CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 BACKGROUND

2.1.1 Wireless Sensor Networks

A wireless sensor network as dealt by Jun Zheng and Abbas Jamalipour (2009) typically consists of a large number of low-cost, low-power, and multifunctional sensor nodes that are deployed in a region of interest. These sensor nodes are small in size, but are equipped with sensors, embedded microprocessors, and radio transceivers, and therefore have not only sensing capability, but also data processing and communicating capabilities. They communicate over a short distance via a wireless medium and collaborate to accomplish a common task, for example, environment monitoring, battlefield surveillance, and industrial process control.

A sensor node is made up of four basic components: a sensing unit, a processing unit, a transceiver unit and a power unit. They may also have application dependent additional components such as a location finding system, a power generator and a mobilizer.

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 sensor block is the focal block of the device. This block may contain one or more sensors, which may
be identical (for redundancy and/or multiple sources for quality of information estimation) or different. The sensor block is driven by the processor, which determines when the sensors should take a reading, according to the operational requirements programmed into the device.

At the center of the sensor device is a processor, which coordinates all other functions. The processors on sensor nodes have lately increased in capability, and can currently offer satisfactory data processing functions. Given the energy constraints of the devices, and the fact that processing requires less energy than communication, it is attractive to process data locally to reduce the amount of data needed to be transmitted. Typical examples of node level processing include averages over time, threshold based alarms, etc. The processing unit, generally associated with a small storage unit, manages the procedures that make the sensor node collaborate with the other nodes to carry out the assigned sensing tasks.

A transceiver unit connects the node to the network. The communication radio block is one of the distinctive components of the sensor node. Wireless communication is fundamental to the WSN concept. The communication radio is required to be duplex (i.e., able to transmit and receive), and must be supported by adequate link layer and network layer protocols to allow effective and efficient use of the physical medium and the deployment topology. Although many devices have appeared using proprietary radios and protocols, the most prominent options include the following standards: Bluetooth and ZigBee/IEEE 802.15.4

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. Power is obviously essential. However, the power sources for these devices are very limited. Given the deployment constraints, the devices may not be mains powered. Additionally, given size constraints, the batteries
cannot be large, and to keep operational expenditure costs down, battery replacement must be rare. This creates one of the main challenges of the WSN, which is the scarcity of energy and should be considered when developing any solution targeted for WSNs.

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. Thus, it is common that a sensor node has a location finding system. A mobilizer may sometimes be needed to move sensor nodes when it is required to carry out the assigned tasks. All of these subunits may need to fit into a matchbox-sized module. The required size may be smaller than even a cubic centimeter which is light enough to remain suspended in the air. Apart from the size, there are also some other stringent constraints for sensor nodes. These nodes must consume extremely low power, operate in high volumetric densities, have low production cost and be dispensable, be autonomous and operate unattended, and be adaptive to the environment.

Wirelessly networking sensor nodes together is one of their key capabilities, which supports the increase in sensing granularity and the low deployment costs. A WSN is made up of two types of devices: wireless sensor nodes and gateways. The wireless sensor node has already been described and is the core of the sensor network. A gateway device is however also necessary in order to connect the wireless sensor network to some external system, such as a wider network and to support the scalability of management operations, as well as the overall WSN reliability and survivability.

The sensors that, when distributed in the environment, comprise WSNs include cameras as vision sensors, microphones as audio sensors, and those capable of sensing ultrasound, infra-red, temperature, humidity, noise, pressure and vibration. Compared with traditional wireless communication
networks, for example, cellular systems and MANET, sensor networks have the following unique characteristics and constraints.

- **Dense node deployment:** Sensor nodes are usually densely deployed in a field of interest. The number of sensor nodes in a sensor network can be several orders of magnitude higher than that in a MANET.

- **Battery powered sensor nodes:** Sensor nodes are usually powered by battery. In most situations, they are deployed in a harsh or hostile environment, where it is very difficult or even impossible to change or recharge the batteries.

- **Severe energy, computation, and storage constraints:** Sensor nodes are highly limited in energy, computation, and storage capacities.

- **Self configurable:** Sensor nodes are usually randomly deployed without careful planning and engineering. Once deployed, sensor nodes have to autonomously configure themselves into a communication network.

- **Application specific:** Sensor networks are application specific. A network is usually designed and deployed for a specific application. The design requirements of a network change with its application.

- **Unreliable sensor nodes:** Sensor nodes are usually deployed in harsh or hostile environments and operate without attendance. They are prone to physical damages or failures.

- **Frequent topology change:** Network topology changes frequently due to node failure, damage, addition, energy depletion, or channel fading.
- No global identification: Due to the large number of sensor nodes, it is usually not possible to build a global addressing scheme for a sensor network. Because it would introduce a high overhead for the identification maintenance.

Wireless sensors have significant advantages over conventional wired sensors. They can not only reduce the cost and delay in deployment, but also be applied to any environment, especially those in which conventional wired sensor networks are impossible to be deployed, for example, inhospitable terrains, battlefields, outer space, or deep oceans. WSNs were originally motivated by military applications, which range from large-scale acoustic surveillance systems for ocean surveillance to small networks of unattended ground sensors for ground target detection. However, the availability of low-cost sensors and wireless communication has promised the development of a wide range of applications in both civilian and military fields.

One major reason for the increasing interest in wireless sensor networks in the last few years has been their potential usage in a wide range of applications. Yet today, most sensor networks function in isolated patches, each with different mechanisms to deliver data to their users, and often have no formal methods to share data with others. In short, the interest is in answering the following question: Given the huge number of devices monitoring the physical world already deployed and available for usage, how to enable people to create applications on top of such WSN systems? To achieve this goal, a crucial requirement is to provide a layer of abstraction to issue sensing tasks and queries to the WSN and to gather the sensor generated data. But current architectures for WSN have inherent limitations that may be summarized as follows.
• **Tight coupling between WSNs and applications:** A tightly coupled network application characterizes the most current sensor network and WSN architectures. This coupling usually takes one of two forms.

  • Network-dependent application development: Most current sensor applications are designed and implemented to be used on a specific sensor network or a specific type of sensor networks with specific characteristics and a specific querying interface. Often, application developers need to know network-specific information such as network topology, nodes’ transmission range and processing/memory capabilities, etc.

  • Application-dependent network design and deployment: In some cases, a decision is first made to use a specific software system to build a sensor application. The requirements of the software system must then be taken into consideration when designing and deploying the sensor network supporting that system.

• **Costly optimization or suboptimal efficiency:** In current WSNs, application dependent optimization makes the sensor network unable to provide the same levels of performance to other applications. Also, several rounds of optimization may be needed as the application evolves or is replaced. The alternative of application independent optimization is often too generic and does not exploit optimization opportunities that a specific class of applications may offer. Typically, this leads to suboptimal efficiency.

• **Limited reusability:** Ideally, for a WSN to be cost effective, it would be necessary to amortize its deployment and
maintenance cost by sharing its functionalities amongst a large group of users and applications. This reusability is generally not easily achievable in current sensor infrastructures due to the tight coupling between networks and applications.

- **Low return on investment:** This drawback follows from the previous one. The monolithic, application specific design of current WSN makes it difficult to reuse most of an application’s modules in developing another application. Often, intensive programming is required each time a new application has to be developed.

- **Non scalability:** Most of today’s WSN applications are designed and optimized for light loads, i.e., destined to be used by a small group of users. As a result, current WSNs often do not scale to support large numbers of simultaneous users.

Considering that WSNs can be potentially useful for a wide range of application domains and that the sensor network infrastructure is expensive, there is a strong trend towards designing commercial scale WSNs as being composed of heterogeneous sensor devices and assisting a large range of applications for different groups of users. Architectures for future sensor systems will have to be able to serve different applications and adapt to different post-deployment query patterns. Networks from different providers will have to be individually programmed and yet be able to interoperate efficiently. However, there is a set of challenges to be overcome in order to realize the true potential of current WSNs.

The first challenge concerns the effort needed to develop WSN applications. Currently, if an application programmer wants to develop a system for monitoring certain phenomena using a WSN, he may need to learn
a new language (e.g., NesC), a new programming paradigm (e.g., component-based programming), a new embedded operating system (e.g., TinyOS), and probably even some details about the underlying hardware platform. This situation is of course far from optimal, as an application programmer should only concentrate on application level issues and ideally use the programming languages and tools that he is accustomed to. Providing programmers with adequate abstractions and tools for developing applications that can incorporate access to the resources provided by WSN is an issue that has caught the attention of researchers.

The second main challenge regards the extraction and use of the sensor generated data. Most of existent WSNs provide data in proprietary formats that can be accessed by final user only through a set of static predefined queries or a graphical interface. Such approach constrains the use of the data reported and queries, hindering the widespread use of WSN data in different applications.

The third challenge is about network heterogeneity. Many research projects assume that the sensors are homogeneous but the network formation is ad hoc. They focus on providing basic network services in the system, such as time synchronization, node localization, topology control, and security. But in sensor rich environments, many kinds of sensors may be deployed in the same space, possibly by different organizations. Wireless sensor networks are becoming increasingly heterogeneous due to two primary reasons. First, heterogeneity allows a network to be both computationally powerful and deployed in high densities. Powerful devices can perform complex operations, but are more expensive and power-hungry. Conversely, weak WSN devices enable higher deployment densities and increase network lifetime as they are cheaper and consume less power. By integrating devices with different resources and capabilities, a heterogeneous WSN can combine the advantages of both powerful and weak devices. Second, network heterogeneity follows
from the natural evolution of WSNs. WSN devices can be embedded in the environment and remain operational for a long time. For example, due to its high deployment cost, a WSN embedded in civil infrastructure for structural health monitoring must operate over several years to be economically acceptable. Similarly, many urban sensing systems must also remain operational for multiple years. During the lifetime of a WSN, new devices may be developed and deployed, resulting in network heterogeneity.

Network heterogeneity presents a formidable problem for application developers. Since the target platform may consist of many different devices, the application must be platform independent to avoid having to custom tailor it to each device. Yet, the application must still be able to access platform specific capabilities like sensing and computing to make full use of the underlying hardware. Furthermore, the application must accommodate diverse device capabilities and resources.

Since service oriented computing discussed by Papazoglou and Van Den Heuvel (2007) allows flexible software compositions using loosely coupled services with applications, there have been increasing efforts for adopting a service oriented computing paradigm into wireless sensor networks. In a service oriented WSN application, each activity (sensing, aggregation, service discovery, etc.) is implemented as a separate service. Since these services are open and self-descriptive, various services from different vendors can be utilized together. Services may consist of multiple sensor nodes, and they are reusable and shared among multiple applications. Service oriented wireless sensor networks provide interoperability, scalability, and reusability of sensors.

2.1.2 Service Oriented Architecture

Service Oriented Architecture is an abstract concept in software engineering. The key components are services that are independent from each
other and interact on a well defined communication channel with each other. As discussed by Papazoglou (2003) there are several properties that services fulfill.

- **Platform independent interface**: Services can be accessed in a standards-based manner.

- **Self-contained**: Services are modular and provide their functionality independently of other services.

- **Loosely coupled**: A service is a “black box”, e.g. service consumers do not need to know about underlying technical internals of the service.

SOA is not tied to any specific technology but rather implies some driving forces according to the following.

- **Distribution**: Software components of the system run in different locations on a network. They need to communicate via a protocol.

- **Heterogeneity**: Different software entities may be implemented in different technologies. Integration must be possible without knowing detailed contexts.

- **Dynamics**: How the system is comprised may change at runtime and cannot be assumed statically.

- **Transparency**: As a result of heterogeneity and dynamics service providers and consumers are oblivious to implementation details of a service.

- **Process-orientation**: Services allow for composition in more coarse-grained workflows.
SOA is a perfect fit for complex systems that need to integrate various independent subsystems. In order to make use of the services they must be discoverable by service requesters and publishable by service providers. This is often accomplished by a service registry, where services can be looked up and registered. Figure 2.1 visualizes the interaction of these roles.

SOA defines three main roles service provider, service requestor and service registry and three main operations—publish, find (lookup) and bind (interact). Service provider is the owner and host of the service. It is responsible for creating a service description and publishing it to one or more registries, and receiving invocation messages from one or more service requestors. The registry is a repository of web service descriptions. Each description contains all the necessary information to use the described service. The requestor is client that is able to discover a web service from the registry and invoke it from its provider. The publish operation is what the service provider uses to make the service descriptions available in the registry. The find operation allows the service requestor to state its search criteria to the registry in determining the service of interest and how to invoke it. The service registry matches the find criteria against its collection of published descriptions. The bind operation is the actual invocation embodying a client-server relationship between the service requestor and the provider.

![Figure 2.1 SOA roles and their relationship](image)
One possible realization of SOA is the classical WS protocol stack as dealt by Papazoglou (2008). It is called WS because most existing standards in the protocol family have abbreviations that start with “WS”.

Alonso et al (2004) discussed that web services became the preferred technology for realizing the Service Oriented Computing (SOC) promise of easy information sharing and better software interoperability. This technology has been designed for creating highly autonomous services that can be used by different clients in different settings without deviating from internal policies. Web services can be defined as “loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard Internet protocols.” This definition reflects main conceptual characteristics of services and points out specific properties of web services. Firstly, it says that web services are reusable software components, that is, they allow developers to reuse the building blocks of code created by others. Secondly, these software components are loosely coupled, that is, they require a much simpler level of coordination and allow for easier replacement with another component. Thirdly, web services semantically encapsulate discrete functionality, which means that they are self-contained and describe their own inputs and outputs in a way that other software can determine what they do, how to invoke their functionality, and what result to expect in return. Fourthly, web services are designed to be accessed programmatically, that is, to be called by and exchange data with other software. Finally, web services are distributed over the Internet and can be accessed using ubiquitous transport protocols like HTTP and thus leverage existing infrastructure.

World Wide Web Consortium’s (W3C) definition of a web service is “A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-
processable format. Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web related standards”. Curbera et al (2002) stated that in web services, there are three core standards that are significant: SOAP, WSDL and UDDI. All of them make heavy use of XML as a basic expression format. Figure 2.2 shows how these standards play out in the roles of SOA. Brief description about these protocols is given below.

**XML:** The eXtensible Markup Language (http://www.w3.org/TR/REC-xml) is a data format or more generically a way to define data formats. It has consistent and clean text tagging, it separates content from format and allows hierarchical data structures. It also has facilities for user-definable data structures, which is a central feature needed by web services.

**HTTP:** The Hypertext Transfer Protocol is the standard application layer protocol to exchange hypermedia and other resources on the web. It is designed for client server style request response communication patterns and it is stateless, which means that every request response interaction is independent from any other. HTTP is a light weight protocol and is widely used and implemented on many systems.

**SOAP:** The Simple Object Access Protocol (http://www.w3.org/TR/SOAP ) is a standard to issue remote procedure calls and send/receive messages over the Internet. Commonly it uses HTTP as underlying transport protocol, but can be used with others as well. Messages are encoded with XML and consist of an envelope for namespace definitions, an optional header for additional information (e.g. security, addressing etc.) and a body containing the message data itself, i.e. service operations and their arguments. There are two types of messages: service requesters send SOAP requests and service providers send back SOAP responses.
**WSDL:** The Web Service Description Language (http://www.w3.org/TR/2001/NOTE-wsdl-20010315) is an XML vocabulary to specify metadata for web services like where and how clients can invoke the service and what operations and arguments are available. WSDL is extensible and is designed as a machine readable format, so that service consumer agents can pick up the necessary information about the service automatically. Currently WSDL 1.1 is the dominant version that is widely accepted. An alternative to WSDL is the Web Application Description Language (WADL), also an XML based description standard but intended specifically for RESTful web services.

**UDDI:** Universal Description Discovery Integration implements the service registry in the SOA model. As discussed by Luc Clement et al (2004) it allows service providers to publish their service descriptions (i.e. WSDL) and service consumers to lookup and locate web services they need. UDDI specifies the API to interact with such a registry via SOAP messages. Instead, there are business specific or internal registries in use, or other channels to exchange web service metadata information are implemented.

![Figure 2.2 Web service standards and their relationship in SOA](image-url)
2.1.3 Service Oriented Wireless Sensor Networks

Service oriented architecture, the proven technology in the information processing, can be very well considered for design and implementation of an efficient sensor network. Since service oriented computing allows flexible software composition using loosely coupled services with applications, there have been increasing efforts for adopting a service oriented computing paradigm into wireless sensor networks.

Service oriented architecture is the ideal architecture to ease the programming and to integrate WSNs. This is highly suitable for enabling applications to handle network heterogeneity. Ibbotson et al (2010) stated that in Service Oriented Wireless Sensor Networks sensor nodes act as data providers and the WSN act as a service provider for client application, which is service requester. Each service is a process that consumes input and produces output. In WSN service may be (i) raw data generated by the sensor nodes, (ii) processed data, generated through several types of analysis, filtering and complex processing, and (iii) value added services. For example, set of services provided by the sensors include human detection, vehicle detection, speed calculation, certain chemical element detections, picture taking etc. Each node in service oriented wireless sensor network exposes its capabilities as services. Conceptually, a service is a computational component that (i) has a unique network wide identifier, (ii) may be invoked asynchronously, (iii) may have one or more parameters and (iv) produces one or more values as a result of the invocation. In SOWSN services are modular, autonomous and have well defined interface that allow them to be described, published, discovered and invoked over a network. Services are open and self descriptive. This allows systems from different vendors to work together. A service may have multiple service instances, each running on a given node.
Specifically, the set of all WSN node capabilities can be divided into those that are platform independent (i.e., common to all platforms) and platform dependent (i.e., unique to specific platforms). Applications can be written using only platform independent capabilities, platform dependent capabilities can be exposed as services. When an application needs to access a platform specific capability, it is dynamically bound to a service that provides the capability. By enabling applications to be platform independent, application development is simplified. The notion of services is adopted in WSN primarily for their interoperability, scalability and retaskability.

By implementing a service oriented approach at all levels of WSN, the rapid development of applications as well as the thorough testing of sensor networks will be possible. It is a good way to integrate WSN with Internet. Also this service oriented approach will allow for the development of a WSN management system that will be able to handle the dynamic addition and removal of different sensors, actors and applications as interoperable services.

### 2.1.3.1 Advantages of service oriented architectures

(i) **Platform and Programming Language Independence**: Since services can be published and consumed across development and operating platforms, an application can leverage existing legacy components that reside on different types of servers and were built using different technologies. The main challenge is to integrate heterogeneous components into a larger application. This applies for wireless sensor networks if standardized protocols for communication between services exist.

(ii) **Focused Developer Roles**: Since a service is a discrete implementation independent of other services, developers in
charge of a service can focus completely on implementing and maintaining that services without having to worry about other services as long as the pre-defined contract is honored. However, the semantics and orchestrability of such a service must be made explicit and must be documented and managed accordingly.

(iii) Location Transparency: Services are often published to a directory where consumers can look them up. The advantage of this approach is that the service can change its location where it is executed any time. Consumers of the service will be able to locate the service through the directory. The underlying traditional assumption that communication is basically for free does not hold for WSN. Communication costs energy.

(iv) Code Reuse: Since SOA breaks down an application into small independent pieces of functionality, the services can be reused in multiple applications, thereby bringing down the cost of development. This demands that the principle of stateless business logic encapsulated into a small and precisely defined service is honored at all times. TinyOS with its concept of components and interfaces provides this kind of feature locally. Distributing functionality can save memory, provided, energy consumption is taken into account carefully.

(v) Greater Testability: Small, independent services are easier to test and debug than monolithic applications. This leads to more reliable software. In WSN this benefit cannot be leveraged in general, since distributed functionality is severely restricted by the unreliable wireless communication.
(vi) Parallel Development: Since the services are independent of each other and contracts between services are pre-defined, the services can be developed in parallel - this shortens the software development life cycle considerably. This holds for services in WSN as well.

(vii) Better Scalability: In principle loosely coupled and distributed systems do scale better. There can be multiple instances of a service running on different servers. This increases scalability. However, the impact of communication, load balancing, and management may interfere with scalability. This is particularly important for WSN with constrained resources.

(viii) Higher Availability: The same argument as above applies to a possible high availability. Services in industrial WSN are usually location based. This limits the possibility for redundant designs and thus higher availability may not be given.

(ix) Dynamic Deployment: New services can be deployed on demand or existing ones can be updated. In WSN this is accompanied by high communication costs and should thus be the exception rather than the rule.

2.1.4 Service Composition

The basic service oriented architecture suffices to implement simple interactions between a consumer and a provider. If the implementation of a service needs the invocation of other services, it is necessary to combine the functionality of several services. In this case, a need for composite service arises. The process of developing a composite service in turn is called service composition.
Kalasapur et al (2007) defined service composition as the process of rapidly combining and linking existing reusable services (atomic or composite) to create new richer working services. It constitutes an essential part of service provisioning since it leads to novel service offering thus adding value that was not existent in the individual services.

This ability of composing service using multiple services allows meeting larger and single user requirements that could not otherwise be met with any of the available smaller services. Thus complex service based applications can be created by composing individual services.

2.1.4.1 Phases in service composition

Traditionally, the task of automatic service composition has been split into four phases: (i) planning (service graph generation) and discovery (ii) selection and binding (iii) execution. The first phase involves generating a plan, i.e., all the services and the order in which they are to be composed in order to obtain the composition. The plan may be generated either statically or dynamically. The second phase involves discovering services as per the plan. Depending on the approach, often planning and discovery are combined into one step. After all the appropriate services are discovered, the selection phase involves selecting the optimal solution based on non-functional properties like QoS properties. The last phase involves executing the services as per the plan and in case any of them are not available, an alternate solution has to be used.

(i) Creation of process model

One of the significant characteristic of a service composition strategy is the degree of automation in the creation of the process model. Traditional service composition methods require the user to define the data flow and the control flow of a composite service manually, either directly or
by means of designer tools, e.g. in a drag-and-drop fashion. Subsequently the
process description is deployed in a process execution engine. Depending on
the abstraction level provided by the tools and also on the applied binding
mechanism, the user either creates the process model based on concrete
service descriptions or based on abstract service templates which are
representatives for sets of services, i.e. for service classes.

More advanced composition strategies actively support the user
with the creation of the process model, which is often referred to as semi-
automated service composition. Corresponding modeling tools may interact
with a broker in order to automatically look up services which match
(regarding inputs and outputs) with the already available control and data
flow, thus facilitating and accelerating the creation of the process model.
Fully automated composition approaches intend to generate a service
composition plan without human interaction. Mostly artificial intelligence
inspired methods based on formal logic are used.

(ii) Selection and binding

The selection of the activities which will participate in a service
composition may be done either at design time or at run-time. In the former,
the bindings are static - that is each instantiation of the composite service will
be made up of the same constituent services. In the latter, the constituent
services are selected at run-time, based on automatically analyzable criteria,
such as service functionality, inputs they need, outputs they produce and QoS
parameters. Late binding implies the dynamic invocation of the constituent
services - that is a sufficient level of interoperability has to be established,
either through fixed interfaces or by applying more sophisticated
matchmaking and mapping mechanisms. For a service provider, the applied
binding mechanism has several implications.
(iii) Execution

When composing services, two different execution models are usually applied: Orchestration and Choreography. In an orchestration all interactions that are part of a business process (including the sequence of activities, conditional events, among others) must be described, like on a traditional workflow system. This description is then executed by an orchestration engine, which has control of the overall composition. On the other hand, choreography is more collaborative and less centralized in nature. Differently from orchestration, there is not an entity that has a global control of the composition.

2.1.4.2 Classification of service composition strategies

Service composition can be classified based on two service composition characteristics, namely the type of service discovery/selection/binding and the degree of automation applied for the creation of a process model: service composition approaches may use early binding or late binding; the process model can be created manually, semi-automatically or automatically. As illustrated in the Figure 2.3, these characteristic values can be used for a classification of existing service composition strategies in six main categories. The fact that the borders between these categories are not strict but fluent is made clear through the smooth transitions between the squares. Some categories may overlap, i.e. there are composition approaches that may be assigned to two or more categories. To give an example: besides early and late binding there may be several variations in between, such as the specification of a restricted set of service candidates at design time from which one service is chosen and invoked at run-time.
In WSN the availability of services changes dynamically, because of nodes movement and resource availability. Hence, static composition is not sufficient. Also nodes could typically be very resource constrained and the resource situation may change rapidly, e.g. battery power may fade away. Therefore, resource requirements have to be considered during a composition process. Schahram Dustdar and Wolfgang Schreiner (2005) stated that automatic service composition means to replace the human expert by a software application that will create/compose the new service. Automatic service composition has the potential to dramatically change the way SOAs are engineered: Given a rich set of base services and efficient, reliable automatic service composition methods, the vision of programming as specifying what a program is supposed to do and not how it is supposed to do it can become a reality. That way software engineers could create flexible programs, able to adapt to changes in the environment completely autonomously.

2.1.5 Basic definitions

Definition 1: (Service): A service is the basic unit of application functionality. A service $S_i$ in a WSN is defined by the input data that it accepts, the transformation function that it applies to its input, the output data

![Figure 2.3 Classification of service composition strategies](image-url)
that it produces and metadata that characterizes the service and its output as given in equation (2.1)

\[ S_i = \{I_i, O_i, F_i, M_i(t)\} \]  

(2.1)

where \( I_i = I_{i1}, I_{i2}, \ldots I_{im} \) and \( O_i = O_{i1}, O_{i2}, \ldots O_{ik} \). \( S_i \) represents the name of a service, \( I_i \) represents the input set of the service, \( O_i \) represents output set of the service and \( F_i \) represents the function that transforms the input to the output. \( M_i \) represents the metadata associated with the service. Metadata contains QoS parameters of the service such as level of reliability, latency, energy consumption and so on. The service metadata depends on time.

**Definition 2: (Service Request):** A service request could be represented by the equation (2.2).

\[ SR_k = (I_k, O_k, QoS_k) \]  

(2.2)

where \( SR_k \) is the name of requested service, \( I_k \) represents the input set of the requested service, \( O_k \) represents the output set of the requested service, and \( QoS_k \) represents the minimum QoS requirements of the requested service, or QoS constraints in other aspects.

**Definition 3: (Service Composition):** A service composition is a service serial, \( S_1, S_2, \ldots, S_n \), which can satisfy some service request \( SR_k \). The serial satisfies the following four requirements.

(i). For \( SR_k (I_k, O_k) \) and \( S_1(I_1, 0_1) \), \( I_1 \subseteq I_k \).

(ii). For services \( S_i(I_i, O_i) \) and \( S_{i+1}(I_{i+1}, 0_{i+1}) \), \( O_i \subseteq I_{i+1} \). \( S_i \) is called as the precedence of \( S_{i+1} \), and \( S_{i+1} \) is called as the successor of \( S_i \).

(iii). For \( SR_k(I_k, O_k) \) and \( S_n(I_n, 0_n) \), \( 0_n \subseteq O_k \).

(iv). The QoS of every service in serial \( S_1, S_2, \ldots, S_n \) satisfy the requirement of QoS_k.
Definition 4: (Abstract service): Abstract service has function description without implementation and standard service interfaces across different service providers.

Definition 5: (Concrete service): Concrete services are service instances published by service providers. They could give the function implementation specified by abstract services. And some service instances may have the same function, but different QoS.

2.1.6 Automatic Service Composition

Automatic service composition (or dynamic, adaptive and even autonomic) remains one of the key challenges of service oriented computing today. In general, service composition can be defined as creating a composite service, obtained by combining available component services. It is used in situations where a client request cannot be satisfied by any single available service, but by a combination thereof. In terms of software engineering automatic service composition will significantly enhance the power of service oriented architectures. SOAs combine available base services in order to build higher level services or distributed applications.

2.1.6.1 Composition patterns

The composition patterns range from simple sequences of activities and loops to parallel or conditional executions with different semantics. Figure 2.4 visualizes the subset of composition patterns. These four composition patterns are usually sufficient to express common compositional logic.

Pattern 1: Sequence: It defines a sequential execution of n services in a composition. The length of a sequence has to be at least greater than one. The pattern is shown in Figure 2.4(a).
Pattern 2: Loop: A loop defines a repetitive execution of its body a given number of n times. The pattern is shown in Figure 2.4(b).

Pattern 3: AND split/AND join: An AND split specifies a parallel execution of n different branches. The AND join represents a barrier that waits for all n parallel executions to complete before continuing with the subsequent activities in the composition. The pattern is shown in Figure 2.4(c).

Pattern 4: XOR split/XOR join: An XOR split chooses one out of n branches (based on a condition) that is executed. The final XOR join waits for exactly one branch to finish before continuing with the execution of subsequent activities in the composition. It is typically used to model conditional execution (if-then) as known from traditional programming languages. The pattern is shown in Figure 2.4(d).

![Pattern Diagrams](image)

Figure 2.4 Composition patterns

2.2 RELATED WORK

2.2.1 Service Oriented Approaches for WSN

From the literature study, it was found that a large pool of examples on service oriented approaches for WSN is available. Several approaches are
currently being investigated at the research level while a few are actually being used in practice. The main issue to handle here is the diversity and heterogeneity of the sensors being used and the networks connecting them. Several issues were addressed by this category of approaches like handling complex and/or continuous queries, data access, aggregation, service publishing, service discovery, real time access to data and services, and limited resources. Following is a brief overview of selected approaches.

**IrisNet:** IrisNet dealt by Gibbons et al (2003) is middleware for globally distributed sensing systems. Sensors themselves are resourceful and capable of providing dense sensor data such as video streams. The project follows traditional two tier architecture, where sensor nodes are the leaves and servers are the core. IrisNet takes the design approach of building and querying wide area sensor databases. The system behavior is programmable in the sense that senselets are placed on sensor nodes, which can filter, store, process and analyze the collected data locally. IrisNet offers a programming interface where the service provider can easily add the new sensor service. Data are only transmitted when queried, and both query and response data are routed intelligently to the requesting user. This work deals with widely-deployed, resource-abundant, powered sensors like webcams and organizes them into a distributed database. The entire data-base is treated logically as a single XML document which is partitioned among multiple sites. The techniques proposed are effective for fragmenting and partitioning a database, routing and processing queries, and caching remote data locally. This work, however, does not consider streaming data and only works with a predefined database, i.e. all the sensors are assumed to be owned by the same organization.

**Service oriented software architecture:** One of the earlier works in taking the service oriented approach in the design of a middleware system for wireless sensor networks was presented by Golatowski et al (2003). In their
work they introduce a simple service oriented model in which the responsibility for handling the services requests is assigned to an external entity. This external server works as a bridge between the requests received from the exterior and the internal network functionality.

**Web service approach:** Delicato et al (2003) proposed a web service approach for the design of sensor networks. Their goal is to enable a flexible architecture where the sensor data can be accessed by users spread all over the world.

**Hourglass:** Harvard’s Hourglass dealt by Shneidman et al (2004) aims to be an Internet-based infrastructure for connecting a wide range of sensors, services, and applications in a robust fashion. To achieve this goal, they introduce a “circuit”, which can be seen as data links that connect one or more sensor networks sources to one or more recipients of that data. The overall architecture consists of an overlay network of well-connected dedicated machines that provide service registration, discovery, and routing of data streams from sensors to client applications so that streams of data elements are routed to one or more applications and the data elements generated from sensors inside of sensor networks can be entirely hidden from participants.

**Web-based Intelligent Sensor Explorer (WISE):** Hua et al (2004) developed a framework to provide the services: (i) a publishing facility to allow sharing of real-time sensor data on the Web (ii) a search mechanism to enable unsolicited users to discover relevant sensors and (iii) an intelligent browser to provide the user interface to view, analyze, and query the sensor data streams in intelligible ways. The WISE environment is not a sensor database management system. It is an extension to the web to enable exchange of sensor data and sharing of sensor applications.
SONGS: Songs dealt by Jie Liu and Feng Zhao (2005) is an architecture for WSNs that allows users to issue queries that are automatically decomposed into graphs of services which are mapped onto actual devices. SONGS does not provide service binding among heterogeneous devices.

Power-aware service-oriented architecture for wireless sensor networks: LiQun Zhuang et al (2005) proposed a middleware for accessing sensor data that is based on the service oriented architecture using web services and grid technologies. The purpose of the middleware is to integrate sensor networks into a networked virtual enterprise, making the physical data available to all applications in the enterprise. The middleware uses the Web Services Resource Framework (WSRF) to manage stateful (sensor) resources. The WSRF specifies how to use web services to access stateful resources. The authors address the issue of power efficiency for sensor application services and provide a query optimization mechanism at web services level based on the Markov Decision Process.

Service-centric model: Gracanin et al (2005) described service oriented model for WSN with Internet. The model separates the complicated query management between WSN and Internet from gateway to a special component outside the network.

Tinyrest: Luckenbach et al (2005) discussed how WSN can be integrated into the Internet and therefore supporting the development of IP based applications in different domains.

Tenet: Gnawali et al (2006) proposed solution that adopted SOA in developing middleware solution for WSN’s. Tenet simplifies application development for tiered sensor networks. It benefits from generic motes in the lower tier and masters, relatively unconstrained 32-bit platform nodes, in the upper tier. The Tenet software aims at distributing processing on sensor nodes
in the form of tasks. Each task consists of a chain of tasklets which are called sequentially. Tenet reduces communication costs with respect to virtual machine-based solution by relying on pre-deployed unitary complex operations. Tenet supports tasking with a very simple programming language but is limited by the number of tasklets available at nodes and does not offer any support for collaboration between sensor nodes.

**Atlas:** Atlas proposed by King et al (2006) is a service oriented sensor and actor platform which was developed by the University of Florida and encompasses the concepts of self-integrative, programmable pervasive spaces. The Atlas platform is a combination of hardware and software elements which allows these sensors and actors to expose themselves as services to other components. The Atlas platform is used in the Gator Tech Smart House and provides the researchers with a plug-and-play type mechanism for integrating new sensors and actors into the house. Each sensor and actor will register its service automatically with the controlling system and their services can then be invoked. The Open Services Gateway Initiative (OSGi) framework is used to handle the required service discovery and registry mechanisms.

**Sens-ation:** The service oriented paradigm proposed by Gross et al (2006) is based on service providers, service consumers, and brokers. It enables standardized communication within the platform, and between the platform and the sensors. Each Sens-ation server can act as a web service provider. These service providers allow encapsulating and hiding of all specific hardware implementation details of their attached sensors. They provide a simple common interface for other application to obtain real-time sensor data, or persistently stored past sensor data. The service consumers are independent of the Sens-ation service provider so that a service consumer does not depend on the implementation of the service and communicates with it according to a
well-defined interface. A Sens-ation broker contains information about Sens-ation service providers such as their registered sensors and their location. A service consumer can discover available sensors and their contact information via a broker. The service consumer can then directly request required sensor data from that service.

**SWAP:** The Sensor Web Agent Platform proposed by Moodley and Simonis (2006) makes use of both of the most promising distributed technologies: services and multi agent systems. It allows integrating arbitrary sensors or sensor networks into a loosely coupled higher level environment that facilitates developing and deploying end user applications across multiple application domains. It also proposed an ontology framework and an abstract hierarchy for fusing and integrating data.

**Wireless sensor networks - new challenges in software engineering:** Blumenthal et al (2003) discuss the need for service oriented architecture for sensor networks. The existing problems are discussed and the different mechanisms that they use are explained. Service oriented architecture is then proposed for the sensor level communication in the sensor network. The proposal states that sensors and sensor nodes should be different components that expose different services in the sensor network. The sensor components could then be dynamically exchanged during runtime. For example, different sensing algorithms could be downloaded from other sensor nodes.

**SOA for WSAN applications:** Prinsloo et al (2006) discussed in great detail the concept and model of service oriented wireless sensor network.

**The Global Sensor Network:** The Global Sensor Network proposed by Aberer et al (2007) platform takes up ideas of SOA and aims at making publication and access to sensor networks and sensor data as simple, powerful, and flexible as accessing web documents.
**OASiS:** Kushwaha et al (2007) proposed an Object centric, Ambient aware, Service oriented Sensor network programming framework and middleware. It enables the development of WSN applications without having to deal with the complexity and unpredictability of low level system and network issues. OASiS takes a service oriented approach to behavioral decomposition and application execution. Each activity (sensing, aggregation, service discovery, etc.) is implemented as a separate service. The middleware supports dynamic service discovery and configuration to address changes in network topology due to failures and unreliable communication links. Heterogeneity is addressed by employing well defined services on heterogeneous platforms that can be composed together in a seamless manner. OASiS supports real world integration in application design by providing the means to specify spatio temporal service constraints. The ability to attach such constraints to services before they are invoked is an important aspect of WSN application programming. The service oriented approach enables in-network data aggregation using services with inputs from multiple sensor services. OASiS supports specification of both application specific and network QoS requirements and can handle QoS violations using application reconfiguration to satisfy QoS requirements.

**TinySOA:** Rezgui and Eltoweissy (2007) proposed a TinyOS-based prototype called Service Oriented Sensor Actuator Networks (SOSANET) that achieves high levels of efficiency, interoperability, and scalability. SOSANET is a novel approach for building open, interoperable, customizable Sensor Actuator Networks (SANETs). In SOSANET, nodes’ sensing and actuation capabilities are exposed to applications in the form of a collection of in-network services. Services in the proposed approach are lightweight code units deployed directly on top of the operating system of nodes. Each service provides some sensing or actuation functionality supported by the local node. Services may be individually invoked or combined in complex ways to form a
‘virtual’ SANET with far richer sensing and actuation capabilities. Applications invoke services using a service oriented query model that offers a high abstraction level and, yet, enables a wide spectrum of low level optimization mechanisms.

Edgardo Aviles-Lopez and Antonio Garcia-Macias (2009) proposed a service oriented architecture that allows programmers to assess wireless sensor networks from their applications by using a simple service oriented API via the language of their choice.

**TinyWS:** Othman et al (2007) proposed a small web service platform that resides on the sensor nodes. It hosts the web services and has SOAP processing engine. The sensor nodes are service providers, the application devices are service requesters and a distributed UDDI acts as an overlay entity. TinyWS implements an HTTP server on each device and enables applications outside of the WSN to invoke services over the Internet using HTTP requests.

**Sensor Web frameworks:** This type of solutions generally aims at making the heterogeneous sensors (and actuators), and sensor reading repository discoverable and accessible for the Internet applications and users over the web as dealt by Xingchen Chu and Rajkumar Buyya (2006). They generally provide a mash-up application that allows visualizing the data. SensorMap discussed by Nath et al (2007) provides a set of tools that data owners can use to easily publish their data and a Graphical User Interface (GUI) that can be used by users to make queries over live data. SensorMap transparently provides mechanisms to archive, index the sensor data, process queries, and aggregates the sensor data. The SensorMap GUI is a mash-up application that lets users submit queries on available sensors and overlays the aggregated results on a map. The framework introduced by Santanche et al (2006) facilitates access to both real time and historical sensed data, through a variety
of access methods. It addresses the scalability issue by introducing a distributed sensor register. Also the authors Rhead et al (2008) proposed an overview about the worldwide sensor web framework.

**Service oriented design methodology:** Meshkova et al (2008) proposed design methodology that relies on service oriented architecture and agile methodology for the WSN. They have also proposed a list of parameters that most of the WSNs can be described with and discussed the major trade-offs WSN face.

**USEME:** Canete et al (2008) proposed a service oriented framework to develop wireless sensor and actor networks applications using middleware support. The main function of the middleware is to handle the publication and discovery of web services, communication related issues, real-time constraints and group management. The architecture of the middleware is divided into five core components including configuration, publication and discovery, invocation and communication, real-time constraint management and group management. Different parameters such as deadlines and discovery frequency can be changed through its configuration component. The publication and discovery component keeps track of the services available for use. The invocation and communication component takes care of translating the remote commands executed by application programmers into communication packets between remote nodes. The real time constraint management component manages constraints like setting the priorities for activities, specifying maximum execution time for a command and setting time intervals for periodical events. To achieve scalability and efficiency, the middleware groups the nodes that share common behaviors and automatically controls the use of services provided by the groups using the group management component.

**WSN-SOA stack:** Leguay et al (2008) proposed a multi-level service oriented architecture for sensor networks. This architecture bridges the gap
between devices having very different capacities and fully handles network
dynamicity by providing auto configuration features at both network and
service level. The authors highlighted the use of the Device Profile Web
Service (DPWS) standard for limited capacity devices (e.g., IP camera, PDA,
sensor network gateways, large deployable sensors) and they have introduced
the WSN-SOA suite which enables SOA based communications in networks
of low capacity sensors. Instead of one-to-one service translation between the
DPWS and WSN-SOA worlds, they propose to use a more efficient and
relevant ways of communication based on high level interfaces and
publish/subscribe event notifications using topics.

**SStreaMWare:** SStreaMWare proposed by Levent et al (2008) is a sensor
mediator Service Oriented Middleware which provides a query service to
access data issued by heterogeneous sensors. The main objective is to provide
a hybrid solution focusing on sensor heterogeneity management, scalability
and complex query evaluation. The main elements of SStreaMWare are the
sensors, the proxies and adapters, the gateways and the control sites. Its
architecture consists of three querying services and a discovery/binding
service. The Query Service is deployed on the control site and is responsible
for query evaluation. The gateway service is deployed on every gateway and
provides a query service for the sensors managed by that gateway. The proxy
service is deployed at concerned gateways and provides services to a specific
type of sensors. The lookup service is for service discovery and binding in
addition to managing the service registry.

Kavi Kumar Khedo and Subramanian (2009) proposed service-
oriented component based middleware architecture for wireless sensor
networks.

**Servilla:** Fok et al (2009) proposed a novel service provisioning framework
that enables applications to be platform independent while still able to access
platform specific capabilities. A salient feature of Servilla lies in its capability to support coordination and collaboration among heterogeneous devices inside a WSN. A specialized service description language is introduced that enables flexible matching between applications and services, which may reside on different devices. Servilla provides modular middleware architecture to enable resource poor devices to participate by contributing services, facilitating in-network collaboration among a wide range of devices.

**Arch Rock’s PhyNet:** PhyNet (http://www.archrock.com/) provides a central gateway that exposes WSN capabilities as a web service. ArchRock primer pack is a set of sensor management product originated in UC Berkeley. Its sensor node software is built on an industrial strength version of TinyOS and it provides a suite of deployment and site planning as well as configuration and management tools for quick installing and managing wireless sensor networks. The individual sensor nodes can be accessed and managed via IP based management tools. Arch Rock products are designed and built upon industry standards and open interfaces. The gateway server and sensor node software provide flexible programming interfaces to enterprise computing systems based on a web services framework including SOAP and REST interfaces. Such approach may be seen as a management tool for deployment and operation of wireless sensing and control networks.

**Web service middleware for LiteOS based wireless sensor networks:** The authors Masaaki Takahashi et al (2009) proposed web service architecture and implement a web service middleware for LiteOS based wireless sensor networks. The layered architecture comprised an application layer, a web service middleware layer, and a WSN layer. The web service middleware interacts with the front end web application as well as the WSNs so that users can remotely query and view the sensor readings of light, temperature, magnet, and acceleration.
2.2.2 Service Composition in Wireless Sensor Networks

There are several detailed surveys on service composition including the one studied by Bronsted et al (2010). Swaroop Kalasapur et al (2007) proposed service composition mechanism that models services as directed attributed graphs, maintains a repository of service graphs and dynamically combines multiple basic services into complex services. Further a hierarchical overlay structure created among the devices to exploit the resource unevenness, resulting in the capability of providing essential service related support to resource poor devices. Jongwoo Sung et al (2009) proposed the toolbox for service oriented wireless sensor networks which encapsulates complexities of sensor networks and enable simple service composition using an intuitive GUI.

Delicato et al (2010) sketched steps for creating an ecosystem of sensor generated data by integrating different WSNs through the use of web services and web mashup technologies. Fok et al (2009a) described how service provisioning in WSNs can adapt to application and network dynamics. Several strategies for achieving higher energy efficiency and more predictable quality of service are presented. Fok et al (2009b) have also proposed adaptive service provisioning framework with the objective of enhancing service availability through application transparent service rebinding and reduce energy consumption through energy aware service selection and sharing.

Xiumin Wang et al (2009) designed service composition solution for the persistent queries so that the involved routing update cost and transmission cost is minimum. They used greedy algorithm and dynamic programming to provide service composition solution with minimum cost. Wang et al (2010) designed cross layer sleep schedule to prolong the network lifetime while satisfying the service availability requirement at the application
layer. They used two approximation algorithms based on the LP relaxation and one efficient re-ordering heuristics algorithm to enhance the dependability of the service composition in a service oriented sensor networks. Sahin Cem Geyik et al (2010) proposed a novel service modeling and dynamic composition method for the unreliable and dynamic sensor network environments. Two heuristic algorithms for composition of sensor services are introduced that differ in the direction of traversing the service graph during the composition process. Centralized and distributed implementations of these algorithms are also described.

2.3 SUMMARY

This literature review provides an overview of some of the major research that formed the foundations for this research. There are good numbers of contributions in bringing the concept of service oriented architecture for wireless sensor network either in the form of middleware services or in the form of in-network services. But most of them assume WSN as flat architecture. This results in higher latency and poor scalability.

Limited work attempted to provide service composition solution over SOWSN. All the attempts towards providing service composition solution assume service graph as input. There is no contribution towards the generation of service graph. Also all the existing approaches aim at providing service selection only based on the energy consumption. QoS optimization problem with multiple global constraints is not considered in any of the proposed service selection approach. So this work proposes energy efficient, QoS based automatic service composition solution for SOWSN that generates the service graph automatically on-demand during run time. Table 2.1 gives the comparison of existing service oriented approaches for wireless sensor networks.
<table>
<thead>
<tr>
<th>Existing Approaches</th>
<th>Parameters</th>
<th>Service composition solution</th>
<th>Network Architecture</th>
<th>User Constraints</th>
<th>Service Sharing</th>
</tr>
</thead>
<tbody>
<tr>
<td>TinySOA</td>
<td></td>
<td>No</td>
<td>Flat</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Adaptive service provisioning</td>
<td></td>
<td>Semi dynamic</td>
<td>Flat</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Service composition Persistent query</td>
<td></td>
<td>Semi dynamic</td>
<td>Flat</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WSN as a service</td>
<td></td>
<td>No</td>
<td>Flat</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Service composition using BPEL</td>
<td></td>
<td>Static</td>
<td>Flat</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Dynamic Service composition</td>
<td></td>
<td>Dynamic</td>
<td>Flat</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SOToolbox</td>
<td></td>
<td>Semi dynamic</td>
<td>Flat</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>HESOR</td>
<td></td>
<td>No</td>
<td>Hierarchical</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SLIM</td>
<td></td>
<td>Static</td>
<td>Flat</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TinyWS</td>
<td></td>
<td>No</td>
<td>Flat</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>