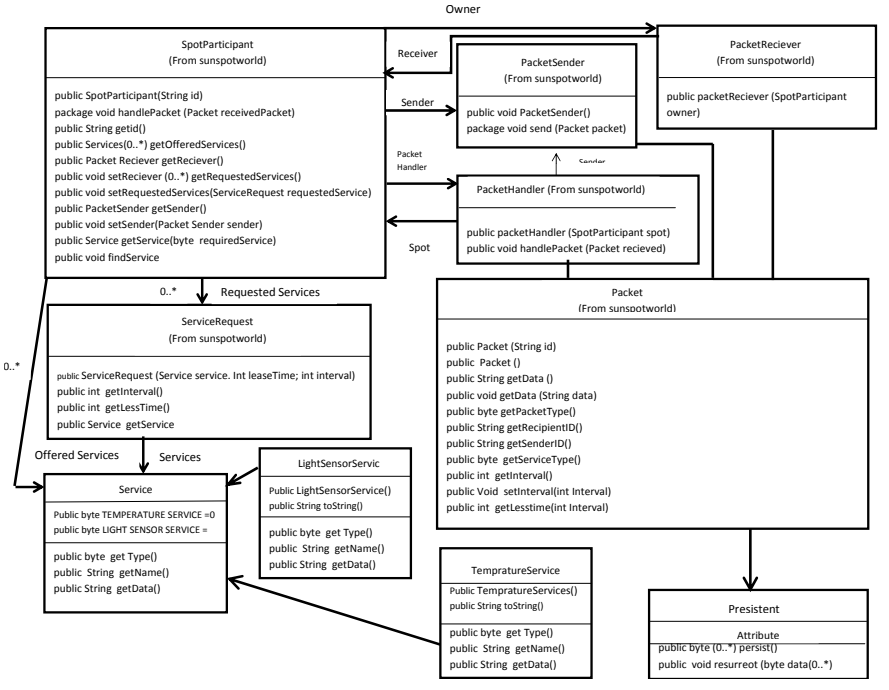

Fig. 2. UML Class Diagram of Service

Packet Handling. The packet handling is done by PacketHandler class. When any kind of packet arrives as defines by the PacketTypes interface, this class reads that packet and takes the necessary action based upon the information in that packet. For example, on the arrival of a lookup packet, it replies with a lookup reply packet. The lease is also handled by this class. It is handled by the ServiceHandler which is an inner class of PacketHandler. The ServiceHandler sends a lease expired packet to client when lease is expired. Additionally, the inner class TimeoutTimer of PacketHandler class is responsible to take care of the connection lost between the client and service provider. When the PacketHandler receives the LOOKUP_REPLY_PACKET packet, it starts a timer countdown and puts this communication session into the active services list of the client. The timeout time of the timer is set to the interval of the service request plus 10 seconds. Every time it receives a LEASE_REPLY_PACKET packet, it resets the timer. If the timer reaches 0 it is assumed that connection has been lost and the session is removed from the list of active services. It means that if it receives any data from that session after this, it will discard that data. Packet loss (i.e., the loss of LEASE_REPLY_PACKET packets) is also handled by the PacketHandler. The service provider sends a sequence number every time it sends a LEASE_REPLY_PACKET packet and the client checks that sequence number to see if any packets have been lost.

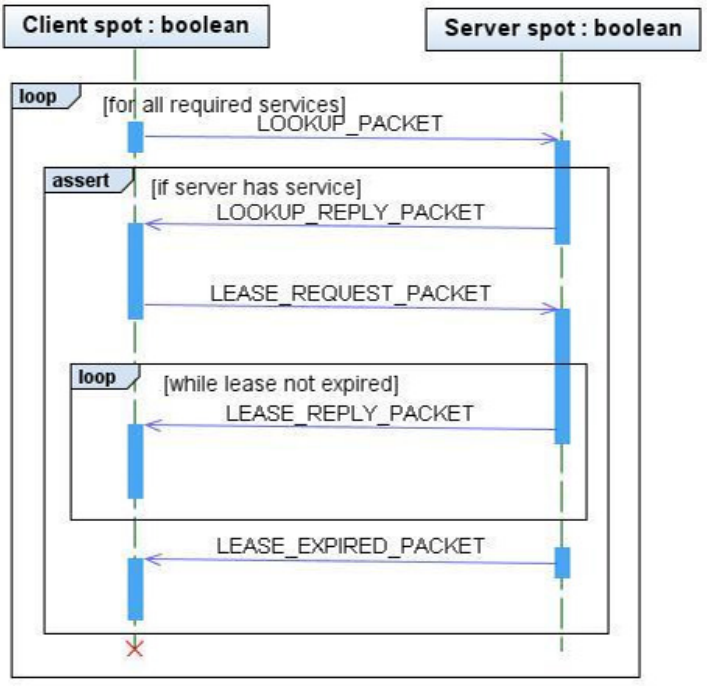

Fig. 3. Sequence Diagram

Communication Flow. To explain how the application works, we assume that a service provider is already running and a client needs the service. The client broadcasts the lookup packet which contains the packet type and service(s) and waits for reply from any service provider. When any running service provider receives this lookup packet, it checks the requested service(s) in offered service list. If it has the requested service(s), it replies back to client with the lookup reply packet with service(s) and its own address. When client gets this packet then it sends the lease request packet to the service provider in which the client asks for the service for the particular time period. As a reply, the service provider sends the requested services according to the lease time. When the lease expires, the service provider sends the lease expired packet to the client and stops sending the data. Figure 3 shows a sequence diagram (using UML notation) which illustrates the flow of communication between the client and the service provider.

In case of a connection loss between client and service provider, the client will wait for additional 10 seconds after the time interval set for the LEASE_REP LY_PACKET packet has passed. The lease (which is with the server) will still be active and the server will still send data over. The data will however be disregarded by the client if it should receive it at some later point.

Output. The output of the application can be obtained on the computer screen if we attach the client node with computer. But in case of field area where no computer is available, the output can be observed through the Sun SPOT(s) led lights. The blue led light moving from led 1 to led 8 demonstrates packet sending and red led light moving from led 1 to led 8 shows the packet receiving. The blinking green led light expresses the service offered in case of a service provider.

5 Related Work

As a whole, analyzing, evaluating, and categorizing service discovery solutions is a difficult task. Solutions are complex due to large number of responsibilities and diverse design approaches. This section serves as a standardized way of decomposing service discovery solutions and categorizes previous work in the area.

The existing service discovery architectures can be divided into two main categories: directory-less architecture and directory-based architecture. A directory is an entity that stores the information about services available in the network so as to enable service discovery and invocation. In the directory-less architecture, nodes do not distribute their service descriptions onto other nodes in the network. A device interested in a special service typically sends its search message to all reachable nodes. If one or more of these nodes can satisfy the request, a response is sent back to the requester.

Konark [7] is a middleware architecture which is designed specifically for the discovery and delivery of services in multi-hop ad hoc networks. It supports both "push and pull" models for the service discovery mechanism, with a cache in all devices. It defines an XML based service description language similar to Web Services Description Language (WSDL) with regard to service description. Konark uses multicast to advertise and discover services and allows for service delivery by running a lightweight HTTP server on every node that hosts services. Every node maintains a service registry, where it stores information about its own services and also about services that other nodes provide. This registry is actually a tree-structure with a number of levels that represent service classification.

PDP [8] is a distributed service discovery protocol designed for ad hoc networks. PDP takes into account inherent limitations of embedded devices such as power-constrained batteries and processing capabilities in such a way that the protocol reduces the number of messages sent through the network. PDP gives priority to the replies of the less-constrained devices by allowing the others to abort their answers. It also does away with the need of a central server and it is a fully distributed protocol that merges characteristics of both pull and push solutions. In PDP, devices maintain a cache of services previously announced, that is also used for the answers. All the messages are broadcast, and all devices cooperate by coordinating their replies and sharing the information in their caches. PDP takes into account different needs of the applications which allows to further reduce the power consumption.

6 Conclusion

Service discovery is not a new problem. Many academic and industrial researchers have proposed numerous solutions (protocol/middleware) of service discovery for both wired and wireless networks. We have proposed the Service Oriented Architecture (SOA) which will handle autonomous sensors networks in which all entities of a network collaborate with each other through a suitable protocol. Furthermore, we designed and implemented a service discovery prototype for wireless sensor networks which allows sensors to take part in the service oriented environment where all nodes of the network offer different services at different levels (different attributes).

References

1. Seok, O., et al.: An integrated approach for efficient routing and service discovery in mobile ad hoc networks. In: Second IEEE Consumer Communications and Networking Conference, CCNC 2005, pp. 184–189 (2005)
2. Lu, Y.: An Adaptive Middleware to Overcome Service Discovery Heterogeneity in Mobile Ad Hoc Environments. IEEE Distributed Systems Online, 1–1
3. Artail, H., Mershad, K.W., Hamze, H.: DSDM: A Distributed Service Discovery Model for Manets. IEEE Transactions on Parallel and Distributed Systems, 1224–1236
4. Bachara, P., Zielinski, K.: SOA-Compliant Programming Model for Intelligent Sensor Networks – SCA-Based Solution. In: 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 471–478 (2010)
5. Smith, R.B.: SPOTWorld and the Sun SPOT," Information Processing in Sensor Networks. In: 6th International Symposium on Information Processing in Sensor Networks, IPSN 2007, pp. 565–566, 25-27 (2007)
6. Zero Configuration Networking (Zeroconf) (last checked on January 07, 2012), `http://www.zeroconf.org/`
7. Helal, S., Desai, N., Verma, V., Lee, C.: Konark - A Service Discovery and Delivery Protocol for Ad-hoc Networks. In: Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans (2003)
8. Campo, C., Munoz, M., Perea, J.C., Martin, A., Garcia-Rubio, C.: PDP and GSDL: service discovery middleware to support spontaneous interactions in pervasive systems. In: Proceedings of the 3rd Int'l Conf. on Pervasive Computing and Communications Workshops (2005)