CHAPTER 3

AUTOMATIC SERVICE GRAPH GENERATION

This chapter deals with the first two phases of service composition. A simple approach is explored to create service graph on-demand for service composition in service oriented wireless sensor network. In order to improve the scalability and energy efficiency the proposed Automatic Service Graph Generation (ASGG) algorithm first organizes the network into clusters, and then discovers abstract services that satisfies user constraints and identifies the relationship between these services. Based on the input/output dependency between these services the algorithm generates service graph. This service graph is also called as workflow or process model.

3.1 INTRODUCTION

Service composition is a technique used to aggregate or combine services in order to build up new, more sophisticated ones. Composition techniques can be classified into static and dynamic (automatic) composition. In static composition, services are combined by adding a central coordinator that is responsible for invoking and combining the individual subservices. This requires an abstract process model before the composition starts. The model includes a set of tasks and their data dependency. In dynamic composition, services are composed on demand, not configured or deployed in advance.
Providing automatic service composition solution for WSN environments imposes challenge, since sensor nodes are likely to be battery powered, the objective is to minimize the energy consumption. The problem is challenging especially in large scale, dense networks, where significant traffic is generated due to the intrinsic broadcast nature of the wireless medium.

In order to guarantee low energy consumption and to achieve the network scalability grouping sensor nodes into clusters has been widely pursued by the research community. Clustering in WSNs as discussed by Abbasi and Younis (2007) guarantees basic performance achievement with a large number of sensor nodes. In other words, clustering improves the scalability of WSNs. This is because clustering minimizes the need for central organization and promotes local decisions. There has been substantial amount of research on clustering protocols for WSNs. Most of the clustering protocols utilize two techniques which are selecting cluster heads with more residual energy and rotating cluster heads periodically to balance energy consumption of the sensor nodes over the network. These clustering algorithms do not take the location of the base station into consideration. This lack of consideration causes the hot spots problem in multi-hop WSNs. The cluster heads near the base station die earlier, because they will be in heavier relay traffic than the cluster heads which are relatively far from the base station. In order to avoid this problem, some unequal clustering algorithms are proposed in literature. In unequal clustering, the network is partitioned into clusters with different sizes. The clusters close to the base station are smaller than the clusters that are far from the base station.

This chapter proposes a Service driven Energy Efficient Clustering algorithm (SEEC). SEEC organizes nodes into service clusters and service clusters into regional clusters. This hierarchy of clustering is based on residual
energy, distance to the base station and service provided by the node. SEEC makes use of fuzzy logic approach to select cluster head.

Automatic service composition requires: (1) services to be located based on their capabilities and (2) service sequence that can be generated to create a composition. The main objective of this chapter is to generate service graph dynamically on demand based on the input/output dependency between services. The key contributions of this chapter are as follows.

- Service based energy efficient clustering algorithm
- Distributed directory based service discovery algorithm
- Service graph generation algorithm.

The remaining part of this chapter is organized as follows. Background and related work are presented in section 3.2. Section 3.3 gives the motivation for service graph generation, in section 3.4 detailed clustering algorithm is presented, the service discovery protocol is presented in section 3.5. Service graph generation algorithm is presented in the section 3.6. Section 3.7 includes performance evaluation and implementation results. Section 3.8 presents the summary.

3.2 BACKGROUND AND RELATED WORK

3.2.1 Background

Service composition is useful when we are looking for a service with specific input and output and there is no single service which satisfies the request. For example, assume that a user wants to find the service whose input is “City” and output is “Weather” among services listed in the Table 3.1. If the service composition is not supported, there is no answer to the user query. However, if the composition is supported, a sequence of
service can be provided as the answer to the user query. The composition of services S1, S2 and S3, in which the output of S1 is equal to the input of S2, and output of S2 is equal to the input of S3 satisfies the user query. Similarly for the user query with input City and output SportsOK the sequence of services are S1, S2, S3 and S4.

**Table 3.1 Example for service composition**

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Name</th>
<th>Operation</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>PlaceLookup</td>
<td>CitytoZipcode</td>
<td>City</td>
<td>ZipCode</td>
</tr>
<tr>
<td>S2</td>
<td>TemperatureFetcher</td>
<td>GetTemperature</td>
<td>ZipCode</td>
<td>Temperature</td>
</tr>
<tr>
<td>S3</td>
<td>WeatherFetcher</td>
<td>WeatherInfo</td>
<td>Temperature</td>
<td>Weather</td>
</tr>
<tr>
<td>S4</td>
<td>SportsStatusViewer</td>
<td>SportsByWeather</td>
<td>Weather</td>
<td>SportsOK</td>
</tr>
</tbody>
</table>

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.

Manual composition approach is generally used in the situation where the requestor has a well-defined process model. Normally this process model is generated off-line. In the Internet processes are defined using a process execution language like Business Process Execution Language (BPEL) or Ontology Web Language for Services (OWL-S). The problem with such an approach is that it demands too much knowledge on the part of the user and it becomes more and more difficult with the explosion of web services resources. Modifying any part of a process may result in the reconfiguration and redeployment of the whole process. In WSN the resource availability changes very frequently due to node movement, energy depletion,
node addition and node deletion. The process model generated by BPEL is static that is it cannot handle unpredictable changes in the runtime environment. So BPEL is not suitable for WSN. Automatic composition (without human involvement) is used when the requester has no process model but has a set of constraints and preferences. In WSN dynamic composition techniques are getting more preference due to their potential for handling unpredictable changes of runtime environment that cannot be handled using static composition techniques.

The first step in creating service graph requires a service discovery system. The role of service discovery system is to locate the service components that provide the functionality to be placed in the new service. It allows the selection of services providing functionalities that match the requested functionalities. More precisely, service discovery systems should also be able to find out all services conforming to a particular functionality. The service discovery system should also be scalable across large-scale networks and adaptable to dynamic changes especially when services dynamically join and leave the network.

The traditional method for service discovery in WSN is based on flooding, which has the advantage of zero maintenance overhead. However, flooding has obvious limitations with regard to energy efficiency and scalability. The problem is how to design a service discovery protocol suitable for wireless sensor networks that reduces the workload of the resource constraint devices and avoids the significant traffic induced by the traditional flood based solutions in dense networks. In this chapter a solution is proposed based on clustering, where a set of nodes, selected based on their capabilities, acts as a distributed directory of service registrations for the nodes in their cluster. In this way, the communication costs are reduced, since the service discovery messages are exchanged only among the directory nodes.
Usually services that are created by same or different providers are meant to be accessed and work independent of each other. But establishment of composite service based applications necessitates interaction, cooperation and communication of services. This leads to different types of dependency among services involved in composite services. Input/output dependency is one of the important dependencies that occurs when a service requires/or provides data from/to another service. Such dependency could occur between two services directly which is called direct dependency or indirectly between two services through one or more intermediate services which is called as indirect dependency. In this chapter a solution is proposed to find out the list of services and their execution order based on this dependency.

3.2.1.1 Assumptions

Before describing the proposed algorithm in detail, assumptions that are made about the network model are listed: (i) sensor nodes are heterogeneous, (ii) all sensor nodes and the base station are stationary after deployment phase, (iii) nodes have the capability of adjusting the transmission power according to the distance of the receiver nodes, (iv) all sensor nodes know their distance from the base station, (v) nodes providing similar services are colocated.

The radio model used is similar to the one used by Sadagopan et al (2005). \( E_{\text{elec}} = 50 \text{ nJ/bit} \) is the energy dissipated by the radio to run the transmitter or receiver circuitry and \( \text{amp} = 100 \text{ pJ/bit/m}^2 \) is the energy dissipation of the transmission amplifier. The energy expended during transmission and reception for a \( k \) bit message to a distance \( d \) between transmitter and receiver node is given in equations (3.1) and (3.2).

\[
E_{\text{tx}}(k, d) = E_{\text{elec}} \cdot k + \text{amp} \cdot k \cdot d^{\frac{3}{2}} \tag{3.1}
\]
ERx (k) = E_{elec} * k \quad (3.2)

where, $\lambda$ is the path loss exponent and $\lambda \geq 2$. To transmit a message, energy expenditure consists of two elements, that is the energy for the transmit electronics and for the transmit amplifiers. Whereas the energy expended for receiving consists of one element, that is, receiver electronics.

### 3.2.2 Related Work

Abbasi and Younis (2007) presented a survey on clustering. Some of the clustering algorithms employ fuzzy logic to handle uncertainties in WSNs. In the fuzzy clustering approach dealt by Gupta et al (2005), the cluster heads are elected at the base station. In every round, each sensor node forwards its clustering information to the base station. Energy Efficient Unequal Clustering (EEUC) proposed by Li et al (2005) is a distributed competitive unequal clustering algorithm. It uses probabilistic model to select tentative cluster heads. Every node has a preassigned competitive range. But EEUC only considers distance to the base station parameter to calculate competition radius. Cluster Head Election mechanism using Fuzzy logic (CHEF) dealt by Kim et al (2008) is a similar approach to that of Gupta et al but it performs cluster-head election in a distributed manner. Thus, the base station does not need to collect clustering information from all sensor nodes. But it utilizes fuzzy logic for assigning cluster head chances to tentative cluster heads. An Energy Aware Fuzzy Unequal Clustering algorithm for wireless sensor networks (EAUCF) proposed by Hakan Bagci and Adnan Yazici (2010) also employs a probabilistic model, but it does not elect the final cluster heads by just depending on this model.

Well known service discovery protocols include Service Location Protocol (SLP) proposed by (Guttman 1999), Jini proposed by (Waldo 1999), Universal Plug and Play (UPnP) proposed by Miller et al (2001), and
multicast Domain Name Service-Service Discovery (mDNS-SD) proposed by Apple computer, Inc. (2005). These service discovery protocols have taken different approaches to enable dynamic service registration, discovery, and service invocation. For example, mDNS-SD and UPnP have a clear focus on enabling address allocation without Dynamic Host Configuration Protocol (DHCP) servers, automatic discovery of computers, devices, and services on IP-based networks. Furthermore, in Jini, services are delivered as Java objects to service users requesting the service, making it possible to perform ordinary method calls. Service delivery and invocation in SLP and mDNS-SD, is on the other hand entirely left out from the protocol description. Marin Perianu et al (2008) proposed solution where cluster head nodes form a distributed service registry.

The Business Process Execution Language for Web Services (BPEL4WS) standard proposed by IBM, Microsoft and BEA is a combination of the concepts of Microsoft’s XLANG and IBM’s WSFL. This XML based language was designed to enable the coordination and composition of a set of services. It is based on the Web Services Description Language WSDL, which is basically an interface description language for service providers. The authors Curbera et al (2002a) discussed that BPEL is a behavioral extension of WSDL using a workflow based approach. It expresses relationships between multiple invocations by means of control and data flow links, and it employs a distributed concurrent computation model with variables.

Nils Glombitza et al (2008) introduced the Graphical Workflow Execution Language for Sensor Networks (GWELS) toolbox as graphical process modeling tool to realize the service oriented programming paradigm for sensor networks and ease the development and integration of sensor network applications. The authors Nils Glombitza et al (2009) also proposed an approach to integrate WSNs seamlessly into business process environments.
using BPEL and web service while using only very few resources on the sensor nodes. They introduce how application developers can use standard-compliant techniques to describe business processes that are using services offered by WSNs, without the need for hand-crafted code for data conversion.

Service Location and Invocation Middleware for Mobile Wireless Sensor and Actuator Networks (SLIM) proposed by Cugola and Margara (2010) is a middleware to support service oriented programming in mobile Wireless Sensor and Actuator Networks. SLIM brings service oriented programming directly inside the sensor network, allowing each node to act as a service provider, a service consumer, or both. SLIM exposes a simple yet powerful API to applications, allowing service registration, discovery, and invocation. To better address the needs of the typical applications for WSANs, SLIM supports both unicast and multicast invocations. The former allows service consumers to get data from a single sensor or to send a command to an actuator, while the latter allows to invoke all the services that satisfy a given query at once, e.g., to gather the data produced by a set of sensors in a single step. Moreover, the layered architecture of SLIM enables the usage of virtually every language for service descriptions and queries. SLIM includes an advanced routing protocol explicitly designed for a mobile, multi-hop WSAN to manage unreliable links and dynamic group membership, while it decouples the service matching policy from the communication layer, allowing applications to choose and install into the middleware the matching component that better fits their needs.

### 3.3 MOTIVATION

The current SOA model for WSN enables the service discovery; however, it does not consider the automatic and dynamic integration and composition of services. More precisely, this model specifies only services and operations that perform, but not the order of a flow specification of exchanged messages between services.
Since the topology of a sensor network changes frequently, Unpredictable changes of runtime environment cannot be handled using static composition techniques. Also it demands too much knowledge on the part of the user.

Automatic service composition dealt by Rao and Su (2004) 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 engineers could create flexible programs and are able to adapt to changes in the environment completely autonomously.

3.3.1 Issues in Static Service Composition

- Specifying manually which base services to use and how to combine them is cumbersome and ineffective with a rising number of relations between services and applications.

- With respect to the multitude of available services and service templates, it may be a time consuming task to manually select reasonable building blocks for the composite service.

- The creation of the data flow, i.e. the parameter assignments between the activities, can be complex and might require the user to have extensive knowledge about the underlying type representations.

- Furthermore the resulting composite process is vulnerable to change as it must be corrected manually when base services become unavailable or new better services appears.
3.3.2 Benefits of Dynamic Service Composition

- Modeling the control flow and the data flow of a composite service may be time consuming tasks. Automated composition techniques promise to speed up this procedure, thus bringing down the costs for developing new services.

- Time-to-market is accelerated since the provider may react faster and is more flexible to the customer requirements.

- The designed composite services improve in quality as the application of "intelligent" tools helps to create more efficient processes, e.g. by proposing parallel execution of functionally independent activities.

- Late binding may enhance fault tolerance and thus reliability. Since the actions in a process are not hardwired to concrete services, the unavailability of a service may be compensated through the invocation of a functionally equivalent one.

- There are some scenarios where important service characteristics (like price) change constantly, which makes the use of run time service discovery almost essential for the success of the composition.

3.4 SERVICE DRIVEN ENERGY EFFICIENT CLUSTERING (SEEC)

3.4.1 Clustering Basics

Clustering is a sample of layered protocols in which a network is composed of several clusters of sensors. Each cluster is managed by a special node, called Cluster Head (CH), which is responsible for coordinating the
data transmission activities of all sensors in its cluster. All sensors in a cluster communicate with a cluster head that acts as a local coordinator for performing intra transmission arrangement and data aggregation. Cluster heads in turn transmit the sensed data to the global sink.

The clustering methods in sensor networks can be categorized into static and dynamic types. The static clustering methods aim at minimizing the total energy spent during the formation of the clusters for a set of given network parameters, such as the number of nodes in the network. A problem that is closely related to the static clustering is the localized topology control, which maintains energy efficient network connectivity by controlling the transmission power at each node, or selecting a small subset of the local links of a node. One way is to minimize the total power levels in all nodes and search for a connected topology. Another way is to select a minimum set of sensors that form a connected communication graph to cover the entire network region, by iteratively searching for one path at a time and adding the nodes of the path to a set of already selected sensors.

The dynamic clustering methods deal with the same energy efficiency problem as the static ones but target for a set of changing network parameters, such as the number of active nodes or the available energy levels in a network. Low Energy Adaptive Clustering Hierarchy (LEACH) proposed by Wendi et al (2000), and Hybrid Energy Efficient Distributed clustering (HEED) proposed by Younis and Fahmy (2004) are the two commonly used dynamic clustering protocols for WSN. In LEACH the position of a cluster head was rotated among the nodes within a cluster depending on their remaining energy levels. It was assumed that the number of active nodes in the network and the optimal number of clusters are the parameters for dynamic clustering. HEED periodically selects cluster heads according to a hybrid of their residual energy and a secondary parameter, such as node
proximity to its neighbors or node degree. HEED does not make any assumption about the distribution or density of nodes, or about node capabilities, e.g., location awareness. The clustering process terminates in $O(1)$ iterations, and does not depend on the network topology or size.

In addition to supporting network scalability, clustering has numerous advantages.

- It can localize the route set up within the cluster and thus reduce the size of the routing table stored at the individual node.

- Clustering can also conserve communication bandwidth since it limits the scope of inter-cluster interactions to CHs and avoids redundant exchange of messages among sensor nodes.

- Moreover, clustering can stabilize the network topology at the level of sensors and thus cuts on topology maintenance overhead. Sensors would care only for connecting with their CHs and would not be affected by changes at the level of inter-CH tier.

- The CH can also implement optimized management strategies to further enhance the network operation and prolong the battery life of the individual sensors and the network lifetime. A CH can schedule activities in the cluster so that nodes can switch to the low power sleep mode most of the time and reduce the rate of energy consumption. Sensors can be engaged in a round robin order and the time for their transmission and reception can be determined so that the sensors retries are avoided, redundancy in coverage can be limited and medium access collision is prevented.
Furthermore, a CH can aggregate the data collected by the sensors in its cluster and thus decrease the number of relayed packets.

In cluster based model only cluster heads perform data aggregation whereas in the multi-hop model every intermediate node performs data aggregation. As a result, the cluster based model is more suitable for time-critical applications than the multi-hop model. A cluster based hierarchy moves the data faster to the base station thus reducing latency than in the multi-hop model.

However, clustering has one drawback, namely, as the distance between clustering level increases, the energy spent is proportional to the square of the distance. This increases energy expenditure. Despite this drawback, the benefits of this model far outweigh its drawback.

There has been substantial amount of research on clustering protocols for WSNs. Most of the clustering protocols like LEACH and HEED utilize two techniques which are selecting cluster heads with more residual energy and rotating cluster heads periodically to balance energy consumption of the sensor nodes over the network. These clustering algorithms do not take the location of the base station into consideration. This lack of consideration causes the hot spots problem in multi-hop WSNs. The cluster heads near the base station die earlier, because they will be in heavier relay traffic than the cluster heads which are relatively far from the base station. In order to avoid this problem, some unequal clustering algorithms are proposed in literature. In unequal clustering, the network is partitioned into clusters with different sizes. The clusters close to the base station are smaller than the clusters that are far from the base station.
In hierarchical clustering appropriate cluster head election can drastically reduce the energy consumption and enhance the lifetime of the wireless sensor network. The proposed clustering algorithm makes use of fuzzy logic to elect cluster head. Fuzzy logic is useful for making real time decisions without the need for complete information about the environment. On the other hand, conventional control mechanisms generally need accurate and complete information about the environment. Fuzzy logic can also be utilized for making a decision based on different environmental parameters by blending them according to predefined rules. The proposed algorithm blends residual energy and distance to the base station. Also it limits the distribution of routing states only to cluster heads. This will reduce the maintenance traffic.

3.4.2 Fuzzy Logic Basics

The basic elements of the Fuzzy Logic Controller are shown in Figure 3.1. They are the fuzzifier, inference engine, fuzzy rule base and defuzzifier. The process is performed in four steps.

**Figure 3.1 Structure of fuzzy logic controller**

**Fuzzification of the input variables:** It takes the input variables and determines the degree to which these inputs belong to each of the appropriate fuzzy sets.
**Rule evaluation:** It takes the fuzzified inputs, and applies them to the antecedents of the fuzzy rules. Because the given fuzzy rule has multiple antecedents, the fuzzy operator (AND) is used to obtain a single number that represents the result of the antecedent evaluation.

**Aggregation of the rule outputs:** It takes the membership functions of all rule consequent previously clipped or scaled and combine them into a single fuzzy set.

**Defuzzification:** It evaluates the rules, but the final output of a fuzzy system has to be a crisp number. The Centre of Gravity (COG) is usually used as a defuzzification method, and it is expressed in equation (3.3).

\[
\text{COG} = \frac{\sum \mu_A(x) \times x}{\sum \mu_A(x)}
\]  

where, \( \mu_A(x) \) is the membership function of set A.

The basic idea of the fuzzy set theory is that an element belongs to a fuzzy set with a certain degree of membership. Mamdani style fuzzy inference is used, which is the most frequently used fuzzy inference technique. The membership functions used are triangular and trapezoidal membership functions because they are suitable for real time operation. They are shown in Figure 3.2 and are given in equation (3.4) and (3.5).

\[
f(x, x_0, a_0, a_1) = \begin{cases} 
\frac{x - x_0}{a_0} + 1 & \text{for } x_0 - a_0 < x \leq x_0 \\
\frac{x_0 - x}{a_1} + 1 & \text{for } x_0 < x \leq x_0 + a_1 \\
0 & \text{otherwise}
\end{cases}
\]  

(3.4)
Figure 3.2 Triangular and trapezoidal membership functions

In f(x), \( x_0 \) is the center of the triangular function. In g(x), \( x_0(x_i) \) is the left (right) edge of the trapezoidal function and \( a_0(a_1) \) is the left (right) width of the triangular or trapezoidal function.

Fuzzy based control algorithm is used in the sink node for electing the cluster heads. Several reasons support the use of fuzzy control in this regard.

- Representing the problem in mathematical (or probabilistic) model domain involves dealing with several variables and parameters at a time. Moreover these variables are to be defined separately for each scenario, in order to provide a collective output on the basis of the multiple input variables. Problem arises as the number of these variables increases. The mathematical model becomes too complex to handle so many parameters at a time, limited by the effective combination of different parameters together. Fuzzy logic systems on the
other hand have got an inherent ability to integrate numeric (‘fuzzy’) and symbolic (‘logic’) aspects of reasoning. Therefore different parameters like residual energy and distance to the base station can be combined easily to give the desired result by defuzzifying the output fuzzy set.

- Fuzzy logic is capable of making real time decisions, even with incomplete information. Conventional control systems rely on an accurate representation of the environment, which generally does not exist in reality. Fuzzy logic systems, which can manipulate the linguistic rules in a natural way, are hence suitable in this respect. In addition, they can be used for context by blending different parameters - rules combined together to produce the suitable result.

- Fuzzy logic offers a full range of operators to combine uncertain information in a better way than any other system. Fuzzy logic control techniques can be used to design individual behavior units. Fuzzy controllers incorporate heuristic control knowledge in the form of if-then rules. They have also demonstrated a good degree of robustness in the face of large variability and uncertainty in the parameters.

3.4.3 Cluster Formation

The proposed clustering algorithm is service based, distributed and unequal. The clusters are formed based on residual energy, distance from the base station and service provided by the node. The nodes providing same service belongs to the same cluster.

If a cluster head is closer to the base station, it has to relay more data forwarding traffic than the sensor nodes which are far from the base
Each sensor node in the network tries to send its data to the base station. Therefore, as we get close to the base station, the data forwarding traffic increases. SEEC is based on the idea of decreasing the cluster sizes that are closer to the base station. If a cluster head closer to the base station has less intra cluster work, then it can contribute to inter cluster data forwarding more. SEEC achieves this goal by assigning smaller competition ranges to the sensor nodes that are closer to the base station. In other words, the competition range of the sensor node decreases as its distance to the base station decreases.

**Competition radius computation**

The first step in cluster formation is selecting the cluster head. To select cluster head each node calculates the Competition Radius (CR) based on the distance from the base station and residual energy. SEEC uses fuzzy logic to compute competition radius.

To compute competition radius, two fuzzy input variables are used. The first one is the distance to the base station of a particular node. The linguistic variables for this fuzzy set are close, medium and far. Trapezoidal membership function is selected for close and far. On the other hand, the membership function of medium is a triangular membership function. The second fuzzy input variable is residual energy of a particular node. Low, medium and high are the linguistic variables of this fuzzy set. Low and high linguistic variables have a trapezoidal membership function while medium has a triangular membership function. The only fuzzy output variable is the competition radius of the node. The outcome to represent the node cluster head election chance was divided into nine levels: very small, small, rather small, medium small, medium, medium large, rather large, large, and very large. This is shown in Table 3.2.
Table 3.2 Linguistics parameters and their fuzzy sets

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Fuzzy sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distance</td>
<td>Close, Medium, Far</td>
</tr>
<tr>
<td>Residual energy</td>
<td>Low, Medium, High</td>
</tr>
<tr>
<td>Competition</td>
<td>Very Small, Small, Rather Small, Medium Small, Medium, Medium Large, Rather Large, Large, and Very Large.</td>
</tr>
<tr>
<td>radius</td>
<td></td>
</tr>
</tbody>
</table>

Each node in the network calculates the competition radius by using fuzzy logic control. If a node has maximum residual energy and it is located at the maximum distance to the base station, then it has the maximum competition radius. If a node has minimum residual energy and it is closest to the base station, then it has minimum competition radius. All remaining possibilities fall in between these two extremes. The mapping of input and output variables are shown in Table 3.3.

Table 3.3 Competition radius calculation

<table>
<thead>
<tr>
<th>Distance to the base station</th>
<th>Residual energy</th>
<th>Competition radius</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close</td>
<td>Low</td>
<td>Very Small</td>
</tr>
<tr>
<td>Close</td>
<td>Medium</td>
<td>Small</td>
</tr>
<tr>
<td>Close</td>
<td>High</td>
<td>Rather Small</td>
</tr>
<tr>
<td>Medium</td>
<td>Low</td>
<td>Medium Small</td>
</tr>
<tr>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Medium</td>
<td>High</td>
<td>Medium Large</td>
</tr>
<tr>
<td>Far</td>
<td>Low</td>
<td>Rather Large</td>
</tr>
<tr>
<td>Far</td>
<td>Medium</td>
<td>Large</td>
</tr>
<tr>
<td>Far</td>
<td>High</td>
<td>Very Large</td>
</tr>
</tbody>
</table>
The competition radius of each tentative cluster head changes dynamically in SEEC, because SEEC uses residual energy parameter with distance to the base station metric of the sensor node to calculate competition radius. It is logical to decrease the service area of a cluster head while its residual energy is decreasing. If the competition radius does not change as the residual energy decreases, the sensor node runs out of battery rapidly. SEEC takes this situation into consideration and decreases the competition radius of each sensor node as the sensor nodes battery level decreases. The change of competition radius according to residual energy and distance to the base station parameters is demonstrated by the examples in Table 3.4.

**Table 3.4 Examples for fuzzy cluster competition radius calculation**

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Distance to the base station (m)</th>
<th>Fuzzy set for distance</th>
<th>Residual energy (J)</th>
<th>Fuzzy set for residual energy</th>
<th>Competition radius</th>
<th>Transmission range (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>174</td>
<td>Close</td>
<td>0.49</td>
<td>Low</td>
<td>Very-Small</td>
<td>17.66</td>
</tr>
<tr>
<td>44</td>
<td>298</td>
<td>Close</td>
<td>0.55</td>
<td>Low</td>
<td>Very-Small</td>
<td>19.66</td>
</tr>
<tr>
<td>6</td>
<td>231</td>
<td>Close</td>
<td>0.70</td>
<td>Medium</td>
<td>Small</td>
<td>32.65</td>
</tr>
<tr>
<td>48</td>
<td>130</td>
<td>Close</td>
<td>0.59</td>
<td>Medium</td>
<td>Small</td>
<td>29.09</td>
</tr>
<tr>
<td>60</td>
<td>291</td>
<td>Close</td>
<td>0.89</td>
<td>High</td>
<td>Rather-Small</td>
<td>56.89</td>
</tr>
<tr>
<td>2</td>
<td>441</td>
<td>Medium</td>
<td>0.53</td>
<td>Low</td>
<td>Medium –Small</td>
<td>77.89</td>
</tr>
<tr>
<td>24</td>
<td>452</td>
<td>Medium</td>
<td>0.56</td>
<td>Low</td>
<td>Medium –Small</td>
<td>78.89</td>
</tr>
<tr>
<td>8</td>
<td>382</td>
<td>Medium</td>
<td>0.69</td>
<td>Medium</td>
<td>Medium</td>
<td>141.28</td>
</tr>
<tr>
<td>43</td>
<td>358</td>
<td>Medium</td>
<td>0.59</td>
<td>Medium</td>
<td>Medium</td>
<td>127.28</td>
</tr>
<tr>
<td>55</td>
<td>520</td>
<td>Far</td>
<td>0.69</td>
<td>Medium</td>
<td>Large</td>
<td>149.43</td>
</tr>
<tr>
<td>81</td>
<td>655</td>
<td>Far</td>
<td>0.71</td>
<td>Medium</td>
<td>Large</td>
<td>170.93</td>
</tr>
<tr>
<td>62</td>
<td>622</td>
<td>Far</td>
<td>0.88</td>
<td>High</td>
<td>Very-Large</td>
<td>201.85</td>
</tr>
<tr>
<td>80</td>
<td>584</td>
<td>Far</td>
<td>0.91</td>
<td>High</td>
<td>Very-Large</td>
<td>200.01</td>
</tr>
</tbody>
</table>
Cluster head election

After each node determines its competition radius, cluster head competition begins. Each node assumes itself as Tentative Cluster Head (TCH) and broadcasts its residual energy along with its ID. Transmission range of this broadcast message is equal to the competition radius of the node. Each receiving node compares its residual energy with received residual energy. If its residual energy is lesser than received residual energy, then the node will advertise QuitElectionMessage and leave the cluster head election process. Otherwise the node will become a Candidate Cluster Head (CCH). All the nodes that advertised QuitElectionMessage will become Non Cluster Head (NCH) node. Each candidate cluster head advertises CandidateCHMessage (CCHM) to compete with other candidate cluster heads locally. This message is advertised to the candidate cluster heads which are inside the maximum cluster head competition radius. It includes node ID, residual energy level of the node, distance to the base station and the service provided by the node. Each candidate cluster head upon receiving candidate cluster head message compares its distance to base station with the received distance. The candidate cluster heads with least distance to the base station will become the Regional Cluster Head (RCH) and, all the other candidate cluster head becomes Service Cluster Head (SCH).

Joining the cluster

All the non cluster head nodes upon receiving CandidateCHMessage will check for the service. If a non cluster head node provides the same service as that of the received message, then it will send the JoinMessage to become the member of the cluster. Upon receiving the JoinMessage the candidate cluster head includes the ID of the non cluster head node in its cluster member list. If the non cluster head node receives message from more
than one candidate cluster head providing the same service, then it will select the candidate cluster head which is closer to the base station.

**Clustering algorithm**

Figure 3.3 shows the procedure for cluster formation that is executed at bootstrapping time and periodically in predetermined time intervals on each node. Cluster heads are rotated in each round in order to balance energy. The term round refers to the interval between two consecutive cluster formation process.

![Figure 3.3 Procedure for cluster formation](image)

**Figure 3.3 Procedure for cluster formation**
3.5 SERVICE DISCOVERY

Service discovery is the action of finding and locating a service in the network. Given a description of a requested service, the result of service discovery is the address of one or more service providers that are able to offer the specified service. When the address is retrieved, the user may further access and use the service offered by the provider. A Service Discovery Protocol (SDP) consists of three participating entities, the client (or user, service consumer), server (or service provider), and the directory (or registry, server, broker, central, resolver). These three entities cooperatively participate in achieving the service discovery objectives. The task of service discovery is to identify services based on matching of their input and output parameters with the corresponding ones in the service request. The common building blocks and techniques of service discovery protocols include.

(i) Service Catalogues: Service discovery protocols can be categorized as either a centralized directory based protocol or distributed directory less protocol. In the former, nodes register their available services with a central repository where service users query for available services. In the latter scheme, the protocol is inherently peer-to-peer and the service catalogue is distributed over the nodes.

(ii) Service Description: The service discovery protocol must define a data description language, representing and describing the service. In addition, the additional capabilities of the service, or attributes, usually have a standard naming convention.

(iii) Registration and Discovery: For service users to be able to find other nodes and available services, the services must be registered and a discovery process has to take place. To
discover services, the discovery process can either be active by issuing queries or passive by listening to service announcements from peer nodes.

(iv) Utilization: Other important characteristics for service discovery protocols are the techniques for supporting service delivery and service invocation. For some service discovery protocols, the responsibility for service invocation is controlled by higher level protocols apart from the actual service discovery protocol. Other protocols provide the necessary means to utilize the service by exporting a service interface.

(v) Service Status: To maintain a consistent state, it is necessary that the service discovery has a mechanism to notify the service users, ensuring that a clients’ knowledge of an announced service is still valid. Either a client can receive a change of a service state by receiving asynchronous notification of a specific event, or by frequently polling the service.

3.5.1 Distributed Directory Based Service Discovery

Sensor Service Information Table

Since services in WSN are not permanent or static, but changeable according to sensor node replacements due to exhausted battery, topology changes or movements of sensor nodes. Service discovery allows consumers to know available services at the time of service compositions. For the discovery purpose, in wireless sensor networks each sink node maintains a directory. The directory is a special data structure called Sensor Service Information (SSI) table. The SSI table describes the dependency relationship between WSN services. The table consists of unique identifier of the service, name of the service, input parameter required for the service, output
parameter produced by the service and some non-functional parameters about
the service. For simplicity it was assumed that each service takes one input
and produces one output. For each entry in the SSI table two lists are
maintained: Forward dependency list (dep_fwd) and backward dependency
(dep_bck) list. Forward dependency list contains all services that take output
of this service as the input. Backward dependency list contains all services
that produce output that is equal to input of this service.

Whenever a new service is added to the table, the algorithm checks
whether output of this new service is input for any of the existing service. If
so the new service is added to the dep_back list which is pointed by the link
field of the existing service. Similarly the algorithm checks whether input of
this new service is output of any other existing service. If so the existing
service is added to the dep_bck list of this new service. Whenever a new
service is added to the table, the algorithm checks whether input of this new
service is output for any of the existing service. If so the new service is
added to the dependency forward list which is pointed by the link field of the
existing service. Similarly the algorithm checks whether output of this new
service is input of any other existing service. If so the existing service is
added to the dep_fwd list of this new service. The structure of the SSI table is
given in the Figure 3.4.

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service name</th>
<th>Input parameter</th>
<th>Output parameter</th>
<th>QoS</th>
<th>Fwd-link</th>
<th>Bwd-link</th>
</tr>
</thead>
</table>

![Figure 3.4 The directory structure of the SSI table](image-url)
To maintain up-to-date information the following mechanism is designed: (i) registration by sensor nodes (ii) update by sensor nodes and (iii) active discovery. Among the three methodologies, registration and update are passive while active discovery is actively operated by the sink node.

- After sensor nodes are turned on, they send registration messages to the cluster head nodes to notify their existence. Registration message contains information about service(s) it provides.

- The sink node receives the registration message from the cluster head nodes and records it in the SSI table.

- A node running a service can wait for significant changes in service conditions to notify the sink node. Examples of such significant changes are threshold on reliability, critical low battery level of the sensor node etc. Sensor nodes send the update message to the sink node whenever change occurs.

- There are possibilities that sensor node leave the network without update message or registration failed with unexpected errors. To actively figure out this situation quickly as soon as possible, sink node sends active discovery message to sensor network in periodic time or when user request arrives.

- Each sink node exchanges the information in the SSI table with other sink nodes to improve the reliability.
3.5.2 Candidate Abstract Service Discovery

In an attempt to find an appropriate service providing a requested functionality it is a common situation to find a service that is able to generate the requested effects but some of its preconditions are not fulfilled. In these cases, chaining of services can be used to overcome that problem as dealt by Kuster et al (2005). The most common approaches to service chaining include graph search, forward chaining and backward chaining.

Graph search approaches rely on building a graph representation of all services available. In a graph, nodes represent available services and edges represent whether one of a service’s output may serve another service as one of its inputs. Then shortest path algorithm is used to find the shortest path from the user’s input to the expected outputs which represent the best available solution. Problem with this approach is that they do not scale well with the number of offered services.

In forward chaining starting with inputs from the service request the planning system uses applicable services. The problem with forward chaining is it tends to search in directions that are unnecessary for the requested effects.

Backward chaining works similar to forward chaining except that the composition starts with those services generating the requested output instead of those services whose input matches with the inputs in the request.

To reduce the search space the ASGG combines the forward and backward chaining. Suppose a service request SR (I, O) arrives, the algorithm searches the table in the backward direction starting with the service with output = O and in the forward direction starting with the input = I. Also the algorithm checks whether the composition of the services exceeds the user specified constraint. If it exceeds, an appropriate message is sent back to the
user. After discovery, the stack contains all the necessary services to answer the user request. Figure 3.5 shows the service discovery algorithm, the time complexity of this algorithm in the worst case is $O(n)$, where $n$ is the number of services.

<table>
<thead>
<tr>
<th>Algorithm 3.2: Service discovery</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Sensor service information table, user request</td>
</tr>
<tr>
<td><strong>Output:</strong> The service/services required to satisfy user request</td>
</tr>
<tr>
<td><strong>Notations:</strong></td>
</tr>
<tr>
<td>E – One instance of the row in the table</td>
</tr>
<tr>
<td>Dep_back - List that stores services during backward chaining</td>
</tr>
<tr>
<td>Dep_fwd - List that stores services during forward chaining</td>
</tr>
<tr>
<td>SF – stack that stores services during forward chaining respectively</td>
</tr>
<tr>
<td>SB - stack that stores services during backward chaining respectively</td>
</tr>
</tbody>
</table>

```plaintext
if E’s input is equal to user input, then
  if E’s output is equal to user output, then
    if cost of the service is < user specified cost constraint, then
      return E.id
  else if E’s output is equal to user output, then
    If total composition cost is less than user specified cost constraints
    push(E) to stack SB
    traverse the dep_bck list
    for each service W in dep_bck list
    L1:
      if W’s input is equal to user’s input, then
        push(W) to stack SB
      else
        traverse dep_bck of W
        for each service W1 of dep_bck of W
        assign W1 to W
        goto L1
    else
      traverse dep_bck of W
      for each service W1 of dep_bck of W
      assign W1 to W
      goto L1
  else
    if E’s input is equal to user input, then
    if total composition cost is less than user specified cost constraints
    push(E) to stack SF
    traverse the dep_fwd list
    for each service W in dep_fwd list
    L2:
      if W is equal to the top value of stack SB, then
        return popped value from stack SF and SB
    else
      traverse dep_fwd of W
      for each service W1 of dep_fwd of W
      assign W1 to W
      goto L2
    else
      user request cannot be satisfied
      return;
end
```

Figure 3.5 Algorithm for service discovery
3.6 SERVICE GRAPH GENERATION

Service graph is wiring of group of services together describing application flow. While several proposals exist for service composition on the Internet, the most important one is BPEL. BPEL is not suitable for wireless sensor networks because the process model generated by BPEL is static.

3.6.1 Extracting Dependency between the Abstract Services

After discovery, candidate abstract services are represented in the form of Dependency Matrix (DM) that indicates all direct dependencies between services. As specified by Li (2003) for component dependency, the matrix that models the dependency will be a square matrix (n*n) where n is the number of component services in the composite service. In the dependency matrix each service is represented by a column and a row. If a service $S_i$ in the $i^{th}$ column is dependent on service $S_j$ in the $j^{th}$ row (i.e output of service $S_j$ is the input of the service $S_i$) then $DM[i,j] = 1$, otherwise $DM[i,j] = 0$. More formally, the values of all the elements in DM are defined as $d_{ij} = 1$ if $S_i$ is dependent on $S_j$ otherwise $d_{ij} = 0$. This dependency information is obtained for each candidate abstract service by traversing the link field in the SSI table. Direct dependency matrix for the service composition example discussed in section 3.2.1 is defined in the following matrix.

$$DM = \begin{bmatrix}
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix}$$

The Figure 3.6 shows the algorithm for finding direct dependency matrix. It takes SSI table as the input and produces direct dependency matrix as the output.
Algorithm 3.3: Computing direct dependency

Input: List of candidate abstract services, SSI table
Output: Direct dependency matrix DM

Notation:
i, j, k – index variables
n – number of services
directdependency()
{
    for(i=0; i<n; i++)
        for(j=i+1; j<n; j++)
            if input of \( s_i \) is equal to output of \( s_j \)
                DM[i][j]=1
            else
                DM[i][j]=0
}

Figure 3.6 Algorithm to compute direct dependency matrix

Because of the transitivity of dependencies, it is possible to gather all indirect dependencies between the services by calculating the transitive closure. For example if service B has a direct dependency on service A and service C has direct dependency on service B, then service C will have indirect dependency on service A. Thus, one should traverse all possible explicit direct service dependency chain to extract indirect dependencies. This dependency chain is a linked list of services that starts from a service in focus and terminates with a service that doesn’t have direct dependency with any service. The link between individual services in a chain represents the direct dependency between services. For example one possible dependency chain for S4 is: S4-S3-S2-S1-none. The Warshall’s algorithm is used to calculate the transitive closure. The Indirect Dependency Matrix (IDM) for the above matrix is shown below.
The Figure 3.7 shows the algorithm for finding indirect dependency matrix. It takes direct dependency matrix as the input and produces indirect dependency matrix as the output matrix.

\[
\text{IDM} = \begin{bmatrix}
0 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0
\end{bmatrix}
\]

**Algorithm 3.4: Computing indirect dependency**

**Input**: Direct dependency matrix DM[i, j]

**Output**: Indirect dependency matrix IDM

```
Indirectdependency()
{
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            IDM[i][j]=DM[i][j]
    for(k=0; k<n; k++)
        for(i=0; i<n; i++)
            for(j=0; j<n; j++)
                if(!IDM[i][j])
                    IDM[i][j]=IDM[i][k]&&IDM[k][j];
}
```

**Figure 3.7 Algorithm to compute indirect dependency**

From the indirect dependency matrix the following observations can be made.

- If the matrix is upper triangular dependency matrix, all dependency relationship between services is uni-direction.
- If the dependency matrix is sparse dependency matrix, the system is a loosely coupled system.
Cyclic dependency is the good indicator of bidirectional communication. It can be identified by comparing the symmetrical elements or by checking its diagonal elements value of dependency matrix. The cyclic dependency exists if any one of the following conditions holds.

- Symmetrical elements of the dependency matrix are equal to 1.
- A diagonal element of dependency matrix is 1.

### 3.6.2 Ordering Services

To generate the process model, knowledge of which services are independent and which services are heavily dependent is essential. This information is obtained by calculating the Dependency Coefficient (DC) for each service. The dependency coefficient of a service $S_k$ can be defined as follows.

$$DC (S_k) = \sum_{j=0}^{n} |d_{jk}|$$  \hspace{1cm} (3.6)

The dependency coefficient represents the degree of dependency between the service and the rest of the services. The bigger the value of the $DC (S_k)$ is, the stronger the dependency between the service and the rest of the system.

Now using simple sorting algorithm the services are ordered based on their dependency coefficient. The service with the lowest dependency coefficient stands first and the service with the highest dependency coefficient stands last in the order. Now services are executed based on increasing order of their dependency coefficient. If the dependency coefficient of the two services is same, then these two services can be executed in parallel. If the dependency coefficient of a service is 0, then the service is an independent
service and it can be executed in parallel with other services. For the above scenario \( DC(S1) = 0, DC(S2) = 1, DC(S3) = 2, DC(S4) = 3 \).

After finding relationship between services the necessary control structure should be attached to the respective services.

### 3.7 PERFORMANCE EVALUATION

The proposed algorithm was implemented in C# and .NET framework and carried out a series of experiments on pentium IV machine with 4GB RAM and 320 GB HDD running windows 2000. To evaluate the performance, 100 distinct cases of services and 100 distinct user queries were considered. Table 3.5 shows sample services and Table 3.6 shows an example service requests submitted and the candidate abstract services order produced. For simplicity, the parameter \( P \) denotes the input and output parameter, \( S \) denotes the services and \( SR \) denotes the service request. For all the service requests the ordering of candidate abstract services produced by the algorithm is valid.

#### Table 3.5 Sample services

<table>
<thead>
<tr>
<th>Service ID</th>
<th>Service Name</th>
<th>Input</th>
<th>Output</th>
<th>Fwd-Dep List</th>
<th>Bwd-Dep List</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>Service0</td>
<td>P1</td>
<td>P2</td>
<td>S1</td>
<td>S3,S4,S9</td>
</tr>
<tr>
<td>S1</td>
<td>Service1</td>
<td>P2</td>
<td>P8</td>
<td>S8</td>
<td>S0</td>
</tr>
<tr>
<td>S2</td>
<td>Service2</td>
<td>P9</td>
<td>P4</td>
<td>S4</td>
<td>Null</td>
</tr>
<tr>
<td>S3</td>
<td>Service3</td>
<td>P3</td>
<td>P1</td>
<td>S0</td>
<td>S5</td>
</tr>
<tr>
<td>S4</td>
<td>Service4</td>
<td>P4</td>
<td>P1</td>
<td>S0</td>
<td>S2</td>
</tr>
<tr>
<td>S5</td>
<td>Service5</td>
<td>P5</td>
<td>P3</td>
<td>S3</td>
<td>S7</td>
</tr>
<tr>
<td>S6</td>
<td>Service6</td>
<td>P11</td>
<td>P12</td>
<td>Null</td>
<td>Null</td>
</tr>
<tr>
<td>S7</td>
<td>Service7</td>
<td>P7</td>
<td>P5</td>
<td>S5</td>
<td>S8</td>
</tr>
<tr>
<td>S8</td>
<td>Service8</td>
<td>P8</td>
<td>P7</td>
<td>S7</td>
<td>S1,S10</td>
</tr>
<tr>
<td>S9</td>
<td>Service9</td>
<td>P13</td>
<td>P1</td>
<td>S0</td>
<td>Null</td>
</tr>
<tr>
<td>S10</td>
<td>Service10</td>
<td>P10</td>
<td>P8</td>
<td>S8</td>
<td>Null</td>
</tr>
</tbody>
</table>
Two metrics success ratio and composition time were considered to evaluate the performance of the algorithm and were compared with the performance of SLIM and naive. Here naive denotes a system without any composition solution.

**Success ratio**

The success ratio is the ratio of the total number of successful attempts to the total number of service requests. Two scenarios were considered to demonstrate the success ratio. In the first scenario, success ratio was measured by increasing the number of services. The more the number of available services the higher is the success ratio. Two different query sets were used each consisting of 20 one time queries and at the maximum each query consists of five component services. The Figure 3.8 shows the success ratio of the query set I and Figure 3.9 shows the success ratio of query set II. For both the query sets, the performance of ASGG is better than naive and SLIM. The reason for this is that more number of queries are answered when service composition is available.
In the second scenario the number of services is kept constant as 100 and the number of requests injected into network varied from 10 to 100. Figure 3.10 shows the success ratio with increased number of service requests. All the requests consist of three component services. Again ASGG performs better than the SLIM for the same reason as in the scenario 1.
Since naive method fails to give solution for the request that demands more than one component services for its solution, it is not included here.

**Figure 3.10  Success ratio for requests with three component services**

Figure 3.11 shows the success ratio with increased number of service requests. All the requests consist of five component services. ASGG performs better than the other two.

**Figure 3.11  Success ratio for requests with five component services**
Execution time

Figure 3.12 shows the time required to discover the components and generate the service graph with number of user requests. The performance of ASGG is more or less similar to that of SLIM.

Figure 3.12 Composition execution time as number of requests increases

Figure 3.13 Execution time for requests with 2, 3, 4, 5 component services
Figure 3.13 shows composition time when number of components services is two, three, four and five. The results show that the execution time increases with the number of component services.

3.8 SUMMARY

In this chapter, an input/output dependency based automated service graph generation method for the purpose of dynamic service composition is proposed. The sequence set is generated based on straightforward analysis of input/output dependency. Algorithm was tested with different test cases. For all these cases the output process model is valid. Simulation results show that the proposed algorithm is better in terms of success ratio and execution time. The result also shows that the proposed algorithm is able to achieve around 15 % improvement in the success ratio.