SERVICE ORIENTED

WIRELESS SENSOR NETWORK

by

CAROLYN M. ORTEGA

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment
of the requirements for the degree of Doctor of Philosophy
The City University of New York

2013

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy

Prof. Theodore Brown

Date

Chair of Examining Committee

Prof. Robert M. Haralick

Date

Executive Officer, Distinguished Professor

Prof. Abbe Mowshowitz, City College

Prof. Zhan-Yang Zhang, College of Staten Island

Dr. Dinesh Verma, Distinguished IBM Fellow

Supervisory Committee

The City University of New York

ABSTRACT

Service Oriented Wireless Sensor Network

By

Carolyn Ortega

Advisor: Professor Theodore Brown

Quality of service (QoS) applications and network lifetime management are key challenges in Wireless Sensor Networks (WSNs). Service-level guarantees that are achievable in Service Oriented Architecture (SOA) can ensure reliability, availability and overall usability in any application. How to create the necessary structure to implement SOA in a WSN to achieve the required levels of quality of service is still a research topic. The SOA paradigm provides the methodology to improve these qualities in WSNs. We present a strategy that uses sensors in a cluster and dynamic service selection to achieve higher levels in quality of service for WSNs, while preserving network lifetime. The realization of these strategies will be achieved through a SOA middleware layer in the WSN.

The concept of SOA dynamic service selection is transformed so that it can be applied to WSNs. The transformation requires that we equate a service in SOA to functions that are

performed in a WSN. In our architecture, a service or function will be achieved through a group of sensors called a sensor cluster. Quality metrics will be captured at the sensor cluster level each time it is executed within the WSN.

We present sensor cluster selection algorithms that will determine the most effective and efficient sensor clusters to participate in a composite function plan. The sensor cluster selection algorithms guarantee a minimum level of quality by selecting services that meet a client's quality constraints while taking into account the network routing requirements to achieve the operation in the shortest amount of time utilizing the least amount of energy. Additionally, we present methods for implementing service selection using the multiple choice, multiple dimension knapsack (MMKP) algorithm within a composite function that has complex workflow.

We make use of the WSN's MMKP selection algorithm using dynamic programming as a proof of concept to demonstrate that there is a significant improvement in the energy utilization, network lifetime and overall quality.

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Theodore Brown for his guidance in the preparation of this manuscript. In addition, special thanks to Professor Zhang, Professor Mowshowitz and IBM Fellow Dinesh Verma who provided insightful ideas on the topic during the early phase of this undertaking. Most especially I would like to thank my husband Dan and children, Danny and Dylan for their endless patience, support and understanding during my pursuit of a lifelong goal. I hope that this accomplishment inspires them to never give up on their dreams.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

In order to implement a SOA solution for WSNs, there are several challenges to consider that

do not exist in a wired stationary network like the internet. In a wired SOA system, services are loosely coupled to the data and services are addressable through a well-known interface. Typical sensor networks are designed for specific applications and the data communication protocols are strongly coupled to the hardware [1]. For SOA to work, the decoupling of software from the hardware it runs on is necessary. In WSNs individual sensors are single purpose. Using the concept of sensor clusters and composite functions, we will address the SOA issues of re-usability and loose coupling through abstraction. WSNs are also challenged with constrained computing and network resources which affect scalability, availability and manageability of the WSNs. In wired SOA environments, services operate in a robust computing and highly resilient network infrastructure. Throughout WSN networks, these high bandwidth and managed properties are not the norm; networks within the core operations do exhibit some of these properties, but within hybrid networks of the kind found at the edge, static, resilience and high bandwidth characteristics do not exist. Processing nodes within the hybrid network, that may host services which collect or process data may be resource constrained in processing power, memory, network bandwidth and residual energy [2]. To overcome these challenges in WSNs, we must effectively and efficiently utilize the resources in the network. We must be able to add or remove sensors when necessary. Through sensor cluster configuration, we will select the smallest group of sensors necessary to provide a result to the client that meets the client's quality requirements. Only those sensors that are

participating in the sensor cluster and composite function plan will be used at full power; all other sensors will be placed in a power save mode to preserve energy.

In order to provide reliable and sustainable service from a WSN, fast paced changes must be managed without halting the operation of the WSN service. In this thesis, we present the concept of service-oriented architecture for handling constant change while delivering reliable service in a WSN and maximizing the network lifetime. To meet this goal, we must achieve the following [3]:

1. Be able to dynamically configure services into an existing network
2. Be able to select services based on a desired quality; that is, that are more reliable, accurate or faster, depending on the need
3. Be able to track services as they are added, moved or deleted from the WSN topology. This adds complexity to the dynamic composition problem

The SOA paradigm provides many methods to address the goals for WSNs. The introductory chapter has six sections. The first section (1.1) introduces the methodologies of Service Oriented Architecture and the major components of the Web Service implementation of SOA. In section 1.2 we present the architecture and methodologies of WSNs. The third section (1.3) reviews the concept of Quality of Service, how it is applied to SOA and how these concepts are applied to WSNs. In section 1.4, we review the fundamental technologies of WSN along with the associated challenges of this technology that can be resolved by SOA methodologies. In section 1.5, we present a survey on current literature in the application of SOA to WSN technology. In our final section, 1.6, we present our conclusion to this chapter.

1.1. **Introduction to Service Oriented Architecture**

SOA provides the developer and analyst with the ability to compose applications and processes with reusable services which are published and well-defined using standard and compliant interfaces. SOA makes available the means with which to compose new applications by integrating existing applications and data with services regardless of their platform or programming language. By supplying the programmer with tools that can expose existing applications and data as services, the programmer is able to combine them with other services which can be published in the organization as new processes. These new processes can then be published as services as well.

A service typically performs a function like registering a student, processing a purchase order or monitoring an environment for the occurrence of an event. It can provide a discrete function or a combination of functions. When one or more services are combined to meet the requirements of a complex process, it is called a composite service [4] or composite function. The composite service is accessed by clients through a set of standard protocols and standard technologies. A client communicates with the service through a known interface and relies on the provider of the service to perform all processing and reply with an expected result. The client is not concerned with the platform or the programming language that is used by the provider as all of these details are hidden from the client.

There are several tools that are required to support the service oriented architecture especially as it relates to the use of web services. A Web service is any piece of software that is available over the Internet and uses a standardized XML messaging system. The web service communicates with

clients through a set of standard protocols and technologies. Tools are available which provide the capability to dynamically discover these web services once they are registered with the description of the functionality that the web service provides. There are several standards of Web Service technologies which address interoperability, security and process modeling such as XML, SOAP, UDDI and WSDL. In the area of security we have WS-Security and for process execution we have WS-BPEL [5].

Service Oriented Architecture is an application development paradigm where programming units are structured as services. The services represent a basic unit of work. Services have the following attributes:

1. They are coarse grained. That is, they offer a set of related functions rather than a single function.
2. They are loosely coupled. This means that the service can operate independently of the client.
3. They are reusable. Services can be reused to meet new requirements.
4. They are abstract. Services hide the underlying logic from the outside world. The service interface is the only part of a service that is visible to the client.
5. They are discoverable. The client is able to find a description of the service along with instructions on how to use it. In addition, the service requestor must be able to invoke the service.

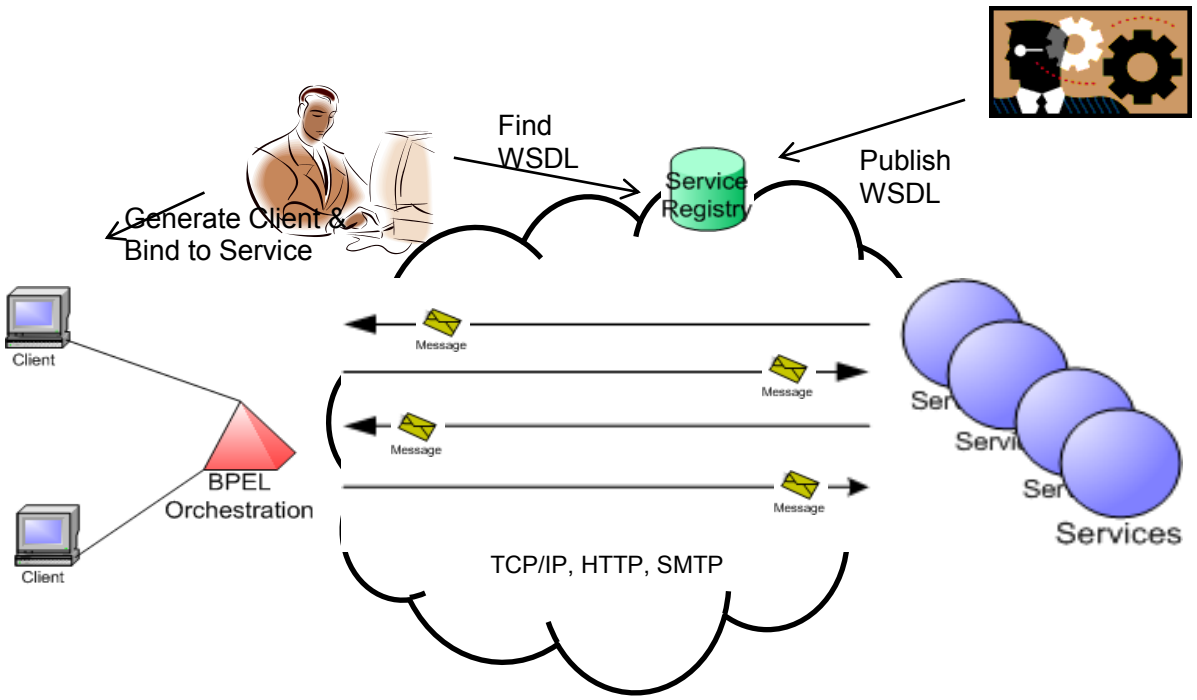


Figure 1: High Level Overview of Web Services

A client invokes a business process using a BPEL orchestration. BPEL contains business process execution commands which consist of multiple (web) services combined with workflow constructs. These services interact with each other via message exchange. The messages use SOAP over HTTP (for web services) to communicate with one another and to pass data. In order to address or locate a service, it must first be discovered. The description and details of how to access a service can be found in a service registry. A typical pattern for use of services is depicted in Figure 1: High Level Overview of Web Services. All of these technologies listed above, use XML the fundamental language for communication. A brief overview of the XML, SOAP, UDDI, WSDL and BPEL technologies are provided in the paragraphs that follow.

1.1.1. XML

XML is a general purpose text-based markup language which is primarily used for the exchange of information between systems. XML is not software, nor is it compiled or executed; it is a plain text language which is used to define other languages through a specification. These documents also have the added benefit, when they are well formed, of being readable and understandable since they are developed in plain text.

The XML specification or schema describes the syntax used to create other markup languages. These schemas specify the grammatical rules that are used to create documents which are well formed. In other words, the schema determines the format and the type (character, integer, etc.) of the elements, the nesting structure and any other associated attributes which are required for the communication between web services and applications. Applications can send XML documents to each other so that information contained in those documents can be processed. This method relies on a common understanding in the applications of the meaning of the XML schemas. Schemas are used to create markup languages which are the foundation for the web services and service oriented technologies: SOAP, WSDL and UDDI.

XML 1.0 and the XML schema standard are defined by the World Wide Web Consortium (W3C). The origins of XML trace back to another standard known as SGML which can also be traced back to a markup language known as GML which was created by C.F. Goldfarb at IBM [6]. The Web Services Interoperability Organization (WS-I) has published the Basic Profile 1.0 (BP) to provide

guidelines and rules to promote interoperability amongst web services. As a part of those guidelines, conformance rules have also been specified for XML.

The structure of an XML document consists of declarations, elements, attributes, text data, comments and possible other components. XML uses tags to identify information and structure data within an XML document. Meaningful tag names will help to associate purpose with information, however, the tags give no inherent meaning to the data. Meaning is associated through common understanding and structure associated with the XML tags amongst applications. The structure that is associated with an XML document is typically referred to as the schema which specifies the grammar rules. Valid XML documents which are created using a given schema must be well formed and conform to the grammar rules of the schema.

XML also has many other powerful java-like features, like inheritance of simple and complex types, polymorphism, and abstraction of types, enumeration, list and union of types as well as anonymous types.

XML schemas provide a convenience in terms of validating instance documents against a template. Parsers are used to perform these validations. JAXP is a JAVA API used for processing XML documents. The XML API's which are used are SAX (Simple API for XML) and DOM (Document Object Model) and either can be invoked in an application to parse an XML document. Both APIs are parsers which parse XML documents and in some cases can validate the document against an associated DTD or XML schema. SAX differs from DOM in the sense that it is read-only, memory efficient and is usually used for large documents. A document is parsed from the beginning with portions of it passed to the application as it is found sequentially. Data is not saved

in memory, so the application must act on the data as it parsed. DOM, on the other hand, is a read-write parser which is memory intensive and typically used for smaller documents. The parser creates a tree of objects in memory that represent the content and organization of data in the document. The application can then navigate through this tree and manipulate the data as appropriate.

1.1.2. SOAP – Simple Object Access Protocol

SOAP is an acronym that stands for Simple Object Access Protocol which is an XML based protocol used for exchanging information amongst web services in a distributed environment. The purpose of SOAP is to provide a common message format for exchanging information and data between clients and services independent of the transport protocol such as HTTP or SMTP. The current definition and widely accepted version of SOAP is version 1.2 which was adopted by the World-Wide Web Consortium Recommendations in June of 2003. The origins of SOAP can be traced back to XML, remote procedure call and distributed technologies [6].

SOAP is another example of an XML markup language whose primary use is for exchanging data and messages over networks. A SOAP document is an XML instance which contains a 'payload' for another network protocol. For example, it is common to transmit SOAP messages via HTTP. However, SOAP messages can be exchanged between applications, carried by email over SMTP or sent over the internet via HTTP.

SOAP specifies the format of the messages that are exchanged between the service requestor, the service provider and the service directory. The basic structure of a soap message consists of a soap envelope which contains an optional header and a mandatory body.

The header element may contain XML elements that describe security credentials, service metadata, data transformation routing instructions or any other information pertinent to the processing of the message. The body will contain the service data which may consists of method parameters and return data. While the SOAP header is optional, the SOAP body in an envelope is mandatory.

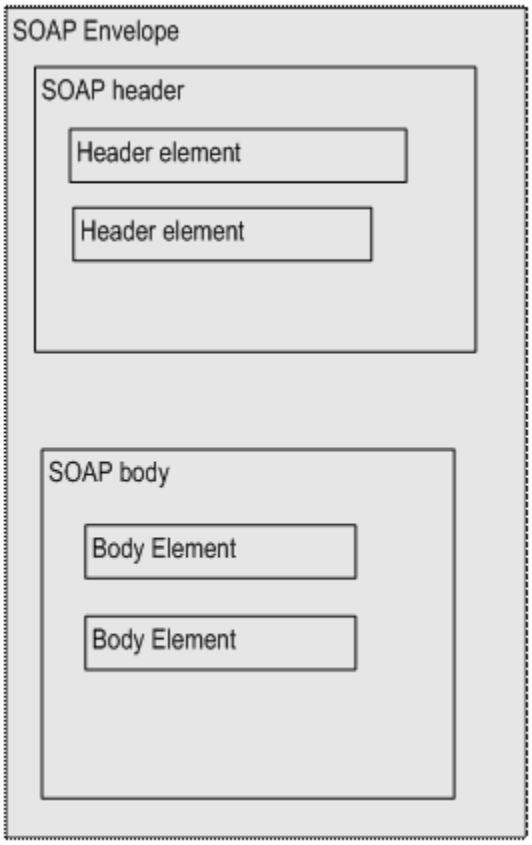


Figure 2. SOAP Envelope

1.1.3. UDDI – Universal Description Discovery & Integration

Service registries support publication and discovery of services. These services may be discovered by searching for categories, keywords or technical capability. There are two types of service registries; the Universal Description, Discovery and Integration (UDDI) and Electronic Business XML (ebXML). In this paper, we will discuss the details of the UDDI.

The UDDI is a specification for registry services. The registry provides a way to catalog web services as well as organizational information in a database that supports standard data structures. The registry contains company and contact information, categorized information for searching and technical information about services offered. The data in the UDDI can be accessed using SOAP like any other web services.

The UDDI provides around 30 different SOAP operations that permit you to add, update, delete and find information contained in the registry. Anyone can set up a UDDI registry for private use in an organization and can provide a catalog of a corporation's software applications in order to provide a centralized source for documentation and discovery. These registries are usually not accessible to the public. There is a massive public UDDI called the UBR (UDDI Business Registry) which is a network of interoperating registries hosted by IBM, Microsoft, HP and SAP. These registries run on 4 separate operator sites under an umbrella organization called the UBR Operators council. Figure 3. UDDI Members illustrates the relationship of the public UDDI registry between the four operators.

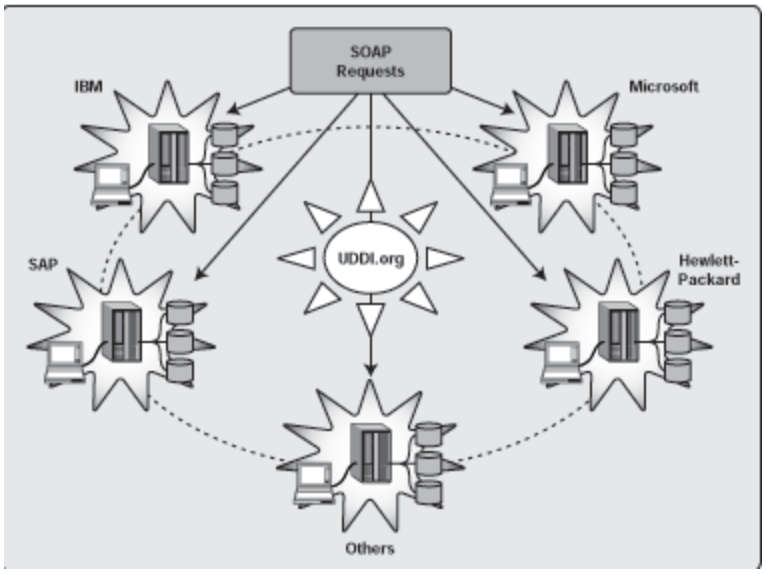


Figure 3. UDDI Members [7]

The UDDI specification in 2000 was developed by IBM, Ariba and Microsoft originally. The UDDI.org formed and invited 12 other companies to develop version 2.0 and 3.0. The UDDI.org turned over the specification to OASIS (Organization for Advancement of Structured Information Standards.) There are several service registry products that are manufactured today by IBM, Microsoft, Sun, Oracle, Fujitsu, Systinet and others. These products run on RDBMS but a registry could also run on LDAP and XML databases.

1.1.4. WSDL – Web Services Description Language

WSDL is a document format used to describe web services. It defines an XML schema for describing a web service as a set of actions which are characterized by a collection of abstract items called ports or endpoints. It describes what a service does, how a service is accessed and where the service is located. A service is described by defining the operations that the service performs as well as the parameters that are used by the service. How a service is accessed is

described by the transmission method and encoding scheme for the service. The location of a service is described by the ports or endpoint URL of the service.

The intent of WSDL is to define an interface definition language (IDL) for web services that is modular and that is not bound to underlying technologies like protocols, programming languages or operating systems.

The origins of WSDL can be traced back to the Network Accessible Service Specification Language (NASSL) defined by the Component systems group of IBM Research as well as SOAP Contract Language (SCL) and Service Description Language (SDL) both defined by Microsoft. NASSL borrows key concepts from languages like MS IDL and OMG IDL which are used to describe RPC interfaces. SCL and SDL are used to support web service based remote models for Microsoft's Component Object Model (COM). The main contribution of SCL to the definition of WSDL was its ability to use different bindings for underlying transport protocols [6].

The current WSDL specification for WSDL 1.1 can be found at www.w3.org. There are many code generators that can read WSDL documents and generate interfaces for accessing web services. These code generators create interfaces and network stubs which can then be used to exchange soap messages with a web service. Some application servers and development environments that support WSDL and follow the specifications for WSDL 1.1 are Apache Axis, IBM Rational Application Developer, Microsoft Visual Studio, BEA Workshop for Web Logic and IONA Artix.

The basic structure of a WSDL document as illustrated in Figure 4. WSDL Format consists of seven (7) main sections which are a part of the definitions element. The seven sections are; types, import, message, portType, operations, binding and service. These sections can be grouped into

two main segments: an interface definition and implementation details. The interface definition segment consists of the portType elements and its sub elements. It describes what the service does. The implementation details refer to the service, binding and port elements. These elements together describe where and how the service works.

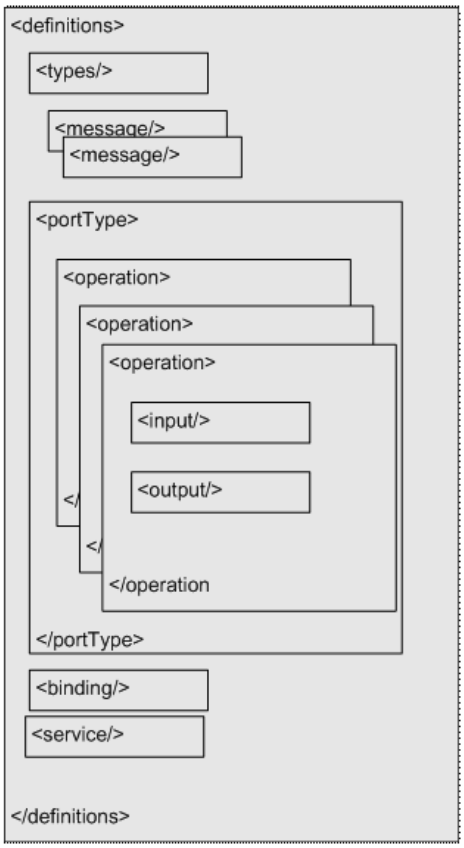


Figure 4. WSDL Format

The <definitions> element is the root element of the WSDL document. It contains all other elements of the WSDL document as well as one or more XML namespace declarations. The type element uses the XML schema language to declare complex data types and elements that are used elsewhere in the WSDL document. The <schema> element is nested within the <types> element and it acts as a container for defining complex and custom simple types. These types are then used

by message definitions when declaring parts of messages that become the 'payload' in the <portType> <operation> section.

The <message> element describes the input and output parameters and is also sometimes referred to as the 'payload' of the service. The import element is used to make definitions specified in the namespace of another WSDL document available in the current WSDL document. The service or abstract interface includes the <message>, <portType> and it's defined <operation> elements. It defines the abstract interface of a web service but does not define its implementation.

The implementation section of a WSDL document consists of the binding, service and its related port information. This section defines the service endpoint(s) and data format used to communicate with the endpoints. SOAP messages will be created based on the WSDL document definition.

1.1.5. WS-BPEL – Web Services Business Process Execution Language

BPEL is an acronym for Business Process Execution Language which provides the ability to combine or orchestrate multiple discrete business functions into a single business process which can then be published as a web service. As these services are combined, the workflow of the process can be implemented using the constructs provided in BPEL. BPEL provides a mechanism for visually defining and managing the interactions between web services. It supports the flexible composition of services so that they can be easily modified to enable process change. A process can be converted to a web service which can then be invoked repeatedly by partner services. With BPEL, the process orchestration is decoupled from the implementation. BPEL also supports long-

running business processes through state management and provides error recovery mechanisms so that activities can be reversed in the event of process errors.

These business processes are considered to be a coarse grained service since the business processes are able to handle more than one operation. A fine grained service can only handle one operation. For example, a coarse grained service could handle all of the processing of a student registration where a fine grained process may be a component of the overall process like adding a course to the student schedule. Several BPEL processes can be combined with their associated workflows implemented creating new business processes which can then be offered as a web service as well. This promotes the development of coarse grained web services by composing processes that can invoke other web services, handle long running transactions and deal with exception faults appropriately [8].

The first specification of BPEL was BPEL4WS which stands for Business Process Execution Language for web services. The origins of BPEL can be traced back to WSDL, XLANG, WSFL and XML. IBM and Microsoft developed the first BPEL4WS back in 2002 which consolidated two languages; WSFL (Web Services Flow Language developed by IBM) and Microsoft's XLANG. WSFL was the next generation language that was derived from IBM's Flowmark. This language follows closely the reference model that was defined by the WfMC (Workflow Management Coalition) [6].

There are several BPEL implementations on the market such as IBM's Websphere, Microsoft's BizTalkServer, Eclipse's BPEL Designer, Oracle's BPEL Process Manager and ActiveEndpoint's ActiveBPEL, Apache's Agila and Process eXecution Engine to name a few.

There are 4 major sections to a BPEL process: Partnerlinks, Variables, Faulthandlers and the Process. Partnerlinks represent the links, interactions and dependencies to all processes and web services interacting within a BPEL process. A partner can be a provider of a service, a consumer of a service or both. Partnerlinks are used to establish the peer to peer conversational relationship between partners through portTypes. PartnerLinkTypes define the roles which characterizes the conversational relationship between two services.

Each partnerlink is characterized by a partnerLinkType and more than one <partnerLink> can be characterized by the same partnerLinkType.

Port types provide an abstract definition of functionality using abstract messages. Ports provide actual access information including community service endpoints. The fundamental use of endpoint references is to serve as the mechanism for dynamic communication of port specific data for services. An endpoint reference makes it possible in WSBPEL to dynamically select a provider for a particular type of service and to invoke their operations.

Business processes consists of message interactions between partners where each of these interactions represents a state. The state of a business process includes the messages that are exchanged as well as intermediate data used in business logic and in composing messages sent to partners. In order to capture, maintain and manipulate the state of a business process, the use of variables is required. The manipulation of data from the state is necessary to control the behavior of the process hence the need for expressions. Finally state update requires a notion of assignment. Both executable and abstract processes are permitted to use the full power of data selection and assignment.

Variables provide the means for holding messages that constitute a part of the state of a business process. The messages held are often those that have been received from partners or are to be sent to partners. Variable types may be schema types, schema elements (simple or complex defined in the WSDL-> Schema section) or WSDL messages. Message type variables are used as input/output data for the invoke, receive and reply operations.

A fault handler defines the activities that must be performed in response to unanticipated or erroneous events which can occur during execution of the process. The Process element is the root element of the BPEL process. It contains the partnerLink definitions, the variables and another important element known as the sequence. The sequence element acts as a container for activities. Each activity will be performed in the order defined within the sequence element and when all activities have completed, the sequence element is completed. The flow element also acts as a container for activities. These activities are typically performed in parallel.

There are several activities which are possible in BPEL. For communication, the receive, reply and invoke activities are used. For message manipulation and validation, the assign and validate activities are used. BPEL has conditional and looping constructs which can be used such as For Each, If, Repeat Until, Pick and While. Container and Flow activities are defined as sequence, flow and scope. There is a rich definition of activities for fault and event handling. They are: catch, catch all, compensate. Compensate scope, empty, exit, rethrow, throw and wait.

Before leaving the description of the BPEL specification, it is worth mentioning the concept of correlation. Since BPEL has the capability to support long running processes, it is possible that the sequential behavior of a process not be followed unless compensating activities are introduced to

handle the sequential process. Correlation ensures that request and responses are correctly associated within a process in the appropriate order.

1.2. Introduction to Wireless Sensor Network

A WSN consists of spatially dispersed, independent sensors devised to work together to monitor physical or environmental conditions. These sensors can observe or measure characteristics of its surroundings ranging from distance, height, width, temperature, motion, vibration, force, sound, pressure, strain, pollutants, chemicals and more. The sensors communicate wirelessly with the intent to extract relevant environment information and report back to a central location. In some sensor networks, there are also nodes which have the ability to perform actions within the environment. These nodes are called actuators [9].

The development of WSNs was motivated by military applications like battlefield surveillance, and is now used in a growing population of industrial and civilian application areas including industrial process monitoring and control, machine health monitoring, environment and habitat monitoring, healthcare applications, home automation, and traffic control [9].

1.2.1. Fundamentals

A sensor network is usually a wireless ad-hoc network, i.e. each sensor supports a multi-hop routing algorithm where nodes function as forwarders, relaying data packets to a base station. Each

sensor node is typically battery-powered, and consists of a processor, sensor, transceiver and other modalities [10]. WSNs involve three types of sensor nodes:

1. Common nodes – nodes whose primary responsibility is to collect sensor data. Occasionally, common nodes will work collaboratively with neighboring nodes to complete a task.
2. sink nodes – responsible for receiving, storing and processing (e.g. Aggregation) data from common nodes
3. gateway nodes – connect sink nodes to external entities

Sensors can vary in size, cost and function. Size constraints on sensors result in corresponding constraints on resources such as energy, memory, computational speed and bandwidth.

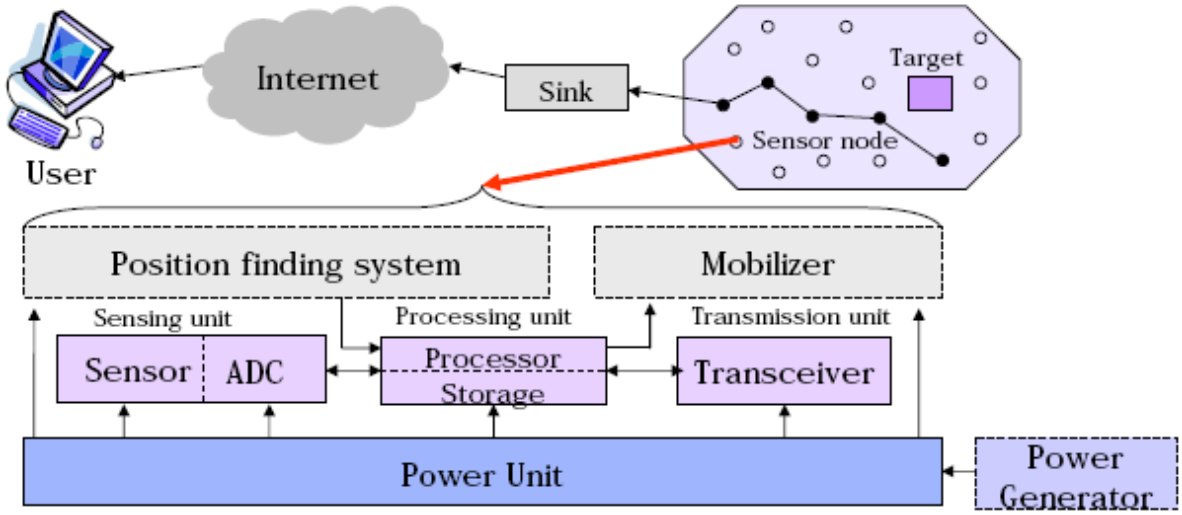


Figure 5: WSN Architecture [10]

There are four basic units in a sensor: A sensing unit, a processing unit, a transceiver unit and a power unit. The power unit supports all activities on a sensor, including communication, local data

processing and sensing. The lifetime of a sensor node is mainly determined by the power supply since battery replacement is not an option in sensor networks. The network topology of a WSN represents the actual sensor placement and the reach ability of sensor nodes within the network. The network topology of WSNs may be dynamic due to frequent node changes. These node changes may occur because sensors become unavailable due to lack of energy or physical destruction. In addition, new sensors may also join the network. Therefore the network must be able to reconfigure itself periodically [11].

1.3. Quality Attributes in SOA and WSNs

Quality attributes in SOA and web services is a well-researched and defined topic. Through the definition and measurement of quality attributes for SOA and web services, objectives are set for levels of acceptable function for a service using these quality measurements. The client can determine which levels of quality are required and factor this information into their decision on which services to use. Following is a survey of relevant definitions in SOA and web services and their relationship to WSNs.

1.3.1. Performance

Performance is a quality measurement that can be characterized in many ways. The W3C group defines performance as follows: “The performance of a web service represents how fast a service request can be completed. It can be measured in terms of throughput, response time, latency, execution time, transaction time, and so on” [12]. The objective is to achieve the maximum

performance in a system and hence maximize the performance measurement. These five quality attributes are reviewed in the paragraphs that follow.

a) Throughput

In web services, throughput is defined as "the number of web service requests served in a given time interval." [12]. Taher defines maximum throughput as "the maximum number of simultaneous requests of web services serviced in a given period with guaranteed performance" [13].

In WSNs, the throughput measurement must take into account that multiple sensors may contribute to the result for a request for a service. "Information throughput at the sink should be derived from a set of correlated sensors instead of an end-to-end data throughput for individual sensors" [14].

b) Response Time

In web services, both the W3C and Taher simply describe response time as "the time required to complete a web service" [13], [12]. In WSN's "The response time is the period from the start time of query processing to the time when all of the nodes have reported their query results…" [15].

c) Latency

Latency is another performance related measurement which, in web services is defined as "the round trip delay between sending a request and receiving the response" [12].

From a WSN perspective, we must take into account the fact that many sensors participate in an operation and so the response time of all sensors must be taken into account. Collective latency "is defined as the difference between the time at which the first packet related to this event is generated by the source sensors and the time at which the last packet related to this event or the last packet used to make a decision arrives at the sink" [14].

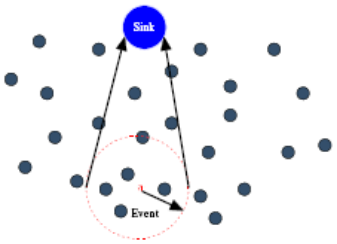


Figure 6: Sensor Node Group – Collective Response [16]

d) Transaction Time & Execution Time

Transaction time for web services is generally defined as the time that passes while the web service is performing one operation. Execution time has been defined as the time for the web service to process its sequence of activities which may involve many transactions [12]. The same definition in web services can be used for WSNs, however the fact that many sensors may participate in a transaction or operation must be taken into consideration.

1.3.2. Availability

A web service should be ready for immediate consumption. The availability quality attribute in web services is the probability that the system is up and ready to provide service. [12]. The

availability metric for web services is also defined as the probability that the service will be available at some period of time. Larger values represent that the service is always ready for use while smaller values indicate unpredictability of service availability at a particular time [13].

In WSNs availability is similarly defined as the time the operation was accessible during the total measurement time [16].

1.3.3. Accessibility

The quality attribute, accessibility, represents whether or not the web service is capable of serving the clients requests. High accessibility is achieved by building highly scalable systems [12].

Accessibility may be expressed as a probability measurement denoting the success rate or chance of a successful service instantiation at a given time. There could be situations when a Web service is available but not accessible because of a high volume of requests. High accessibility can be achieved, by building highly scalable systems. The accessibility metric is typically provided by the provider. This definition can be used to describe accessibility for WSNs.

1.3.4. Accuracy

Accuracy is defined as the error rate generated by the web service. The error rate is the number of errors that the service generates over a time interval. In WSN, the error rate is related to packet loss among many sensors. Collective packet loss is defined as the number of packets related to an

event that are lost during information delivery. The measurement of collective packet loss will determine the level of accuracy for a group of sensor nodes [14].

1.3.5. Reliability

The quality attribute reliability is defined as the ability for a web service to perform its required functions under stated conditions for a specified time interval. The overall measure is related to the number of failures per a specified period [12]. In WSN, reliability is defined as the number of times that an operation has been successfully executed divided by the number of times the operation has been invoked [16].

1.3.6. Capacity

The quality attribute capacity is defined as the number of simultaneous request which should be provided with guaranteed performance [12]. In WSN, capacity is associated with the bandwidth of a group of sensors that provide a response to a request for service. Collective bandwidth is defined as the bandwidth that the reporting of the event requires. The sink should be concerned about an end-to-end event of a group of sensors instead of the packets from individual sensors [14]. The definition is similar to throughput.

e) Cost

For both web services and WSN, the cost of a service is defined as the dollar amount to execute the operation [16]

1.3.7. Energy Consumption

The energy consumption quality attribute is only defined for WSNs. There is no equivalent quality attribute in web services. The closest related attribute in SOA to residual energy is price. Energy consumption is defined as the amount of energy required to perform the service. Residual energy which is closely related to energy consumption is defined as the remaining energy of a node in the wireless sensor network. "The residual energy must be greater than the amount of energy (energy consumption) needed for the service execution" and transmission. [16]

1.4. WSN Challenges That Can Be Resolved with SOA

There are many challenges in WSNs which can be addressed by applying SOA principles in a WSN. Following is a summary of these challenges.

1.4.1. Rapid Application Development

One challenge that is found in sensor networks is that the sensors in the network are typically application specific. This means that the sensors are designed for one purpose and are inflexible to change as a result [17], [11]. SOA resolves this issue since a service is loosely coupled to the hardware enabling flexibility and reusability of the service.

Heterogeneity of the operating environment is another application specific challenge. Without having a middleware layer to negotiate communication between heterogeneous hardware and operating systems, combining services across WSN platforms or adding heterogeneous sensors to

the network would require extensive programming. [18] In addition, WSNs typically have the data communication protocols coupled within the application. [17], [19]. This is an example of tight coupling of the hardware with the application. SOA resolves this issue through middleware that negotiates the communication between different hardware and operating system platforms. As mentioned previously, services are loosely coupled which provides reusability of services, adaptability of new services and scalability of services. [4].

Another major challenge in WSNs is the inflexible application development and deployment methodology. Since typical WSN applications are tightly coupled rapid application development is practically impossible. A framework for rapid application development which addresses the dynamic reconfiguration of the WSN is necessary [18], [9], [20]. In SOA the architecture is inherently loosely coupled allowing for flexibility and reuse of services which promote rapid application development. The inability to address the changing needs of the application environment quickly affect quality issues such as adaptability, scalability, reliability and reusability.

1.4.2. Network Lifetime Management & Network Reliability

It is typical in a WSN to have limited resources like memory, battery life and computing resources. [2], [18], [11]. Some WSNs also have challenges like limited transmission power, buffer size limitations and energy depletion [14]. Another challenge in WSNs involves the network topology and in some cases, reachability of sensor nodes is sporadic [21]. There is no equivalent issue in SOA regarding limited resources as most computing sources have adequate power and are reachable over an intranet or the internet. However, the network reliabilty challenges experienced

in WSN affect availability, reliability, scalability and reusability of network resources from a quality perspective so these are WSN issues that must be monitored and addressed.

1.4.3. Fault Tolerant Communication

In WSNs, fault tolerant communication is a challenge as a result of sensor mobility and access difficulty due to external factors [17], [18], [11]. In addition, there are no formats or standards for data measurement leading to inconsistencies and incompatibilities in measurements derived from different sensors [18]. SOA addresses these issues through the use of defined standards for data that is transmitted and for the communication network between services. Without these standards reliability, security, accuracy and reusability are affected from a quality point of view.

1.5. Survey of Research on WSN-SOA Implementations

A great deal of research has been performed on how to implement SOA in a WSN. The research can be grouped into three areas: the application layer, the middleware layer and the network layer. In this approach the network layer consists of the sensors, the middleware layer provides the communication from the network layer to the application layer and the application layer is where the business process execution language is managed. The end user can invoke the service at the application layer and the application layer communicates with the hardware (network) layer via the middleware layer. In other words, the middleware layers acts as an interface to the hardware layer. It provides seamless communication for the developer to manipulate the sensors without knowing the details of how to call the hardware services. It provides an abstraction to the developer so that different types of sensors using different communications protocols can be easily accessed

through the network layer without the developer writing code specifically for each sensor device. The middleware layer will handle the translation of the developer's commands to a language that each sensor can understand. Many researchers have approached implementing SOA for WSN's by developing solutions for one or more of these layers. A summarization of the layer specific research is presented in the paragraphs that follow.

1.5.1. Application Layer

Delicato describes the need to have application specific features separated from the data communication protocol in WSNs [17]. In his approach he combines features of WSDL and data centric communication protocols like directed diffusion [22, 23]. This approach is similar to the OSGI framework described by Prinsloo [9]. Delicato uses a Web Services approach for the design of sensor networks, in which sensor nodes are service providers and applications are clients of such services. The author envisions the future sensor networks as being composed of heterogeneous sensor devices servicing a large range of applications. To achieve this goal, a new architectural approach is proposed, where application specific requirements are separated from the data dissemination functions.

In sensor networks, the same data can be transmitted by many sensors. In Delicato's model, sensors that send the same data can be aggregated to reduce redundancy and minimize the traffic on the network. Delicato also uses an addressing scheme for the nodes on the sensor network where nodes are identified by the data generated by them or by their geographical location. A special data centric protocol called directed diffusion is used in Delicato's sensor networks. Data dissemination protocols like SPIN [22] or directed diffusion provide an application independent

solution. Most networks rely on IP-based communication and do not consider dynamic and resource constrained environments like sensor networks. Service location protocols (SLP) [17] facilitates discovery and use of heterogeneous network resources using centralized discovery agents.

WAP [24], performs the function that WSDL performs in web services. WAP uses a compact binary xml content format called WBXML. It is intended to reduce the size of xml documents for transmission and simplify parsing them. In Delicato's sensor network there are two main physical components: sensor nodes and sink nodes. Sensor nodes contain one or more sensing devices and they have aggregation capabilities. Sink nodes provide application interfaces through which external systems can obtain information collected by the sensor network. They can aggregate data but do not have sensor devices.

The operations defined for the WSN Web services specified in Delicato's system are:

1. Publish_Content, used by the sensor node to create and disseminate a SOAP message containing its service descriptions;
2. Publish_Data, used by sensor nodes to create SOAP messages communicating generated data;
3. Subscribe_Interest, used by an application to submit a query to a sink node;
4. Subscribe_Filter, used by an application in a sink node to inject a new filter in the network.

Another SOA approach to WSN's is Prinsloo's Ceberus project [9]. Ceberus is a WSAN home security application. During the development of this application it became clear to the authors [9] that a framework that promoted rapid application development of WSAN software applications

was necessary. Generic Application Development Framework (GADF) will allow developers to deploy motes in an environment and then quickly and easily develop an application to extract the information from the WSAN and use it in a way corresponding to the specific application under consideration. Several WSN research projects are related to the Ceberus project. Atlas is a service oriented sensor and actor platform [20] that encompasses concepts of self-integrative, programmable pervasive spaces. It has a combination of hardware and software elements that allows sensors and actors to expose themselves as services to other components. Atlas was used in the Gator Tech Smart House [25]. The Gator Tech Smart House provides 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 be then be invoked

The Ceberus Project developed a complete working system with sensor network simulators before deploying the required code on the actual motes. MICA2 and MICA2DOT[1] motes were used as the senor nodes for the implementation along with the execution model, component model and communication mechanisms supplied by TinyOS (v1). Code for the motes was written in nesC [26] and tested on the TOSSIM simulator [27]**.** Dynamic configuration of certain aspects of the motes behavior and the sensing of environmental data threshold value can be set and dynamically changed for each sensor on a mote. The mote only transmits data when a threshold value is exceeded. Mote state can also be dynamically changed by sending a mote a control message which is able to put it into a sleep or sensing mode. The Cerberus server is implemented as a management bean in JBoss (J2EE application server) that handles all of the communication with

[1] MICA2 and MICA2DOT are motes which are commercially available from Crossbow Technologies Inc. (http://www.xbow.com)

the WSAN including the retrieval of all information from the sensor network and the dynamic configuration of motes. For the web interface, Tomcat was used as the web container and requests were also sent to the server through web services. A stand-alone client also contains additional functionality which allows the user to create rules for the sensing environment and trigger certain actions for a specific rule.

1.5.2. Middleware & Network Layer

In order to implement SOA architecture for WSNs the need for a system abstraction layer or something similar arises that will conceal the low level details of the hardware and its functionality to the developer. One possible solution is the creation of middleware that will supply higher level abstraction of the lower level functionality. Another related problem is that it is difficult to extend or modify a WSN to accommodate changes once deployed. In the absence of an adequate mote management system, the software to acquire access to motes and their information in a WSN generally has to be developed from scratch. As a result, the software is not developed in the context of a generic architecture and is consequently not suitable for reuse. In order to make architecture of this type work, standardization in its implementation must be developed to maintain consistency in application development created by different groups. It is therefore necessary to have a standardized middleware layer to provide the generic architecture for the developer to communicate with at the application layer. The following is a brief summary of research in the WSN middleware layer.

Mead proposes a model driven development paradigm where application developers specify a domain specific application structure and a domain specific modeling language to create domain

specific system models. He introduces the notion of a surrogate (intermediary) to represent a device as a Jini service [2] . The surrogate resides in a host on the fixed network and advertises its service using standard Jini protocols. A client's request for service is routed to the service surrogate who may communicate with the device using inter-connects. The most notable features of Mead's middleware toolkit are: the configurable inter-connect, allowing seamless switching between different interconnects at runtime and a switching process that can be initiated proactively.

In Pandey's solution, he developed SOA –like middleware for WSNs with a goal to reduce the load processing of WSN components through the use of a notification services system for events that a client requires for presentation of measured data. Pandey uses an "Ambient Programming Model with the ported code in GALS (Globally Asynchronous, Locally Synchronous) mode using TinyGALS given by Tiny OS." [18]. The middleware component consists of object code, a data filter box, messaging server, a database management system and a business integration framework. The nodes of the sensor network use object code that follows an ambient programming model allowing two way asynchronous communications between nodes tolerant towards communication failures. The data filter box contains intelligence to separate out normal and exceptional events in the network. The messaging server contains message queues that store messages by topic. A database management system is used to store the data coming from the data filter box. The goal is to reduce the load on the processing components in the WSN by introducing a notification services for events of interest. The middleware receives data input from the ambient sensor network. The event sources are the sensor nodes which are programmed for detecting a temperature above or below certain level. The gateway passes this data to a data filter box which will be an interface between the object code and middleware. The Client has a web interface for looking at warning

signals and any time can enquire about the current scenario. The benefits of Pandey's middleware is to provide a data filter box that filters lots of unwanted events thereby reducing processing load in the WSN and to direct critical data to the Java Messaging Server (JMS) providing asynchronous messaging components that can redirect messages to clients through a series of processes [18].

As mentioned earlier, King has developed Atlas, a commercially available service oriented sensor and actuator platform that enables the concepts of self-integrative, programmable pervasive spaces using Atlas hardware and middleware components. The middleware component consists of a combination of hardware, firmware and software middleware that provides services and an execution environment. The majority of the middleware framework operates on a stand-alone server and uses OSGI as the basis for middleware [20].

Marin-Perianu defines a three-layer SOA architecture which consists of the application layer (back-end), a platform abstraction layer (the gateway) and a device layer (the front end). The back-end layer provides business applications access to services offered by the WSN at the level of Web Services. The gateway layer serves an essential role in matching different sensor platforms. The key feature of the gateway is the dynamic instantiation of service proxies. The front end layer encompasses the large number of WSN and RFID technologies. The SOA architecture has three main operations: Service Lifecycle management, Service Invocation and Event Notification.

1.6. Conclusion

It is clear that the research community recognizes the need for applying the SOA paradigm to WSNs. We aim to develop concepts and models to achieve dynamic composition of services in WSNs.

Chapter 2

WSN QoS Objectives

Improving the quality of WSNs is our primary objective. To improve the quality of WSNs we must define the levels of quality of service that are acceptable and expected. We must be able to select services based on these quality requirements and monitor the delivery of service from the WSN to ensure that the quality levels are being met and maintained.

In this chapter we will review five qualities of service objectives for a WSN. In section 2.1, we will review the reliability QoS attribute and discuss its relevance in a WSN. In section 2.2, we will examine the network lifetime and residual energy as a quality attribute for a WSN. In section 2.3 we will study the importance of achieving re-usability and loose coupling as a quality attribute in a WSN. In section 2.4 we will examine execution time as a quality attributes and in section 2.5 we will discuss the desire for higher levels of accuracy in a WSN. In section 2.6 we provide a conclusion to the chapter.

2.1. **Improve reliability**

Reliability in SOA refers to the integrity and dependability of a service over time. Message reliability is of particular concern in SOA as it relates to the dependability of message exchanges between applications and services [28]. Message reliability left to the application developer to incorporate within the service may be implemented in various and inconsistent ways across services which could result in unreliable end-to-end messages.

In addition to message reliability in SOA, we also must ensure the reliability of the actual service; that is, we must guarantee the integrity and dependability of the service. These guarantees may be provided in a Service Level Agreement (SLA) between the provider and the consumer. In spite of the SLA, problems may still occur which will require remediation in the case that a service fails. In some cases, failure is critical to the process and the reliability of the service is of extreme importance. In these cases, trust and SLAs may not be enough as quality control is essential. Additional research may be required to not only enforce the SLA's but to monitor the service prior to failure occurring as well as to detect a failure in a reliable service once it has occurred.

This problem is comparable in WSNs as it relates to communication and reachability. Pandey indicates that "volatility of communication may occur due to mobility or other external factor" within a WSN [18]. Yu writes that "nodes may fail (either from lack of energy or from physical destruction), and new nodes may join the network. [11]." In order to provide guarantees of service in a WSN we must be able to guarantee the reliability of the services provided. To improve reliability in a WSN, we must select services that have a track record of reliable service. These services are resilient, reachable and have adequate resources to perform the task necessary [11] , [2], [18], [14].

2.2. Improve Network Lifetime and Residual Energy

SOA and web services do not have the challenge of limited energy or network lifetime that are exhibited in a WSN. Consequently, there is no comparable research or methods available in SOA to address these limitations. In a WSN nodes have limited resources like memory, battery life and computing resources [2], [14]. Therefore, residual energy of a sensor is extremely important to the

longevity of the network. Residual energy can be used to determine if a service group can achieve the requested service. While there is no comparable research in SOA for improving network lifetime it is the aim of this research to prove that introducing the selection of appropriate services for execution will help to achieve efficiencies in the use of sensors and hence improve the network lifetime. By selecting services that have the appropriate energy levels to participate in the service execution, we will eliminate costly communication failures, re-execution of request and also improve the reliability of the WSN.

2.3. Improve Reusability and Loose Coupling

Operability, deploy-ability and interoperability are all quality attributes of a service oriented environment that describes the Operational Capability of the environment. In SOA these quality attributes are realized through the definition of standards, the loose coupling of services and the reusability of services. To achieve these goals, a new architectural approach is required that will separate application requirements from the hardware in WSNs [17], [11]. The heterogeneity of the WSN operating environment should appear seamless to the developer [1], [19]. A new development and deployment framework for WSNs that promotes rapid application development and eliminates manual integration of sensors is needed [18], [9], [20].

2.4. Improve Execution Time

Selecting services based on their execution time will save residual energy while meeting the needs of a client. This is achieved through selection of services that are the closest match to a client's

response needs. [It is the concept of providing just enough service to the client at the time of need and not exceeding what is necessary.] It may be the case that a client does not require the fastest response time for a particular service. It is also possible that an average or lower response time can be adequate to meet a client's needs. The benefit for the client in this scenario is that the cost of the service may be lower if the response time is conservative and better aligned with a clients' need [29].

2.5. Improve Accuracy

When a client selects a service, the level of accuracy or precision that the service group can provide should be known. A client may not need an extreme level of accuracy in response to a service request. This may result in a smaller number of sensors participating in a service group [30], [17]. These decisions, affect cost, residual energy and possibly network lifetime and reliability. Making the right choice for the number and type of sensors required could potentially improve the quality overall in the WSN and increase the network lifetime. Take for example the k-coverage problem in WSN. In an over-deployed sensor network, one approach to conserve energy is to keep only a small subset of sensors active at any instant. If we are able to select the minimum size of sensors to perform the required function, known as the K-cover optimization problem, it is possible that we conserve energy in the network [31].

2.6. Conclusion

As mentioned earlier, a great deal of research on quality attributes and how to achieve quality in SOA and web services has been performed and this is evidenced by the vast developments in the

standards and specifications realm for web services. The quality attributes and how to achieve the desired levels of quality in WSNs still requires further definition as well as the development of standards and methodologies that can be realized in WSNs.

One objective of this research is to provide the framework for achieving SOA in WSNs. From this framework, the definition of standards can be developed to achieve the interoperability, reusability and efficiencies achieved in web services in a WSN. Identifying the primary objectives of quality for a SOA WSN is necessary to define the QoS standard for operation.

Chapter 3

SOA Strategies for WSN

The promise of SOA is to promote easy modification to systems through simplified application development and to remove the barriers of working with disparate operating systems, hardware platforms and data environments. These improvements are realized by viewing software and data as a service. SOA is an architectural methodology for developing solutions based on a set of services.

In order to implement a SOA for WSN, we must be able to view the functions and data that are captured by the sensors as a service [17], [20], [32], [19].

In chapter four we discuss several strategies for applying service oriented architecture concepts to a WSN. In section 3.1, we will look at a fundamental construct in SOA known as the service. We will define an equivalent construct known as the sensor cluster to be the fundamental service element in a WSN. In section 3.2, we review an important service oriented architecture feature called service orchestration. Service orchestration allows us to reuse services by constructing new processes from existing services, dynamically. In section 3.3, we discuss network work flow patterns. Since our aim is to compose new processes dynamically, workflow is a critical feature for defining new orchestrations. One of our primary objectives is to improve quality in a wireless sensor network. To achieve this objective we must be able to measure and calculate quality in the WSN. In section 3.4, we provide the necessary formulas to compute QoS for various attributes in a WSN. In section 3.5 we provide the conclusion to this chapter

3.1. Equating Sensors to Services

Services are the basic building block in a SOA. A service is a self-contained unit that performs a specific task that is reusable, non-context specific, stateless, and can be dynamically discovered across the enterprise, in partner systems, or in the cloud. It is treated like a black box that performs a function where the client or invoker of the service does not need to know about the implementation details within the service. The service encapsulates all of the details so that the only information that is required to execute the service is to know what function the service performs, how to call or invoke the service and what to expect as a result.

3.1.1. Sensor Clusters

In order to achieve these desired features in a WSN, we must be able to equate a service in SOA to functions that are performed in a WSN. In our architecture, a service or function will be achieved through a group of sensors called a sensor cluster as illustrated in Figure 7: Sensor Cluster. A sensor cluster is capable of executing a specific task. The details of how the task is performed are hidden from the user. Each sensor cluster consists of a sensor cluster head or manager and several sensors. Sensor cluster heads are entry points for the application or are data gathering points for a collection of sensors. Information about the sensor clusters will be stored in a repository. The sensors within the sensor cluster are one hop away from the sensor cluster head.

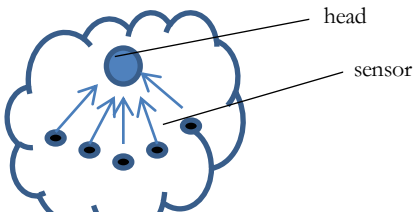


Figure 7: Sensor Cluster

3.2. Service Orchestration

In a SOA, services can be combined with other available services in a network, through a process called service orchestration, to create higher-level composite services and applications. Services provide the ability to develop new applications rapidly. The newly formed application is called a composite service [4].

A composite service is a collection of services connected via workflow constructs like those found in a workflow language like Business Process Execution Language (BPEL) to automate a multi-service process.

3.2.1. Composite Function Graph for WSN

In SOA, a service composition represents a process that consists of workflow and services to achieve a particular goal. Jaeger describes the distinction between service composition workflow at build time and at run time [33]. At run time, a dynamic binding of services in the service composition can be performed. However, during the build time, services can be selected for the composition. We provide a more formal definition for the service composition during the build time for WSNs. A Composite Function Graph (CFG) [32] consists of an orchestration of functions representing a process in a WSN. A CFG is an abstract representation of the process workflow (Figure 8: Composite Function Graph). The CFG describes the functional process flow without having actual sensor clusters selected for each function.

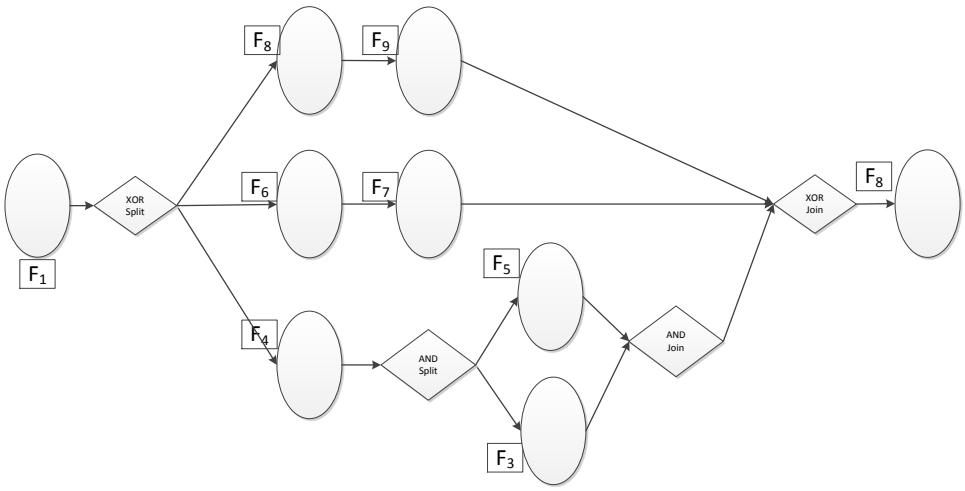


Figure 8: Composite Function Graph

A CFG is a blueprint for a process. It cannot be executed. Sensor clusters must actually be selected to perform each function in order to execute the process. Once sensor clusters are selected for each function in the CFG, the CFG becomes a Composite Sensor cluster Instantiation (CSI) (Figure 9: Composite Sensor Cluster Instantiation) [32] . At this point, the CSI is executed to provide the desired results.

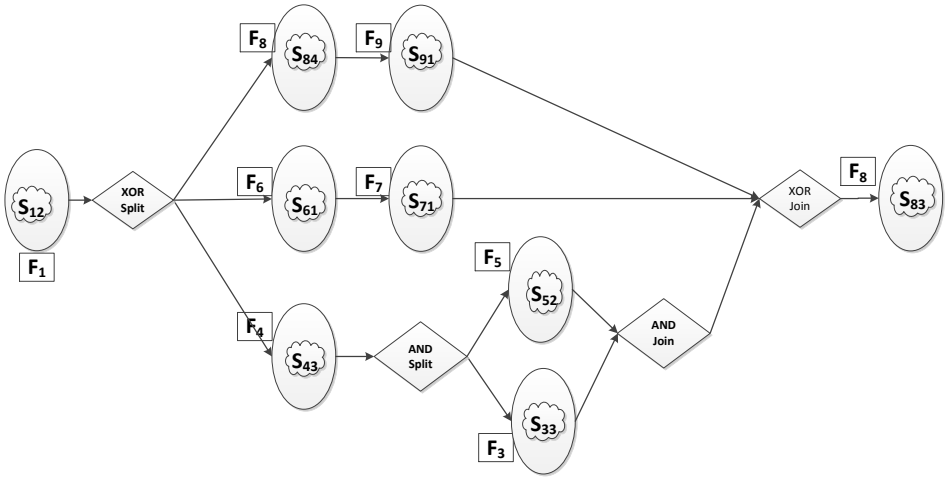


Figure 9: Composite Sensor Cluster Instantiation

3.3. QoS Network Workflow Patterns

The workflow operations consist of basic operations, simple conditional operations and complex conditional operations.

3.3.1. Basic Workflow

In basic workflow operations are sequential. The workflow may either be a sequential sequence of sensor clusters or a loop of sequential clusters. Basic sequential workflow consists of one or more sensor clusters that operate one after the other in time without any conditional operations between them. Figure 10: Sequential Sensor Clusters is an example of sequential workflow.

Figure 10: Sequential Sensor Clusters

This definition is true with the exception of a special case: loops. Loops are considered to be iterations of sequential flow. The loop construct is considered to be a special case because at the end (or the beginning) of the sequential workflow a condition may exist that will determine if the number of iterations of the sequential workflow is complete. In a basic sequential workflow, we can assign a variable k, to represent the number of times the sequential workflow will execute. Since the sequential workflow will operate k consecutive times, without any parallel operations,

we consider the basic loop to be a sequential operation. We assume that k is a known value at the time of sensor cluster selection.

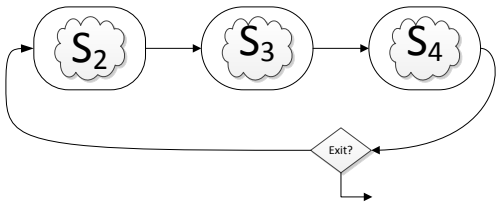


Figure 11: Loop of Sensor Clusters

3.3.2. Conditional Workflow

In conditional workflow, a choice is made to determine the path(s) of execution. Based on the outcome of the condition a single path or multiple paths can be executed and then merged. The conditional workflow operations are AND-Split followed by an AND-Join and XOR-Split followed by an XOR-Join.

3.3.2.1. AND-Split

In the AND-Split, if the condition is true, a single thread of control separates into multiple threads of control. In the event that the AND-Split condition is true, both paths will be executed. The separate threads of the AND-Split can be executed in parallel.

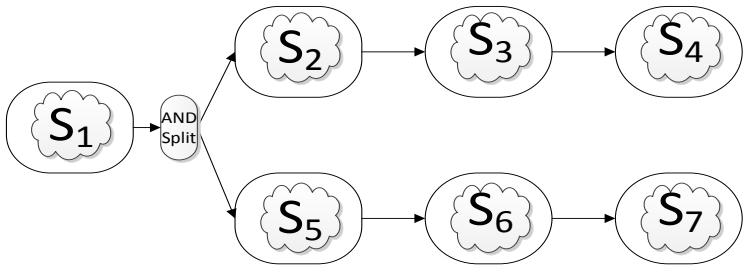


Figure 12: AND-Split of Sensor Clusters

3.3.2.2. **AND-Join**

In the AND-Join workflow, multiple parallel threads of sensor clusters converge into one single thread. The AND-Join operation waits for all paths to complete execution before joining them into a single thread.

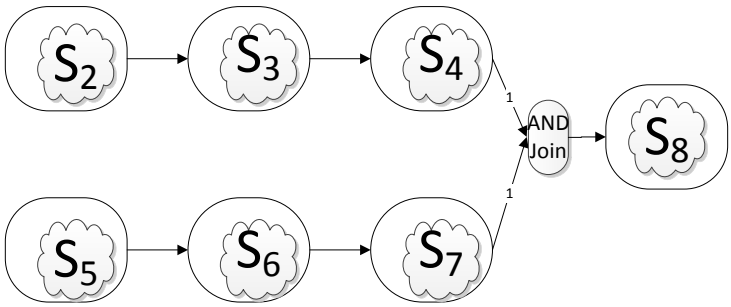


Figure 13: AND-Join of Sensor Clusters

3.3.2.3. **XOR-Split**

The XOR-Split workflow operation will select one and only one path if the condition is true as illustrated in Figure 14: XOR-Split of Sensor Clusters. It will make an exclusive choice of one path out of several paths unless all inputs are false.

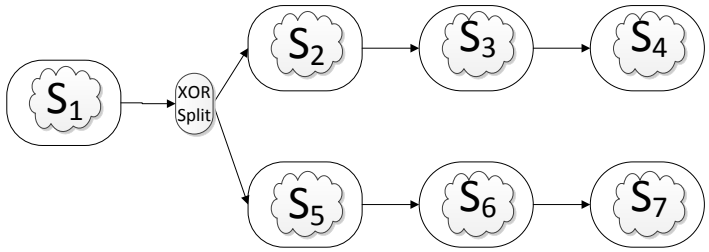


Figure 14: XOR-Split of Sensor Clusters

3.3.2.4. **XOR-Join**

The XOR-Join consists of a non-synchronizing merge of paths. It requires that only one path complete to satisfy the join condition as shown in Figure 15: XOR-Join of Sensor Clusters.

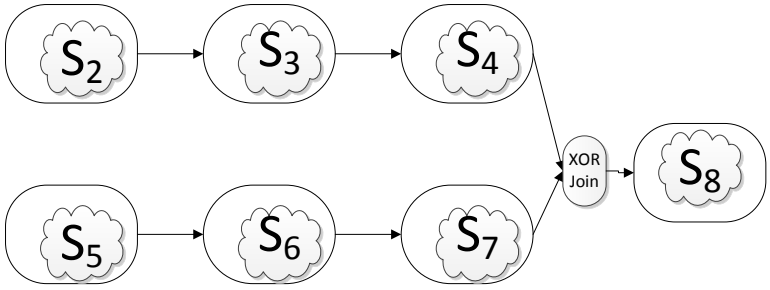


Figure 15: XOR-Join of Sensor Clusters

3.3.3. Complex Workflow Operations

There are more advanced workflow operations that can be used in a workflow process. There are several complex workflow operations like the Multi-Choice, Synchronizing Merge, Multi-Merge and Discriminator that are used in workflow processes. However, each of these can be implemented using the basic workflow operations of AND-Split, AND-Join, XOR-Split and XOR-Join. Since these workflow operations can be derived from simpler conditional operations we call them complex.

3.3.3.1. Multi-Choice (OR-Split)

The Multi-choice workflow operation is based on a decision or workflow control data that selects a number of branches for execution. It performs the same function as an OR-Split conditional operator. That is one or more branches can be executed if the condition is true. The OR-Split can be implemented using the XOR-Split, AND-Split, AND-Join and the XOR-Join [34] as illustrated in Figure 16: OR-Split of Sensor Clusters.

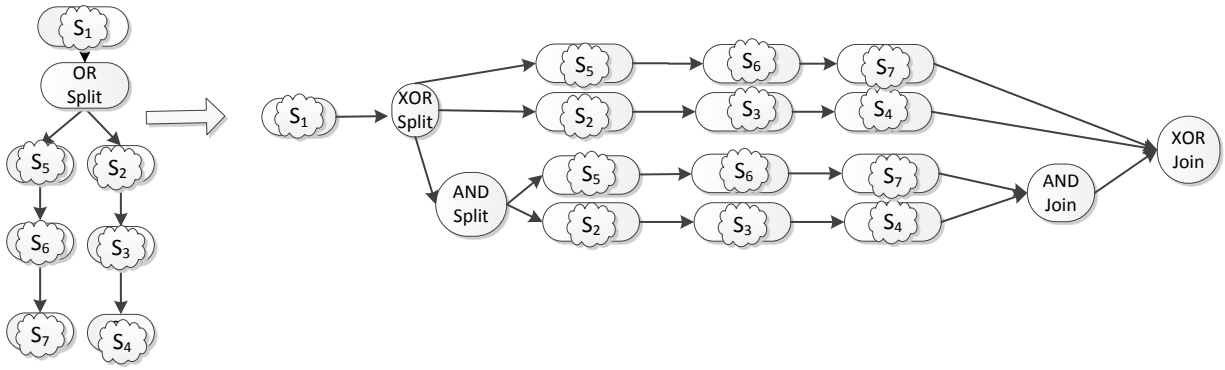


Figure 16: OR-Split of Sensor Clusters

While the two examples in Figure 16: OR-Split of Sensor Clusters are not equivalent, they perform the same function.

3.3.4. Multi-Merge

The Multi-Merge operation is where two or more branches re-converge without waiting for all branches to complete. The activity following the merge is started for every activated incoming branch. This can be implemented as an AND-Split followed by the activity following the merge for each path as seen in Figure 17: Multi Merge Operation [34].

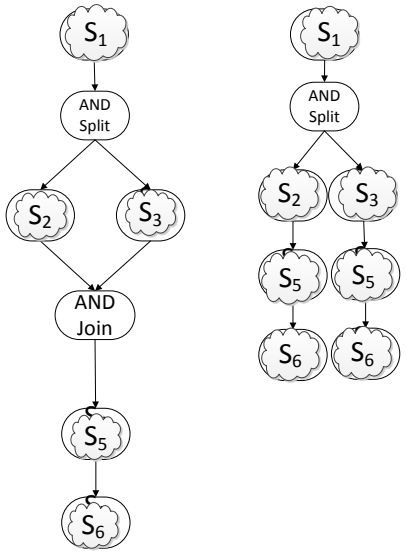


Figure 17: Multi Merge Operation

3.3.5. Discriminator

The Discriminator waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on, all remaining branches to complete are ignored. Once all incoming branches have been triggered, it resets itself so that it can be triggered again. This can be implemented using AND-Join and XOR-Join as shown in Figure 18: Discriminator Operation. [34]

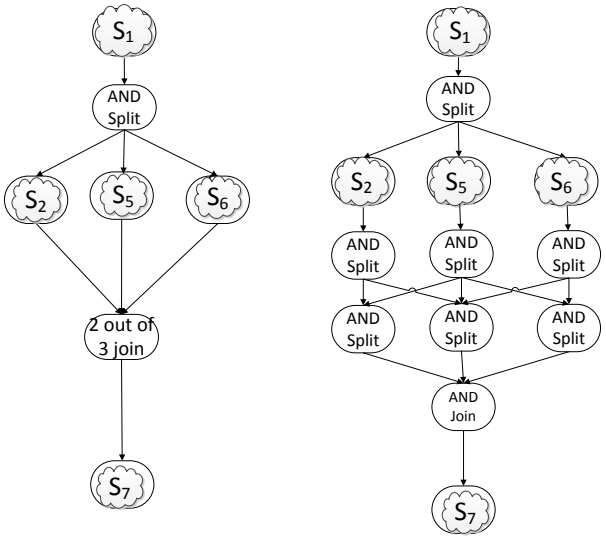


Figure 18: Discriminator Operation

Since the function of the complex conditional operations can be implemented by the AND-Split and XOR-Split conditional operators, our proofs will focus on these operators only even though our diagrams may include any of the complex conditional operators.

3.4. Calculating QoS for a Composite Sensor Cluster using A QoS Aggregation Model

When executing a CSI it is necessary to be able to specify constraints that the CSI is required to adhere to. These constraints are typically expressed in the form of a service level agreement. The constraints indicate the level of Quality of Service (QoS) that must be met when the CSI is executed. In our model the QoS that we are concerned with are response time, reliability, cost and energy required. A method for calculating these QoS metrics in a CSI is necessary. We can then specify the QoS constraints that the CSI must meet.

3.4.1. Calculating QoS

Given a CSI, we must calculate the execution time, reliability, cost and energy required for the workflow. In our simplest case, a sequential workflow, the calculation is straightforward.

Figure 19: CSI(G) - Sequential Sensor Clusters

Each sensor cluster has an execution time, reliability, cost and energy required metric that is stored in the QoS table. In Figure 19: CSI(G) - Sequential Sensor Clusters, we show that for function F1, sensor cluster S2 will perform that service, for function F2, sensor cluster S3 will perform that service and for function F3 sensor cluster S4 will perform that service. The following function-sensor cluster notation can be used to represent the relationship: either the letter F followed by the function number and S followed by the sensor cluster number or S followed by the function number and sensor cluster number. For example, for the first function-sensor cluster pair in Figure 19, F_1S_2 and S_{12} can be used to represent that relationship.

To calculate the total execution time, reliability, cost and energy required, the following formulas are used:

1. $F_{Reliability}(CSI(G)) = F_{Reliability}(S_{12}) * F_{Reliability}(S_{23}) * F_{Reliability}(S_{34})$ EQ [1]

   Total Reliability of the CSI(G) is calculated as the product of the reliability of the individual sensor clusters in a sequential workflow

2. $F_{ExecutionTime}(CSI(G)) = F_{ExecutionTime}(S_{12}) + F_{ExecutionTime}(S_{23}) + F_{ExecutionTime}(S_{34})$ EQ [2]

   Total Execution time of the CSI(G) is calculated as the sum of the execution time of the individual sensor clusters in a sequential flow

3. $F_{Cost}(CSI(G)) = F_{Cost}(S_{12}) + F_{Cost}(S_{23}) + F_{Cost}(S_{34})$ EQ [3]

Total cost of the CSI(G) is calculated as the sum of the cost of the individual sensor clusters in a sequential workflow

4. $F_{Energy}(CSI(G)) = F_{Energy}(S_{12}) + F_{Energy}(S_{23}) + F_{Energy}(S_{34})$ EQ [4]

Total energy required for the CSI(G) is calculated as the sum of the energy required of the individual sensor clusters in a sequential workflow. There must be enough energy available to satisfy the energy required for the CSI.

3.4.2. Incremental Graph Transformation

The QoS formulas for sequential workflow are straight forward. However, to calculate QoS for conditional workflow, a methodology for dealing with conditions and parallel computation is required. To calculate QoS for conditional workflow, we propose using a QoS aggregation algorithm [34] on the CSI.

The QoS aggregation performs an incremental collapse of the graph into a single node by alternately aggregating simple sequences and parallel service executions. We look at local composition patterns which are differentiated into sequential and parallel service executions. For sequential workflow, we are able to calculate the QoS for cost, energy required, reliability and execution time using the given formulas EQ[1] through EQ[4].

To achieve this level of simplicity in a parallel workflow, we can treat each workflow thread in a parallel workflow as a sequential workflow. We find the QoS totals for each of the sequential workflows and collapse that section of the graph. We continue this pattern until all parallel workflows are collapsed in the CSI. In Figure 20: Incremental QoS Aggregation - Step 1 we

perform the QoS calculations for each sequential workflow between the AND-Split and AND-Join operators and we obtain a result for each sequential workflow.

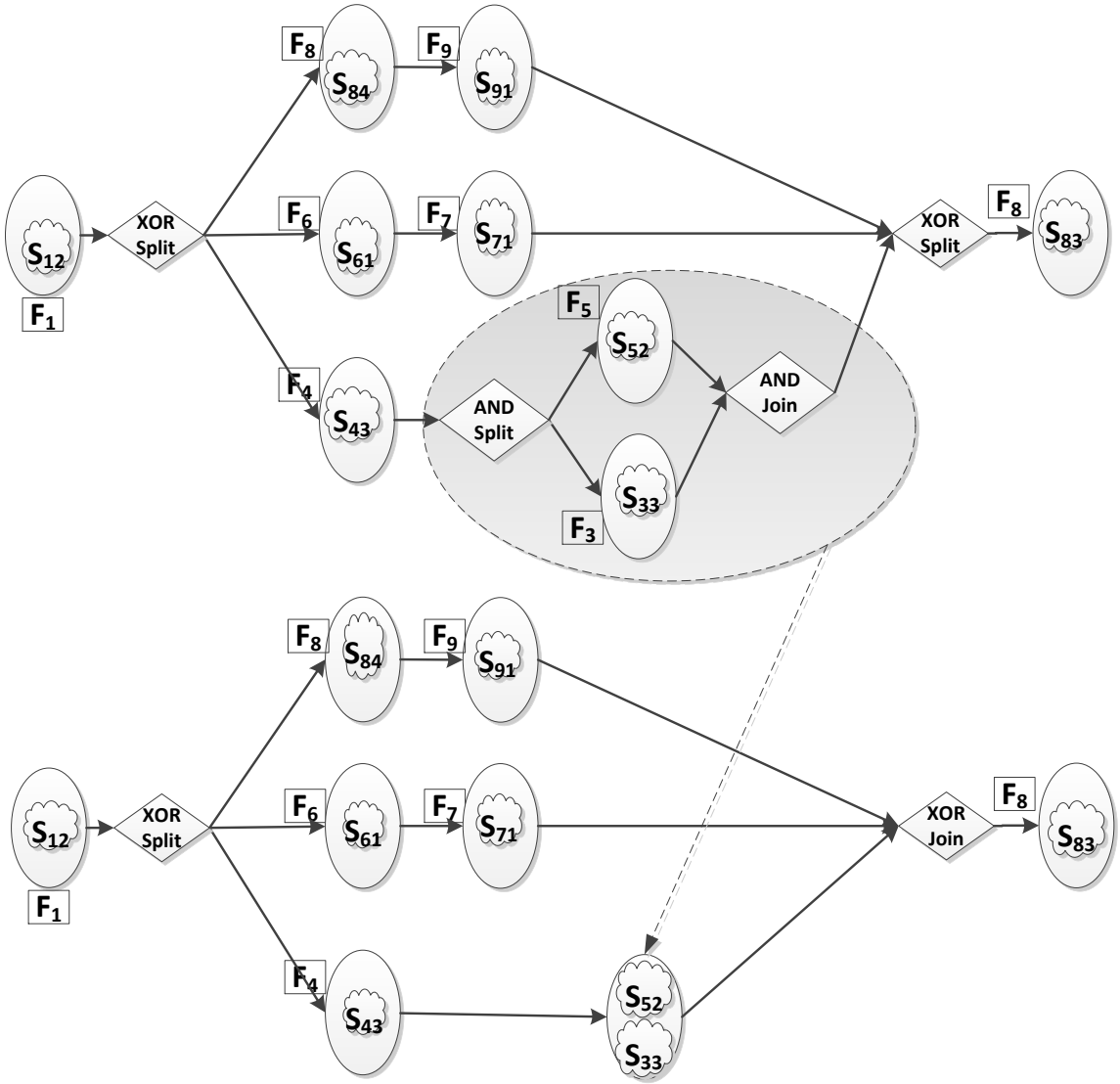


Figure 20: Incremental QoS Aggregation - Step 1

In our next step, we are able to compute the QoS for all three sequential workflows with the XOR-Split and XOR-Join operators as illustrated in Figure 21: Incremental QoS Aggregation - Step 2 . We return the QoS totals for each of the sequential paths.

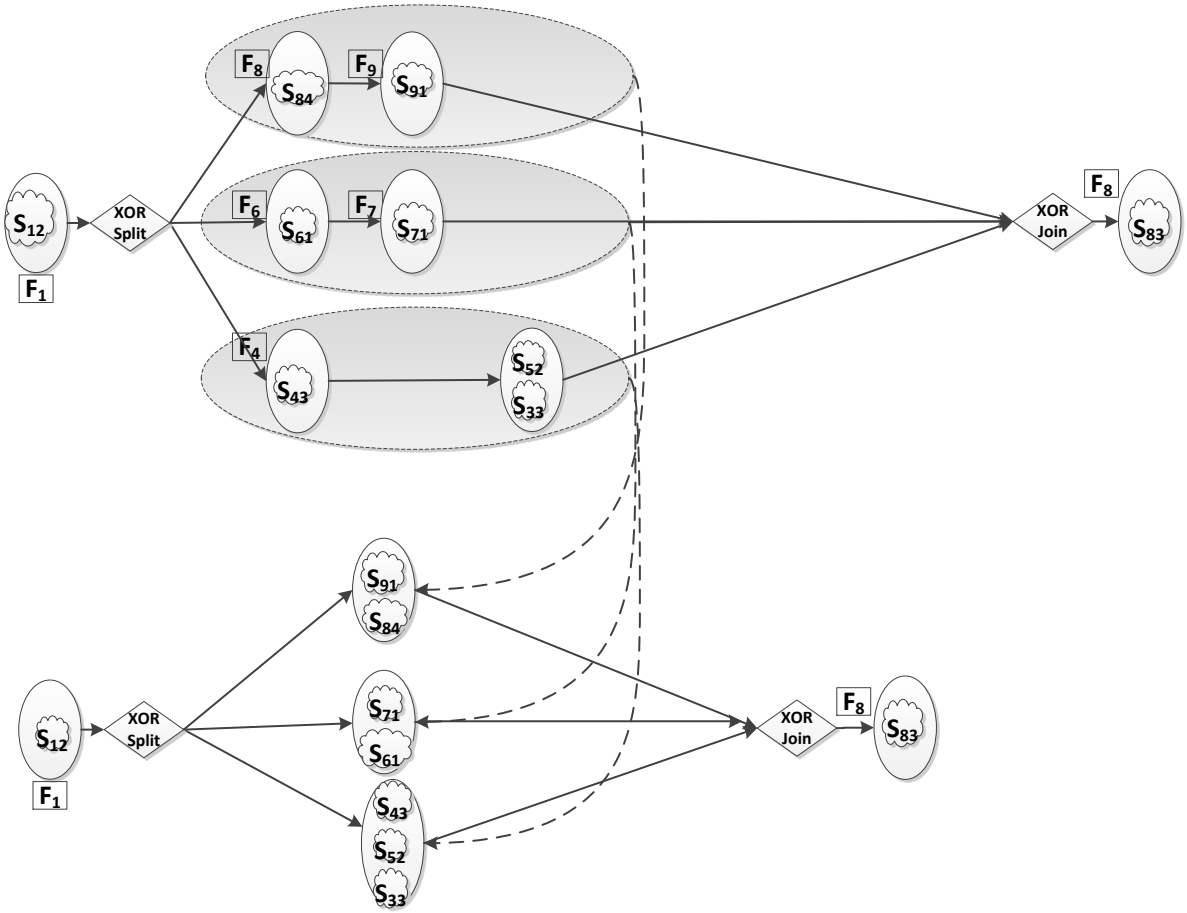


Figure 21: Incremental QoS Aggregation - Step 2

In our third step, we are able to apply the XOR operation on our previous results to obtain a new result as seen in Figure 22: Incremental QoS Aggregation - Step 3

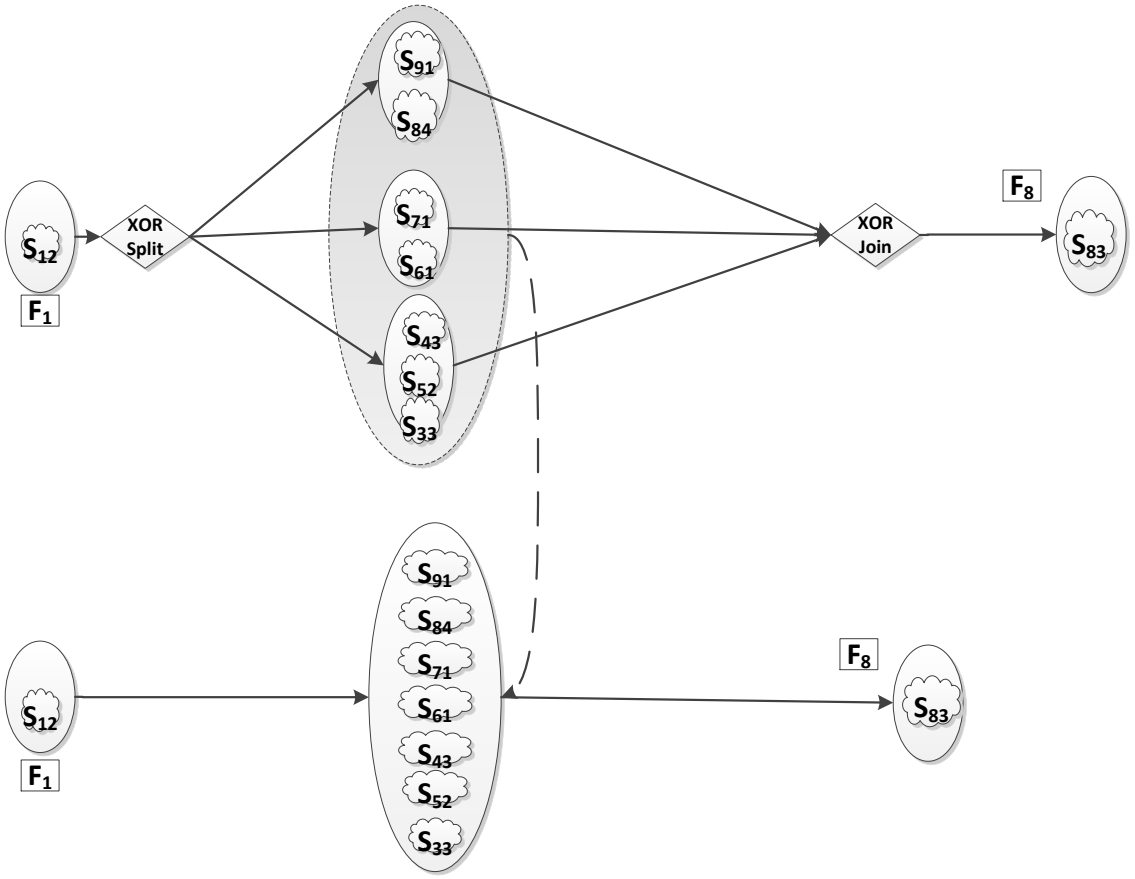


Figure 22: Incremental QoS Aggregation - Step 3

Finally, we are left with one sequential workflow, which we are able to solve using our QoS formulas for cost, execution time, energy required and reliability. This results in the final QoS aggregation as illustrated in Figure 23: Incremental QoS Aggregation - Final Step, leaving us with the end result.

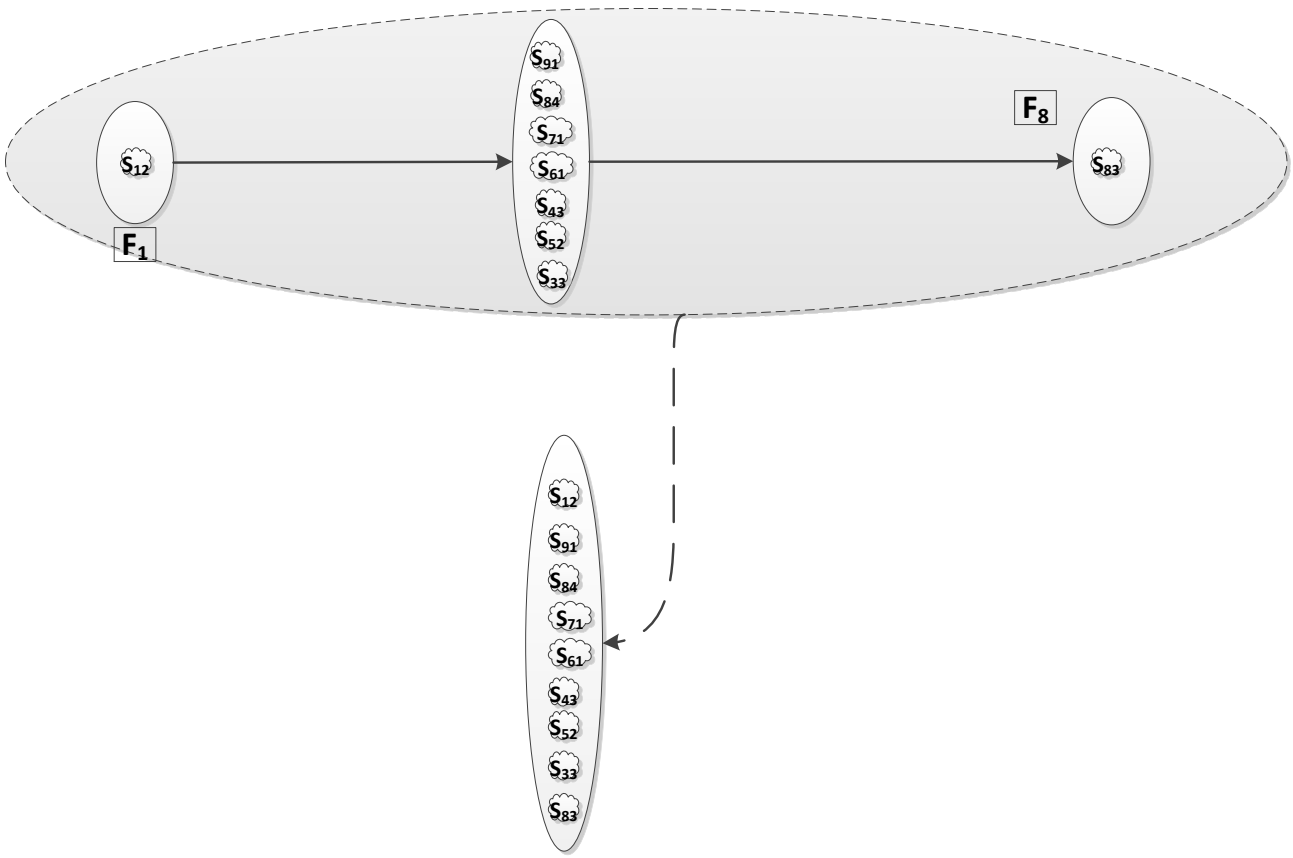


Figure 23: Incremental QoS Aggregation - Final Step

3.4.3. Execution Time QoS Aggregation

We can now apply the QoS aggregation method to calculate QoS in a CSI. However, we must take into account how to apply the conditional operations of the AND-Split and XOR-Split to our formula. We will show how to calculate QoS for execution time, cost, reliability and energy required with conditional operations within the workflow.

3.4.3.1. **Sequential Composition and Loops**

Let S_{ij} represent a sensor cluster j, where $i = 1, \dots, n$ and n is the number of functions in the $CSI(G)$; $j = 1, \dots, q_i$, where q_i is the set of sensors clusters for a function i in the $CSI(G)$. The function $F_{ExecutionTime}(S_{ij})$ will retrieve the execution time for sensor cluster S_{ij}. The formula to find the total execution time for $CSI(G)$ is:

$$TotalExecTime = \sum_{i=1}^{n} F_{ExecutionTime}(S_{ij}) \quad j \in \{1, \dots, q_i\} \qquad \text{EQ [5]}$$

To calculate the k iterations of the sequential workflow, the formula is

$$TotalExecTime = k \sum_{i=1}^{n} F_{ExecutionTime}(S_{ij}) \quad j \in \{1, \dots, q_i\} \qquad \text{EQ [6]}$$

3.4.3.2. **Parallel Composition Patterns**

From a QoS perspective in respect to execution time, the smaller the value the better. Hence, the largest value of an execution time for a sequential workflow in a parallel arrangement denotes the worst case. A conservative approach in estimating the total execution time for a CSI would be to use the worst case scenario in calculating the execution time. As a result, we find the path in a parallel workflow with the maximum execution time to factor into our total execution time for the CSI.

A parallel workflow consists of two or more sequential paths. Let $\{P_1, P_2, \dots, P_m\}$ be the set of sequential paths in an XOR-Split in $CSI(G)$ where $x = 1, \dots, m$, and m represents the total number

of sequential paths in the parallel workflow, n_x represents the number of functions in that sequential path, then

$$P_x = \sum_{i=1}^{n_x} F_{ExecutionTime}(S_{ij}) \quad j \in \{1, \dots, q_i\} \qquad \text{EQ [7]}$$

For a sequential path A we can use the formula in EQ[7]. A parallel workflow consists of two or more sequential paths. Let $\{A_1, A_2, \dots, A_g\}$ be the set of sequential paths in an AND-Split in $CSI(G)$ where $d = 1, \dots, g$, and g represents the total number of sequential paths in the parallel workflow, n_d represents the number of functions in that sequential path, then

$$A_d = \sum_{v=1}^{n_d} F_{ExecutionTime}(S_{vu}) \; u \in \{1, \dots, q_v\} \qquad \text{EQ [8]}$$

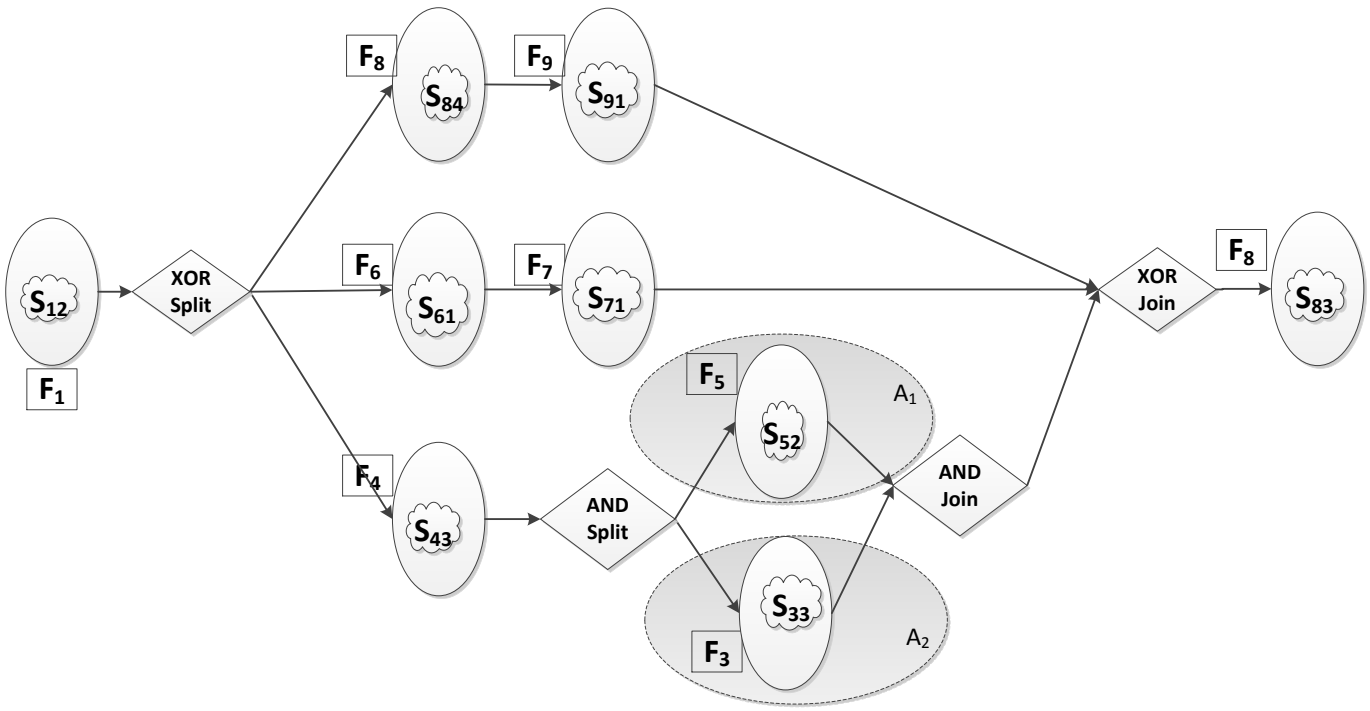


Figure 24: Calculating Execution Time for Parallel Workflow - Step 1

Specifically for this example the formula is:

$$A_1 = \sum_{v=5,u=2} F_{ExecutionTime}(S_{vu}) \qquad \text{EQ [9]}$$

$$A_2 = \sum_{v=3,u=3} F_{ExecutionTime}(S_{vu}) \qquad \text{EQ [10]}$$

Since we are searching for the path with the maximum execution time in the parallel workflow as shown in Figure 25: Calculating Execution Time for Parallel Workflow, the formula is

$$MaxExecutionTime = max(A_1, A_2)$$ EQ [11]

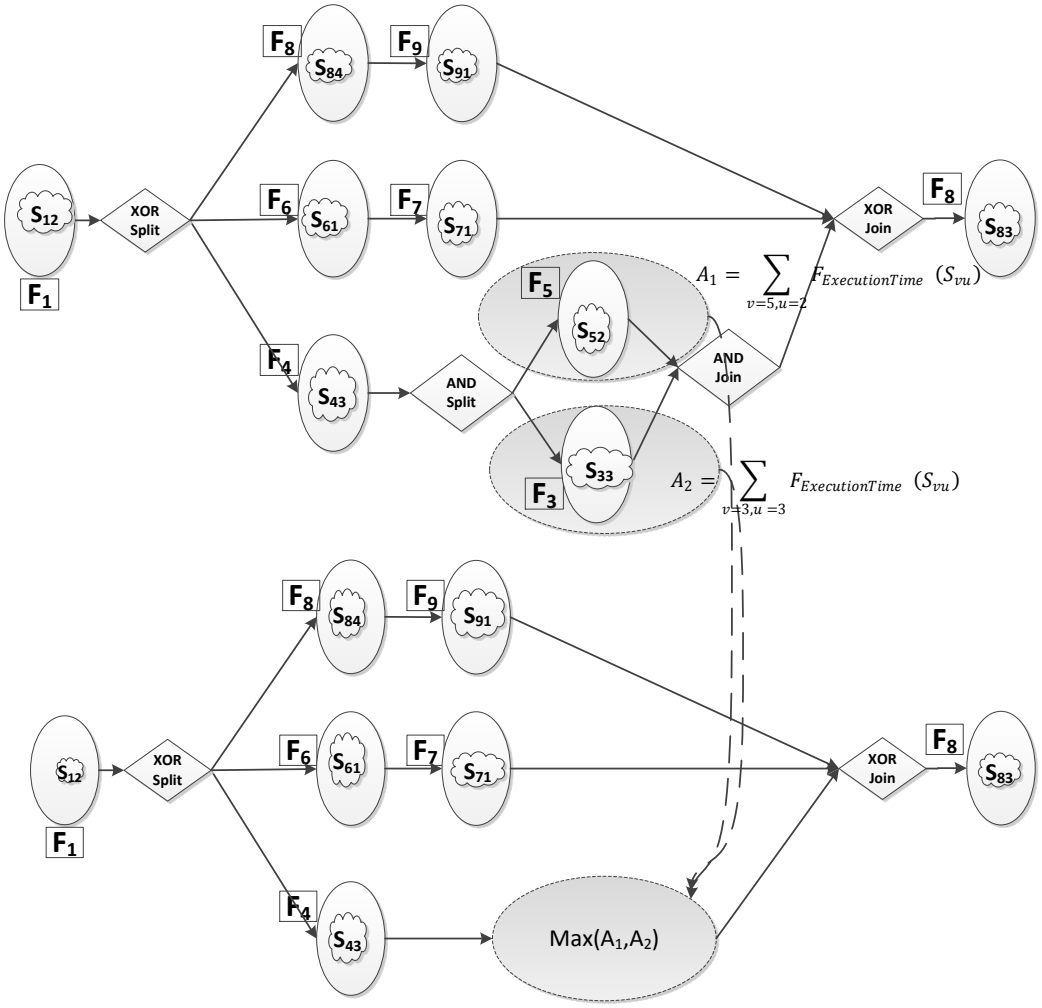


Figure 25: Calculating Execution Time for Parallel Workflow - Step 2

In the next step we will calculate the formulas for the remaining sensor clusters.

$$P_3 = \sum_{i=4,j=3} F_{ExecutionTime}(S_{ij})$$ EQ [12]

$$P_4 = \sum_{i=6, j=1}^{7,1} F_{ExecutionTime}(S_{ij})$$ EQ [13]

$$P_5 = \sum_{i=8, j \in \{4,1\}}^{9,\{4,1\}} F_{ExecutionTime}(S_{ij})$$ EQ [14]

These formulas are used in the three sequential paths of the parallel workflow within the XOR-Split operation as shown in Figure 26: Calculating Execution Time for Parallel Workflow - Step 3.

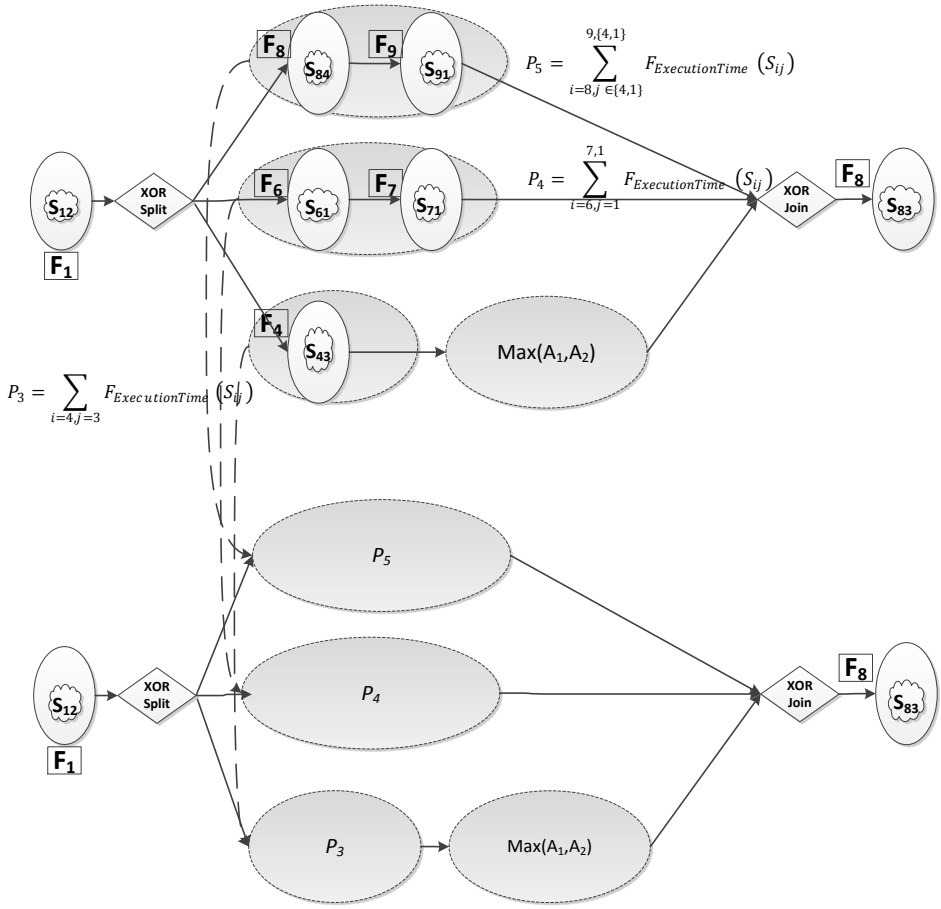


Figure 26: Calculating Execution Time for Parallel Workflow - Step 3

We now have three sequential paths that we can compute the largest execution time for XOR-Split as illustrated in Figure 27:

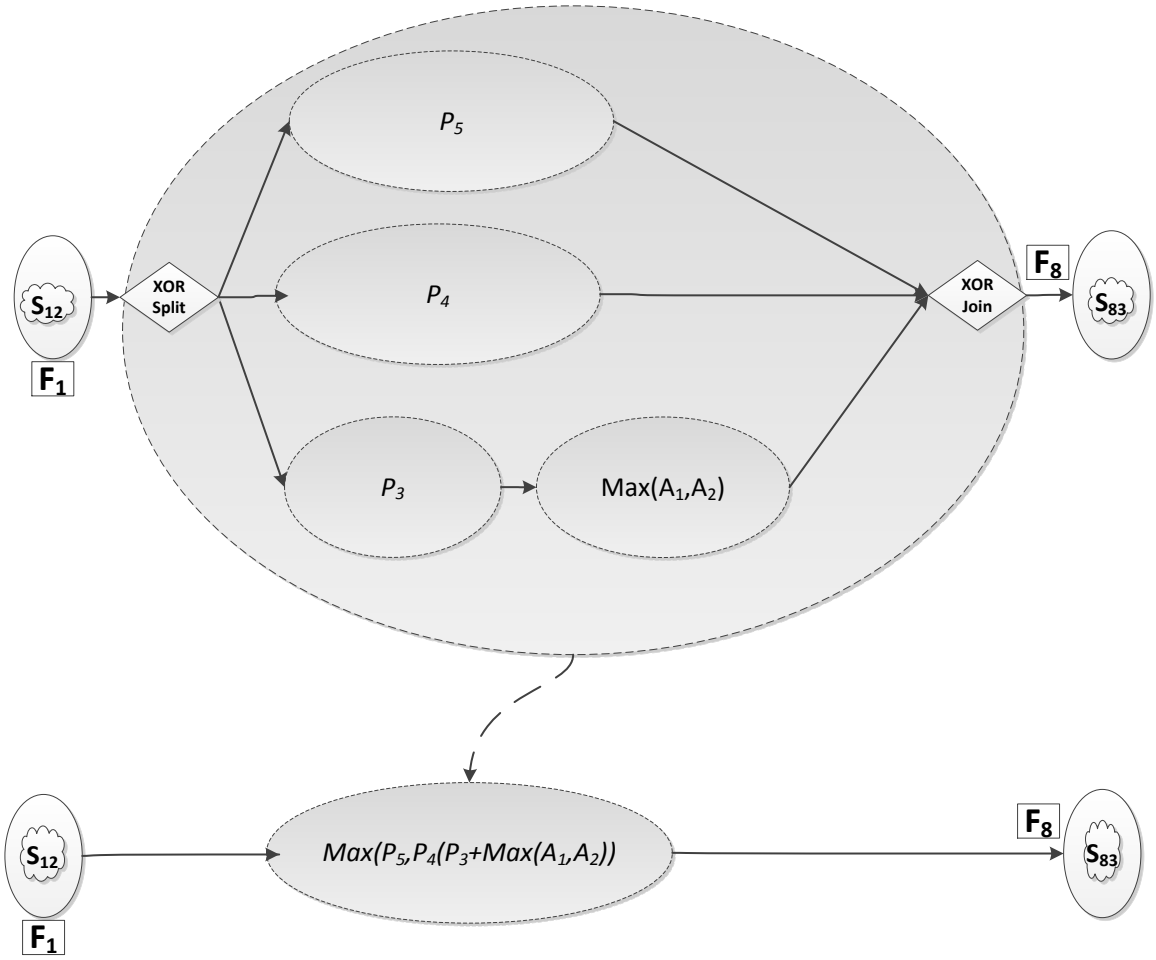


Figure 27: Calculating Execution Time for Parallel Workflow – Final Step

In our final step we can now solve the equation to obtain the $TotalExecutionTime$.

$$TotalExecutionTime = F_{ExecutionTime}(S_{12}) + \max\left(P_5, P_4, \left(P_3 + max(A_1, A_2)\right)\right) + F_{ExecutionTime}(S_{83}) \quad \text{EQ [15]}$$

We have already defined the formula for a sequential path P in EQ[5]. We have also provided the formula for the set of sequential paths in a parallel operation in EQ[7].

To summarize, from a QoS perspective in respect to time, the smaller the value the better. Hence, the largest value in a parallel arrangement of Sensor Clusters denotes the worst case or its upper bound. To determine the minimum execution time in the XOR operation, we find the path with the minimum value as our best case or lower bound.

So we have the following: P is the execution time value of the sensor clusters in a sequential path in an XOR workflow, A represents the execution time value of the sensor clusters in a sequential path that are in an AND workflow. We use variable T going forward for clarity to represent the execution time value of the sensor clusters in a sequential path that is not a part of any parallel workflow.

Let $\{T_1, T_2, \dots, T_h\}$ represent the execution time value of the set of sequential paths that are not in any parallel workflow in the CFG where $b = 1,2, \dots, h$.

Let $\{P_1, P_2, \dots, P_m\}$ represent the execution time value of the set of sequential paths in an XOR-Split where m is the number of sequential paths for that operation.

Let z = the number of XOR-Splits of the workflow in the CFG where $m = 1,2, \dots, z$. So for each r there will be a set of parallel paths in which we find the path with the maximum execution time value, that is we seek to obtain $\max\{P_1, P_2, \dots, P_m\}_r$ for each r.

Let $\{A_1, A_2, \dots, A_g\}$ represent the execution time value of the set of sequential paths in an AND-Split where g is the number of sequential paths for that operation.

Let y = the number of AND-Splits of the workflow in the CFG where $f = 1, \dots, y$. So for each f there will be a set of parallel paths where we obtain the maximum of the execution time of all paths, that is we seek to obtain $\max\{A_1, A_2, \dots, A_g\}_f$.

Then our formula is:

$$TotalExecutionTime\big(CSI(G)\big) = \sum_{b=1}^{h} T_b + \sum_{f=1,o=1}^{y,g_y} max(A_1, A_2, \dots, A_o)_f + \sum_{r=1,x=1}^{z,m_z} \max(P_1, P_2, \dots, P_x)_r \quad \text{EQ [16]}$$

3.4.4. Reliability QoS Aggregation

Reliability denotes the probability that the execution of the sensor cluster performs successfully. From a QoS perspective, the higher the reliability, the better the service from the sensor cluster.

3.4.4.1. Sequential Composition

Let S_{ij} represent a sensor cluster j, where $i = 1, \dots, n$ and n is the number of functions in the $CSI(G)$; $j = 1, \dots, q_i$, where q_i is the set of sensors clusters for a function i in the $CSI(G)$. The

function $F_{Reliability}(S_{ij})$ will retrieve the reliability for sensor cluster S_{ij}. The formula to find the total reliability for $CSI(G)$ is:

$$TotalReliability = \prod_{i=1}^{n} F_{Reliability}(S_{ij}) \quad j \in \{1, \dots, q_i\} \qquad \text{EQ [17]}$$

However for a loop of the sequential workflow with k iterations, the formula is

$$TotalReliability = \left(\prod_{i=1}^{n} F_{Reliability}(S_{ij})\right)^{k} \quad j \in \{1, \dots, q_i\} \qquad \text{EQ [18]}$$

3.4.4.2. **Parallel Composition Patterns**

From a QoS perspective in respect to reliability, the greater the value the better. Hence, the smallest value of reliability for a sequential path in a parallel arrangement denotes the worst case. A conservative approach in estimating the total reliability for a CSI would be to use the worst case scenario in calculating the reliability. As a result, we find the path in a parallel workflow with minimum reliability to factor into our total reliability for the CSI.

Let $\{P_1, P_2, \dots, P_m\}$ be the set of sequential paths in an XOR-Split in $CSI(G)$ where $x = 1, \dots, m$, and m represents the total number of sequential paths in the parallel workflow, then

$$P_x = \prod_{i=1}^{n_x} F_{Reliability}(S_{ij}) \quad j \in \{1, \dots, q_i\} \qquad \text{EQ [19]}$$

Let $\{A_1, A_2, \dots, A_g\}$ be the set of sequential paths in an AND-Split in $CSI(G)$ where $d = 1, \dots, g$, and g represents the total number of sequential paths in the parallel workflow, then

$$A_d = \prod_{v=1}^{n_d} F_{Reliability}(S_{vu}) \qquad u \in \{1, \dots, q_v\} \qquad \text{EQ [20]}$$

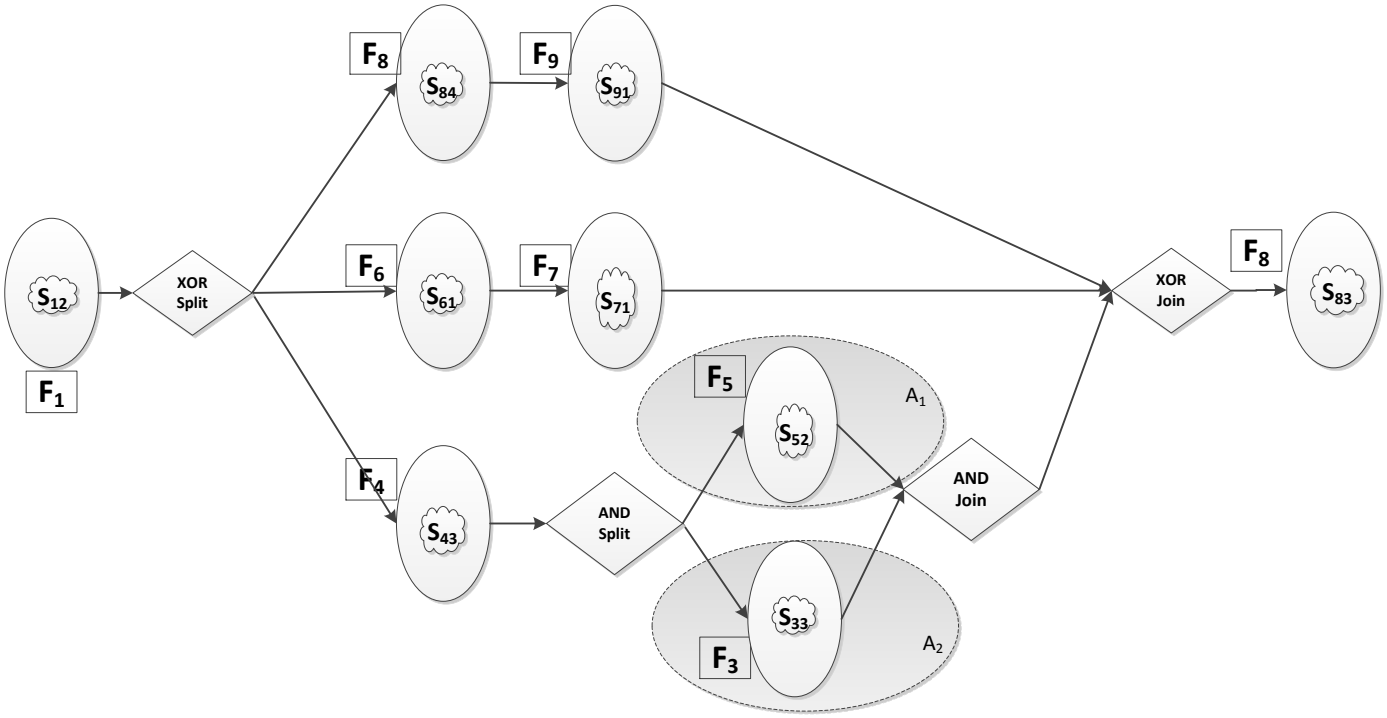


Figure 28: Calculating Reliability for Parallel Workflow - Step 1

Specifically for this example the formula is:

$$A_1 = \prod_{v=5,u=2} F_{Reliability}(S_{\mathrm{vu}})$$ EQ [21]

$$A_2 = \prod_{v=3,u=3} F_{Reliability}(S_{\mathrm{vu}})$$ EQ [22]

In the case of reliability, for the AND-Split operation we do not take the max or min since all of the sequential workflows will be executed in the event that the AND-Split operation is true. We must factor all of the sequential workflows in the AND-Split, hence we take the product.

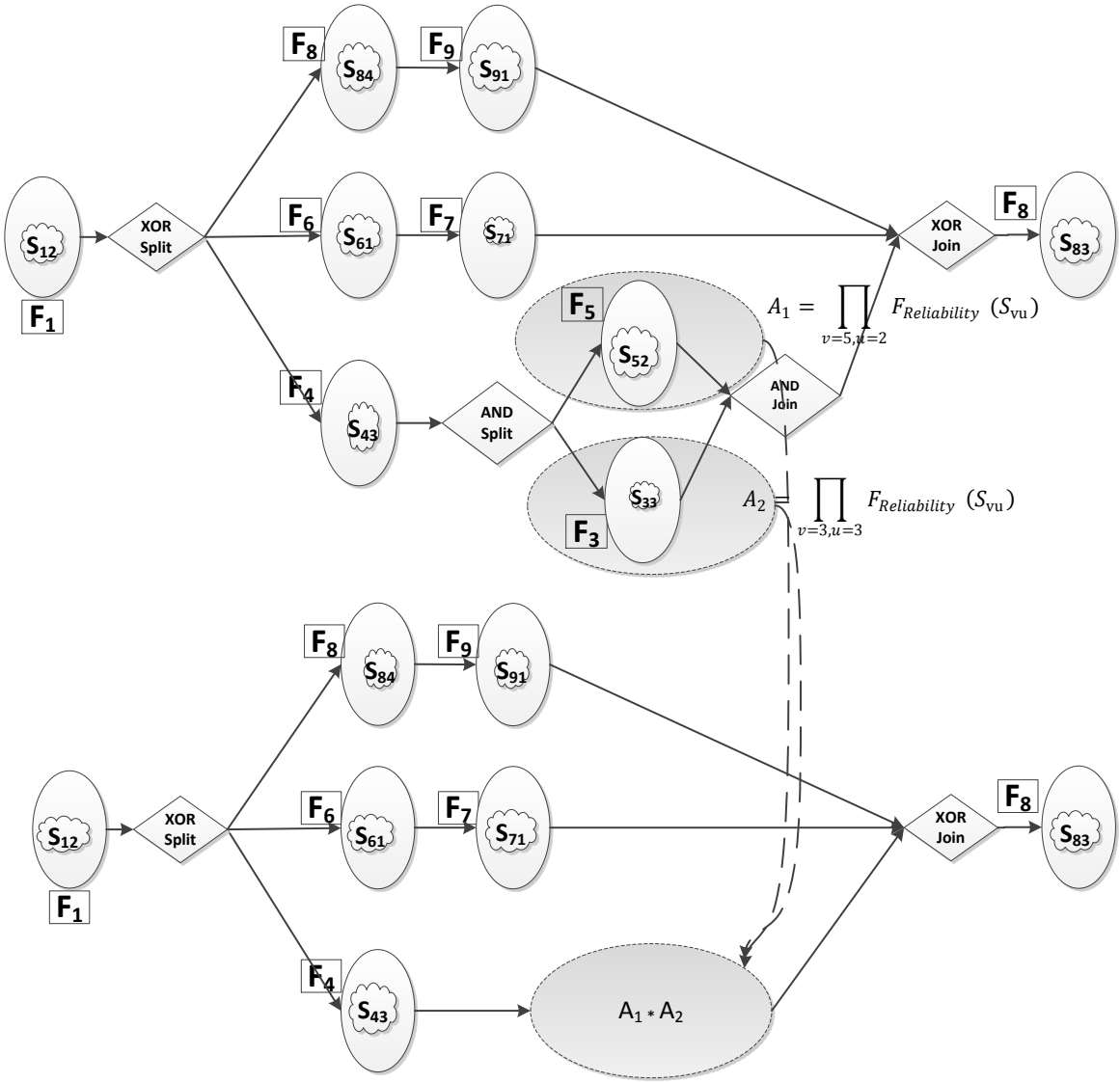


Figure 29: Calculating Reliability for Parallel Workflow - Step 2

For the next step in the aggregation we obtain the following formulas:

EQ [23]

$$P_3 = \prod_{i=4,j=3} F_{Reliability}(S_{ij})$$

$$P_4 = \prod_{i=6,j=1}^{7,1} F_{Reliability}(S_{ij})$$ EQ [24]

$$P_5 = \prod_{i=8,j\in\{4,1\}}^{9,\{4,1\}} F_{Reliability}(S_{ij})$$ EQ [25]

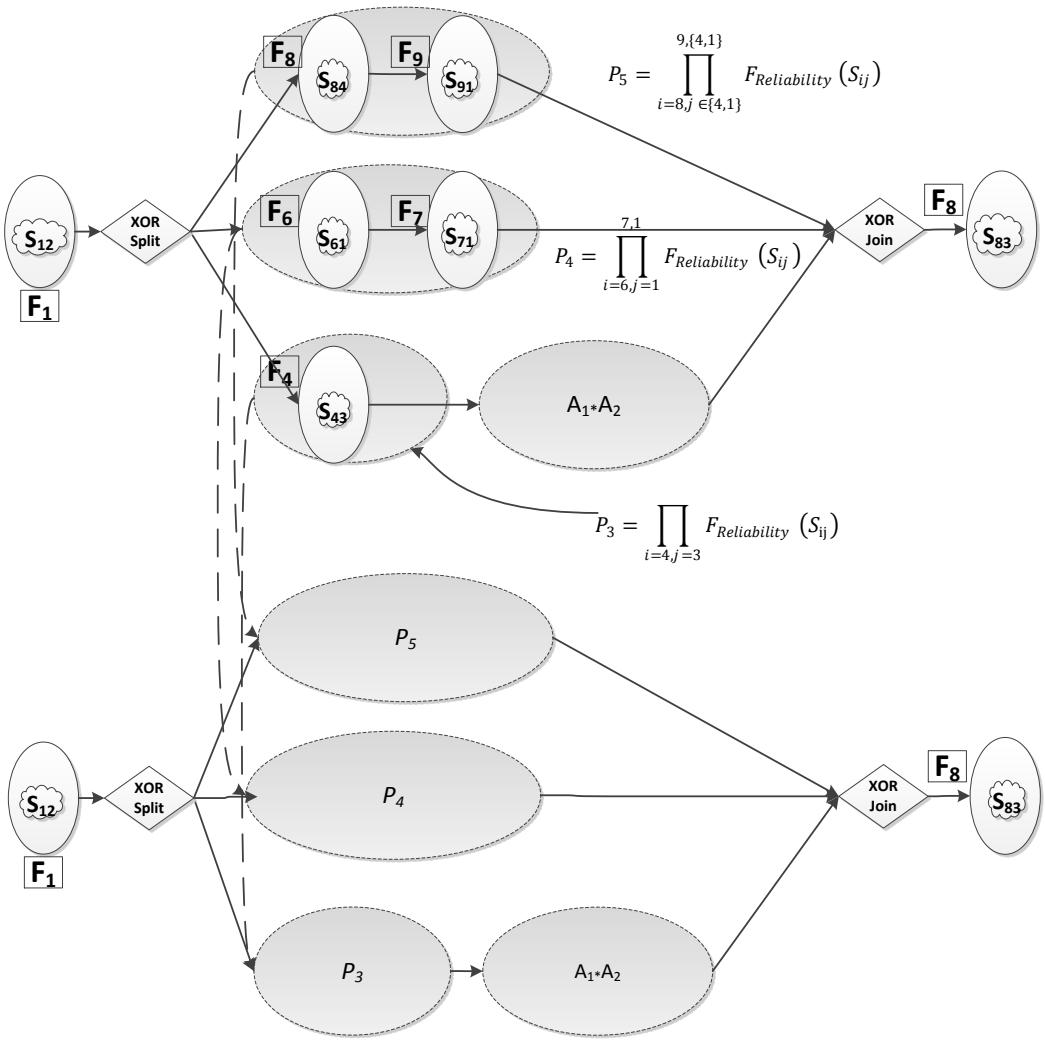


Figure 30: Calculating Reliability for Parallel Workflow - Step 3

In the case of the XOR-Split, only one path will be chosen. As a result, we can take the minimum value of the sequential paths as our worst case to factor into the total reliability for $CSI(G)$.

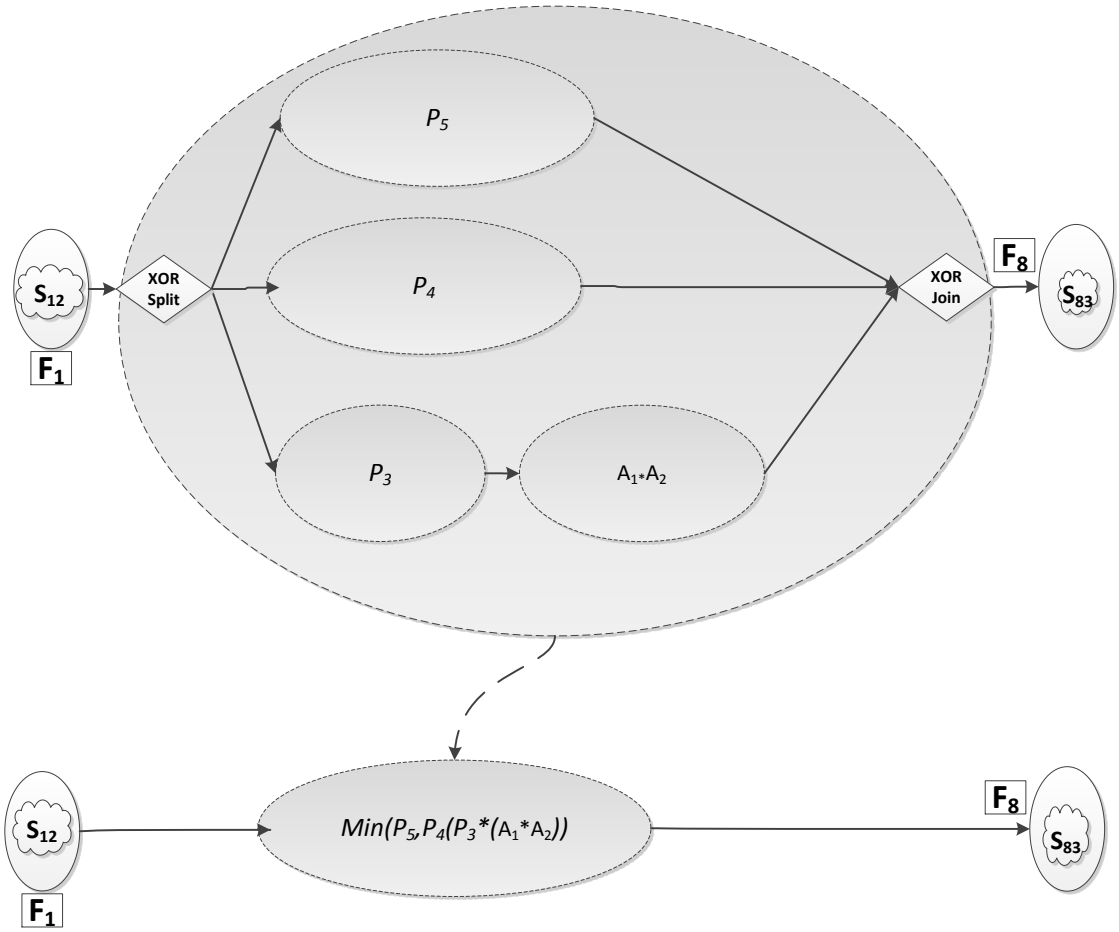


Figure 31: Calculating Reliability for Parallel Workflow - Final Step

For our final step, we obtain the formula:

$$TotalReliability = F_{Reliability}(S_{12}) * min\left(P_5, P_4, \left(P_3 * (A_1 * A_2)\right)\right) * F_{Reliability}(S_{83}) \quad \text{EQ [26]}$$

To summarize, P is the reliability value of the sensor clusters in a sequential path in an XOR workflow, A represents the total reliability value of the sensor clusters in a sequential path that are in an AND workflow and T represents the total reliability value of the sensor clusters in a sequential path that is not a part of any parallel workflow.

Let $\{T_1, T_2, \dots, T_h\}$ represent the total reliability value of the set of sequential paths that are not in any parallel workflow in the CFG.

Let $\{P_1, P_2, \dots, P_m\}$ represent the total reliability value of the set of sequential paths in an XOR-Split where m is the number of sequential paths for that operation.

Let z = the number of XOR-Splits of the workflow in the CFG where $r = 1,2, \dots, z$. So for each r there will be a set of parallel paths in which we find the path with the minimum reliability value, that is we seek to obtain $\min\{P_1, P_2, \dots, P_m\}_r$ for each r.

Let $\{A_1, A_2, \dots, A_g\}$ represent the total reliability value of the set of sequential paths in an AND-Split where g is the number of sequential paths for that operation.

Let y = the number of AND-Splits of the workflow in the CFG where $f = 1, \dots, y$. So for each f there will be a set of parallel paths where we obtain the product of the reliability of all paths, that is we seek to obtain $\{A_1 * A_2 * \dots * A_g\}_f$.

Then our formula is:

$$TotalReliability(CSI(G)) = \prod_{b=1}^{h} T_b + \qquad \text{EQ [27]}$$

$$\prod_{f=1,o=1}^{y,g_y} (A_1 * A_2 * \dots * A_o)_f \; + \prod_{r=1,x=1}^{z,m_z} \min(P_1, P_2, \dots, P_x)_r$$

3.4.5. Cost QoS Aggregation

The cost of a sensor cluster is the amount of money required to execute the service. The objective is to spend as little as possible on a service that meets the needs of the client. In that respect, the lower the cost the more desirable the services as long as all other qualities are comparable.

3.4.5.1. Sequential Composition

Let S_{ij} represent a sensor cluster j, where $i = 1, \dots, n$ and n is the number of functions in the $CSI(G)$; $j = 1, \dots, q_i$, where q_i is the set of sensors clusters for a function i in the $CSI(G)$. The function $F_{Cost}(S_{ij})$ will retrieve the cost for sensor cluster S_{ij}. The formula to find the total cost for $CSI(G)$ is:

$$TotalCost = \sum_{i=1}^{n} F_{Cost}(S_{ij}) \quad j \in \{1, \dots, q_i\}$$ EQ [28]

However for a loop of the sequential workflow, the formula is

$$TotalCost = \mathrm{k}\sum_{i=1}^{n} F_{Cost}(S_{ij}) \quad j \in \{1, \ldots, q_i\}$$ EQ [29]

3.4.5.2. **Parallel Composition Patterns**

In cost aggregation, all services started must be taken into account regardless if they are relevant for the synchronizing join or not.

From a QoS perspective, in respect to cost, the smallest cost value for a sensor cluster is considered to be the best choice. Hence, the smallest value of cost for a sequential workflow in a parallel arrangement denotes the best case. A conservative approach in estimating the total cost for a CSI would be to use the worst case scenario in calculating the cost. As a result, we find the path in an XOR parallel workflow with the maximum cost to factor into our total cost for the CSI.

Cost is similar to Reliability in the sense that this is the case where the XOR-Split is handled differently than the AND-Split. In the XOR-Split, we look for the path with the largest cost value as our worst case. In the AND-Split, we must take into account the cost of all paths of the split. This is necessary because in the event that the AND clause is true, all of the sensors clusters will be executed. (This is the same for reliability and energy required. The only special case is with execution time.)

Let $\{P_1, P_2, \ldots, P_m\}$ be the set of sequential paths in an XOR-Split in $CSI(G)$ where $x = 1, \ldots, m$, and m represents the total number of sequential paths in the parallel workflow, then

$$P_x = \sum_{i=1}^{n_x} F_{Cost}(S_{ij}) \qquad j \in \{1, \dots, q_i\}$$ EQ [30]

A parallel workflow consists of two or more sequential paths. Let $\{A_1, A_2, \dots, A_g\}$ be the set of sequential paths in an AND-Split in $CSI(G)$ where $d = 1, \dots, g$, and g represents the total number of sequential paths in the parallel workflow, then

$$A_d = \sum_{v=1}^{n_d} F_{Cost}(S_{vu}) \quad u \in \{1, \dots, q_v\}$$ EQ [31]

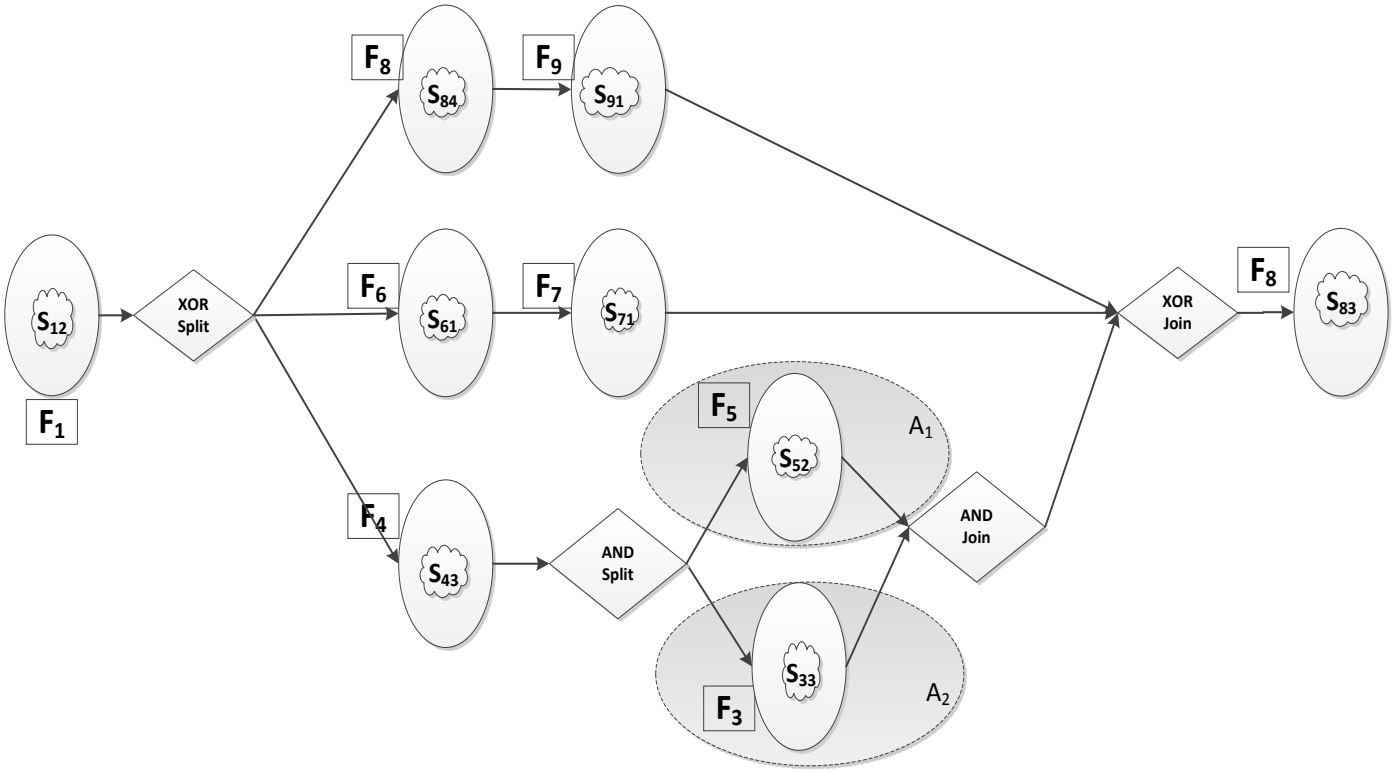


Figure 32: Calculating Cost for Parallel Workflow - Step 1

Specifically for this example the formula is:

$$A_1 = \sum_{v=5,u=2} F_{Cost}(S_{\mathrm{vu}}) \quad \text{EQ [32]}$$

$$A_2 = \sum_{v=3,u=3} F_{Cost}(S_{\mathrm{vu}}) \quad \text{EQ [33]}$$

In the case of Cost, for the AND-Split operation we do not take the max or min since all of the sequential workflows will be executed in the event that the AND-Split operation is true. We must add all of the sequential workflows in the AND-Split to get the total cost.

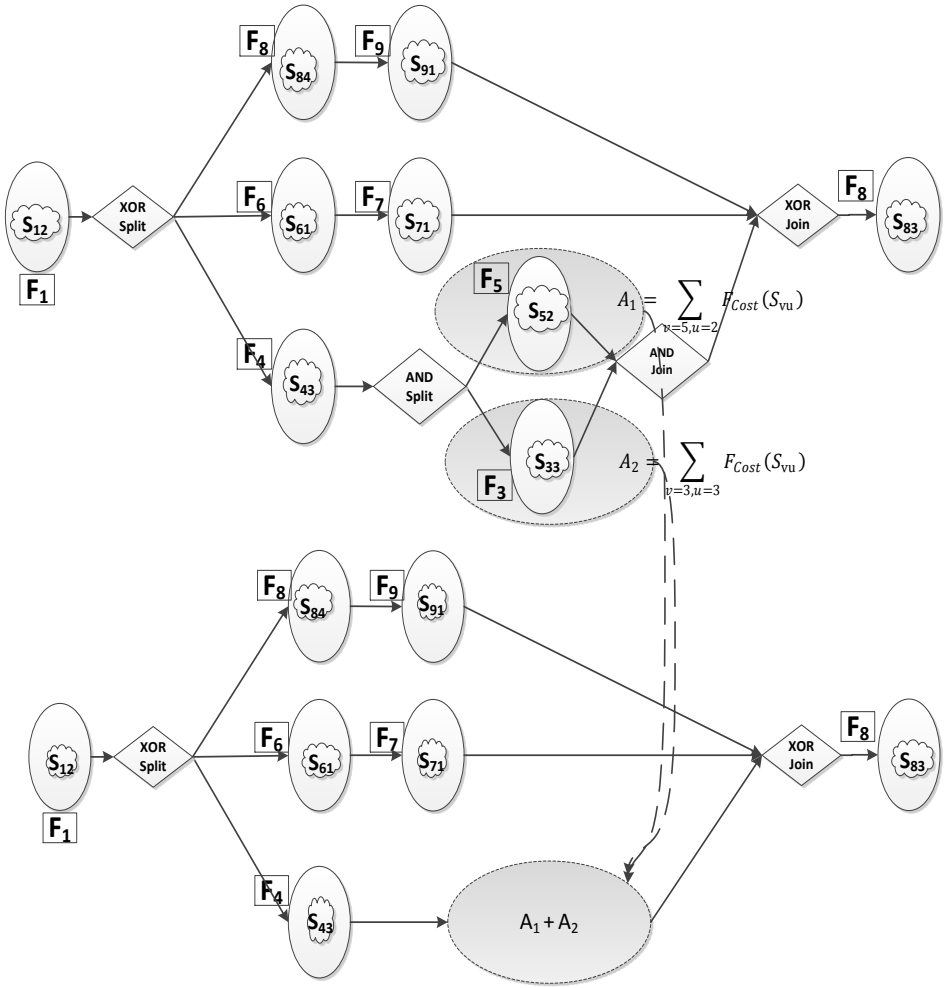


Figure 33: Calculating Cost for Parallel Workflow - Step 2

For the next step in the aggregation we obtain the following formulas:

$$P_3 = \sum_{i=4,j=3} F_{Cost}(S_{ij}) \quad \text{EQ [34]}$$

$$P_4 = \sum_{i=6,j=1}^{7,1} F_{Cost}(S_{ij}) \quad \text{EQ [35]}$$

$$P_5 = \sum_{i=8, j \in \{4,1\}}^{9,\{4,1\}} F_{Cost}(S_{ij})$$ EQ [36]

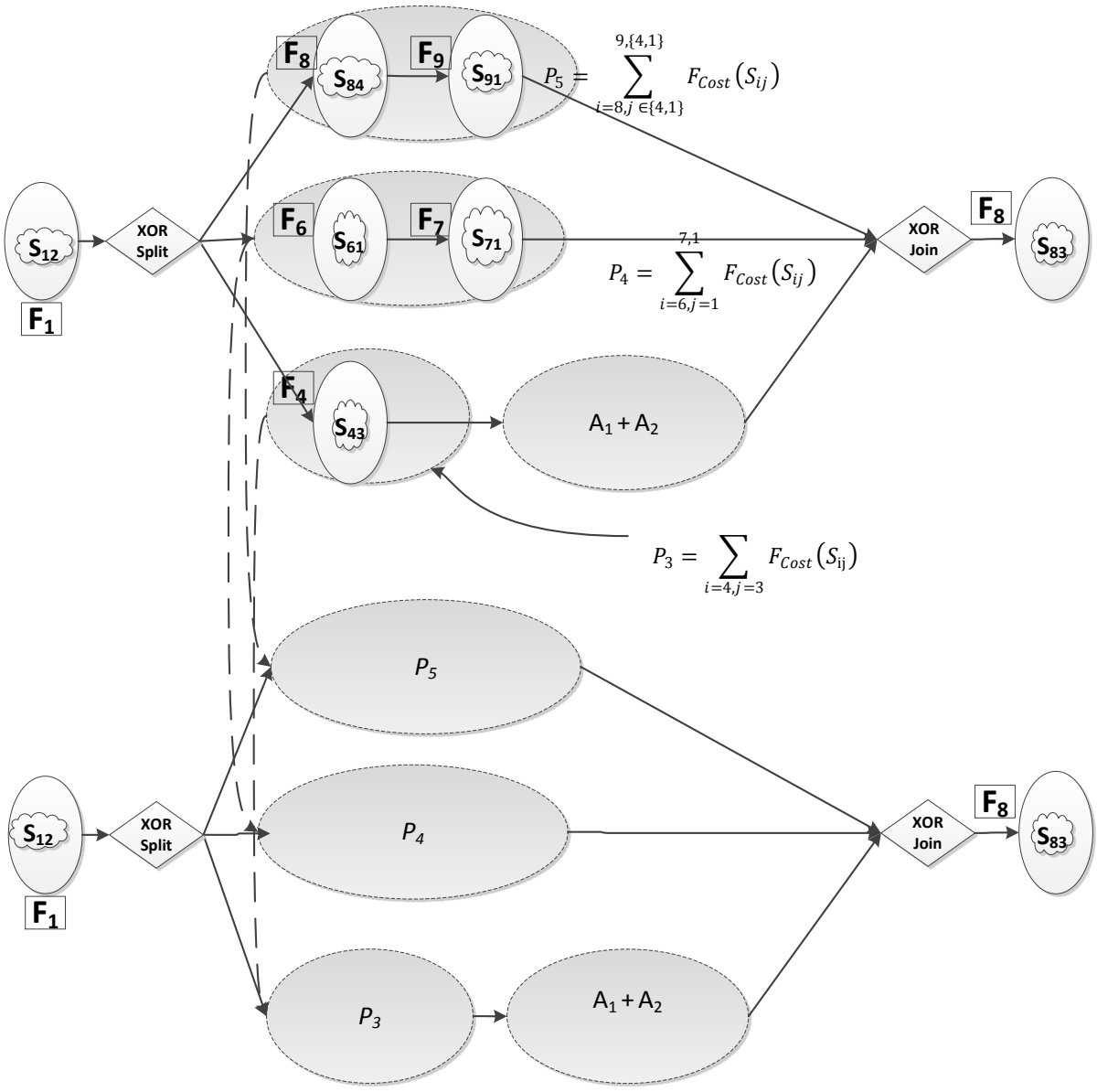


Figure 34: Calculating Cost for Parallel Workflow - Step 3

In our next step, we find the maximum cost of the sequential paths to factor into our formula.

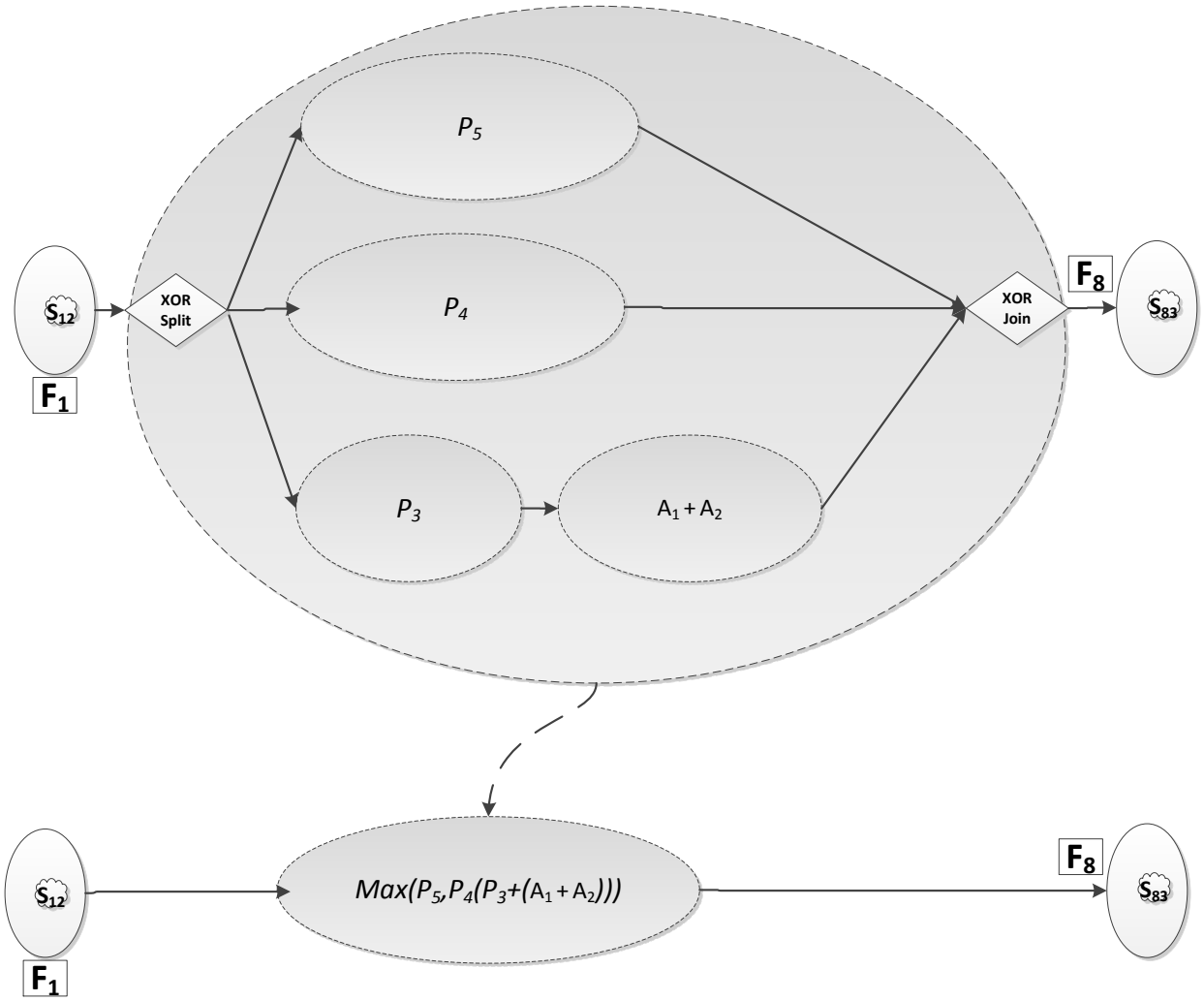


Figure 35: Calculating Cost for Parallel Workflow

For our final step, we obtain the formula:

$$TotalCost = F_{Cost}(S_{12}) * max\left(P_5, P_4, \left(P_3 + (A_1 + A_2)\right)\right) + F_{Cost}(S_{83}) \quad \text{EQ [37]}$$

To summarize, from a QoS perspective in respect to cost, the smaller the value the better. Hence, the largest value in a parallel arrangement of Sensor Clusters denotes the worst case or its upper bound. To determine the minimum cost in the XOR operation, we find the path with the minimum value as our best case or lower bound.

So we have the following: P is the cost value of the sensor clusters in a sequential path in an XOR workflow, A represents the cost value of the sensor clusters in a sequential path that are in an AND workflow and T represents the cost value of the sensor clusters in a sequential path that is not a part of any parallel workflow.

Let $\{T_1, T_2, \dots, T_h\}$ represent the cost value of the set of sequential paths that are not in any parallel workflow in the CFG.

Let $\{P_1, P_2, \dots, P_m\}$ represent the cost value of the set of sequential paths in an XOR-Split where m is the number of sequential paths for that operation.

Let z = the number of XOR-Splits of the workflow in the CFG where $= 1,2, \dots, z$. So for each r there will be a set of parallel paths in which we find the path with the maximum cost value, that is we seek to obtain $\max\{P_1, P_2, \dots, P_m\}_r$ for each r.

Let $\{A_1, A_2, \dots, A_g\}$ represent the cost value of the set of sequential paths in an AND-Split where g is the number of sequential paths for that operation.

Let y = the number of AND-Splits of the workflow in the CFG where $f = 1, \dots, y$. So for each f there will be a set of parallel paths where we obtain the sum of the cost of all paths, that is we seek to obtain $\max\{A_1, A_2, \dots, A_g\}_f$

Then our formula is:

$$TotalCost\big(CSI(G)\big) = \sum_{b=1}^{h} T_b \; + \sum_{f=1,o=1}^{y,g_y} (\, A_1 + A_2 + \cdots + A_o)_f \; + \sum_{r=1,x=1}^{z,m_z} \max(\, P_1, P_2, \ldots, P_x)_r \qquad \text{EQ [38]}$$

3.4.6. Energy Required QoS Aggregation

The energy required formula is the same as the cost formula with the exception that $F_{Energy}\ (S_{ij})$ returns the energy required to execute the sensor cluster. Since these two formulas are so similar, we will not perform the example.

3.5. Conclusion

In this chapter, we have covered several topics which continue to define the framework for implementing SOA methodologies in a WSN. First, we reviewed the strategy for equating sensors in a WSN to services in SOA using sensor clusters. The sensors cluster performs a specific function in the WSN and can be addressed by the sensor cluster head.

Second, we reviewed a powerful feature of SOA called service orchestration and defined the methodology for implementing service orchestration in a WSN. The foundation of a service orchestration is the composite function graph which is a pictorial representation of the desired

process to be executed in the WSN. Once sensor clusters are selected for the CFG, it is called a composite sensor cluster instantiation (CSI).

Third, we reviewed quality of service and network workflow operations, a related component to service orchestration. The workflow operations are divided into basic and complex workflow. We showed that with the XOR and AND conditions that we are able to implement a variety of conditional operations including the OR condition. As a result, our proofs going forward will focus on the AND and XOR conditions even if the example has an OR condition in the workflow.

Finally, we studied how to calculate QoS for cost, execution time, reliability and energy required in a workflow using incremental graph transformations and applying specific rules based on the workflow operation.

With this foundation in sensor cluster orchestration, WSN workflow operations and methods for calculating QoS in sequential and parallel workflow, we have provided a structure to achieve great flexibility and powerful manipulation of existing assets in a WSN.

In the next chapter, we will expand the capabilities of the framework by reviewing techniques to select sensor clusters dynamically for a CFG.

Chapter 4

Composite Sensor Cluster Selection

In service oriented architecture the ability to select services from a group of services that perform the same function is a feature that provides flexibility in defining new composite services. The end user can select services that best meets their quality requirements like cost and execution time for a group of services. Additionally this selection of service can be made right before execution of the composite sensor cluster. We will discuss the various methods for handling this process as well as methods for CFG planning and sensor cluster selection in this chapter.

In Chapter five we extend our SOA WSN framework with the introduction of composite sensor cluster selection for composite function graphs. In section 4.1 we discuss the concept of functionally equivalent services which provide us with many options for sensor cluster selection for a CFG. In section 4.2 we review the centralized versus the decentralized methods for service selection. In section 4.3 we examine the QoS vector and its relationship to the service selection problem. Finally in section 4.4 we provide our concluding remarks.

4.1. Functionally Equivalent Services

Composite sensor cluster selection is a method used to select sensor clusters for functions in a CFG. We introduce the concept of functionally equivalent sensor clusters which are sensor clusters that perform the same service but differ in QoS levels. This provides the client with the opportunity to select sensor clusters that meet their functional needs as well as specific quality

criteria. As a result, service providers must provide sensor clusters which are 'quality aware' in order to ensure that client service levels are met.

Sensor clusters are selected based on quality criteria and once the execution of the sensor cluster is performed, real time statistics may be captured for each quality criteria and stored in relational table for subsequent sensor selection processes.

A sensor cluster execution request requires that we select services for each function in our CFG that meet requested quality criteria. The request can specify that the CSI meet a certain level of quality for the entire CFG (globally) or that individual sensor clusters meet a certain level of quality (locally). Our research focuses on satisfying multiple constraints globally for the CFG and algorithms are required to select the sensor clusters that meet the global quality criteria. For example, a CFG request may require that the following criteria are met for the overall CSI:

Cost <= $100
Execution Time < 10ms
Reliability > 95%

Sensor clusters must be selected that meet these constraints for the overall CSI and an algorithm is required to select these services such that the overall cost, execution time and reliability of all the services combined in the CSI meet the overall objectives and constraints.

As services execute, many quality metrics can be captured. To keep the quality metrics current, these numbers should be stored in a relational table and updated regularly. Sensor cluster information management and monitoring consists of this process.

4.1.1. Logical View of the WSN Composite Function Graph

As shown in Figure 36: WSN Composite Function Graph – Logical View, a WSN may consist of many sensor clusters and the CFG is concerned with a specific set of these sensor clusters that can perform the functions in the process. The same graphical representation of a process workflow in SOA can be used for a WSN. The CFG for a WSN G, consist of a set of vertices representing functions, directed edges that represent data flow and workflow constructs that indicate conditional operation for a composite sensor cluster.

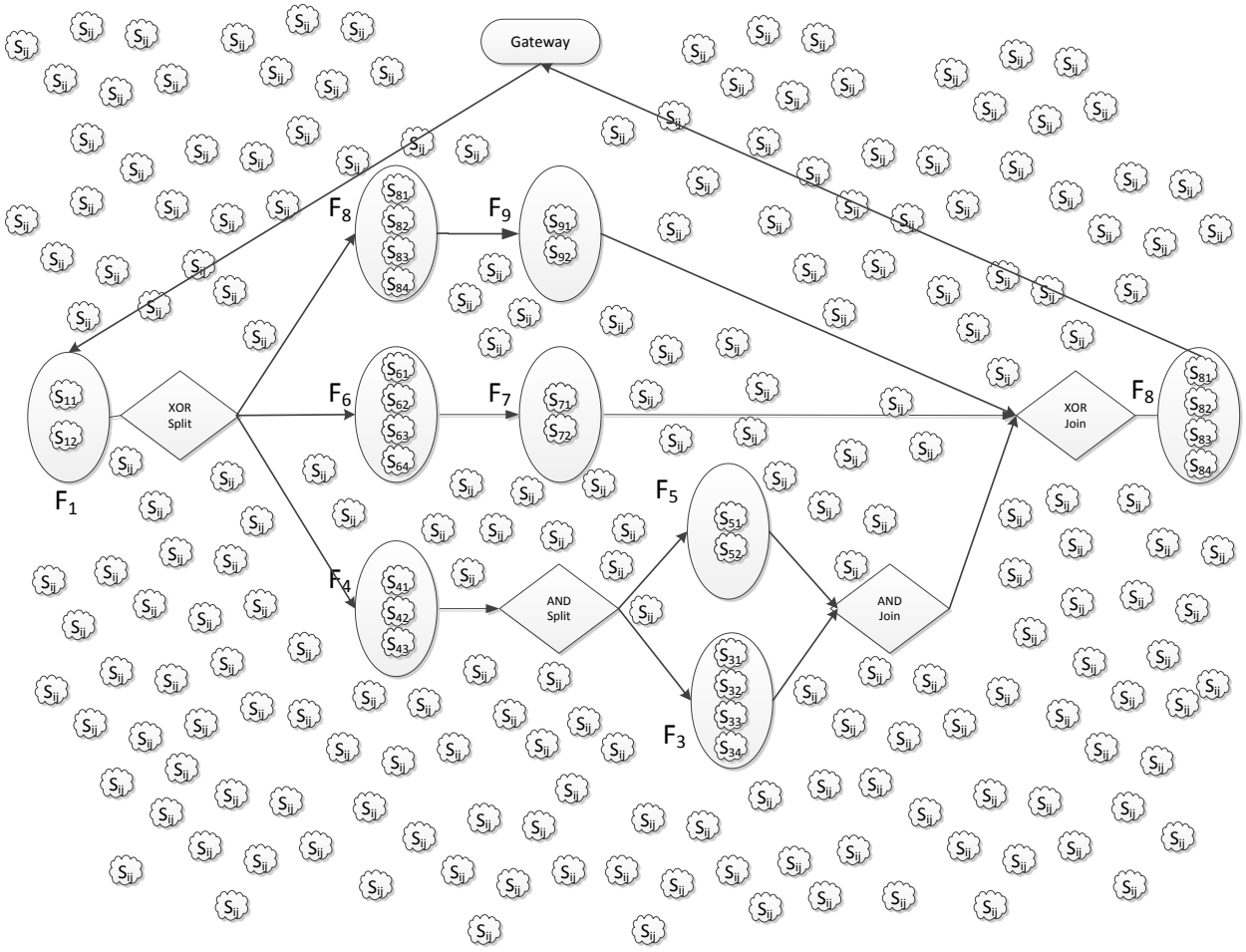


Figure 36: WSN Composite Function Graph – Logical View

The logical view of the CFG depicts the relationship between sensor clusters in the workflow. The CFG representation of the WSN is defined as $G = \{V, E\}\ where\ \mathrm{V} = \{F_1, F_2, \dots, F_p\}$, p is the number of vertices in a WSN graph G, $F_1, F_2, \dots, F_p$ are functions, and E consists of all the ordered pairs of vertices for which the output of F_i has an element in common with the input of F_j. $i.e.,\ E = \{\, e_{i,k} \mid 1 <= i, k <= p \}$,

$$where\ \{e_{i,k}\} = \begin{cases} \{(\mathrm{i}, \mathrm{k})\}\ if\ output\ from\ F_i\ \cap input\ F_k\ \neq\ \emptyset \\ \emptyset\ otherwise \end{cases}$$

4.1.2. Physical View of the CFG Representation

In the physical view of the CFG, the specific paths between sensor clusters are identified as edges. In addition, in the physical representation of the WSN graph at each function F_i, there are several sensor clusters S_{ij} that can perform the function at vertex V. Each sensor cluster has a communication link to its successor candidate sensor clusters in the graph as illustrated in Figure 37: WSN Composite Function Graph – Physical View. The graph can be represented as:

$$G = \{V, E\}\ where\ V = \{\mathrm{S}_{ij}, \mathrm{i} = 1, \dots, \mathrm{p};\ \mathrm{j} = 1, \dots, q_i\}$$

Where V is a set of vertices in a WSN graph G and

$$E\ \subseteq V\ \times V, and\ E = \{\, e_{ij,kl} \mid \mathrm{i} = 1, \dots, \mathrm{p};\ \mathrm{j} = 1, \dots, q_i;\ k =\ i + 1, \dots, p;\ l = 1, \dots, q_k \}$$

$$where\ \{e_{ij,kl}\} = \begin{cases} \{(\mathrm{ij,kl})\}\ if\ output\ from\ S_{ij}\ \cap input\ S_{kl}\ \neq\ \emptyset \\ \emptyset\ otherwise \end{cases}$$

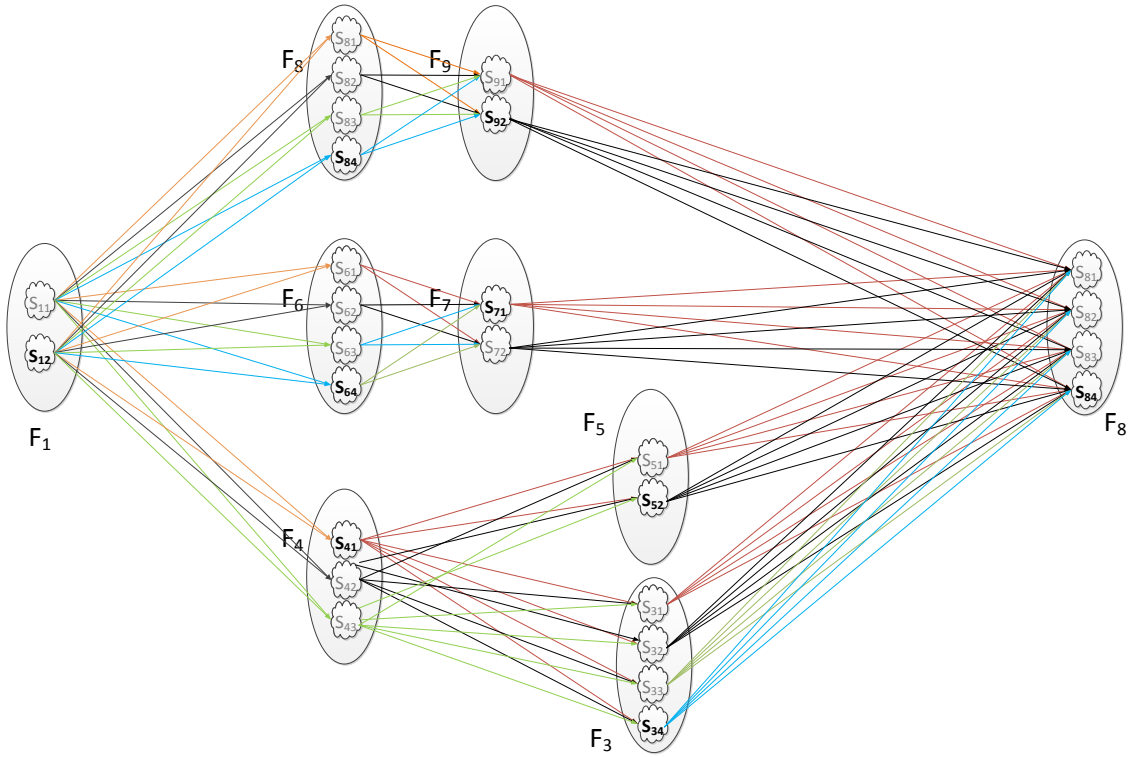


Figure 37: WSN Composite Function Graph – Physical View

Sensor cluster S_{ij} performs a service and produces an output. This output can be used as input to another sensor cluster. A composite service $CSI(G)$ is a sequence of one or more sensor clusters that can depend on branches that rely on the output from one sensor cluster to another.

4.1.3. Energy Required

Each edge in the physical WSN represents a communication link to its successor candidate sensor clusters in the graph. For a sensor cluster to communicate to a successor candidate, there is an energy cost associated with the transmission. The energy required as depicted in Figure 38: Energy required from one sensor cluster to its successor, is the energy necessary for sensor cluster

S_{11} to transmit its output to its successors. The lower the energy value the more efficient the execution and the route. We want to select the route that uses the least amount of energy. This will minimize the energy used from the energy available. For example, let's assume that that sensor S_{11} is chosen for sensor cluster F_1 and S_{84} is chosen for sensor cluster F_8 in a CSI. Then the amount of energy required to transmit the output from S_{11} to S_{84} is 3. The total energy available (EA) for sensor cluster S_{11}, at the beginning of the execution, is 8. After S_{11} executes and the value is transmitted to S_{84}, the total EA for S_{11} will be 5, as 3 units of energy were used to transmit the output to S_{84}.

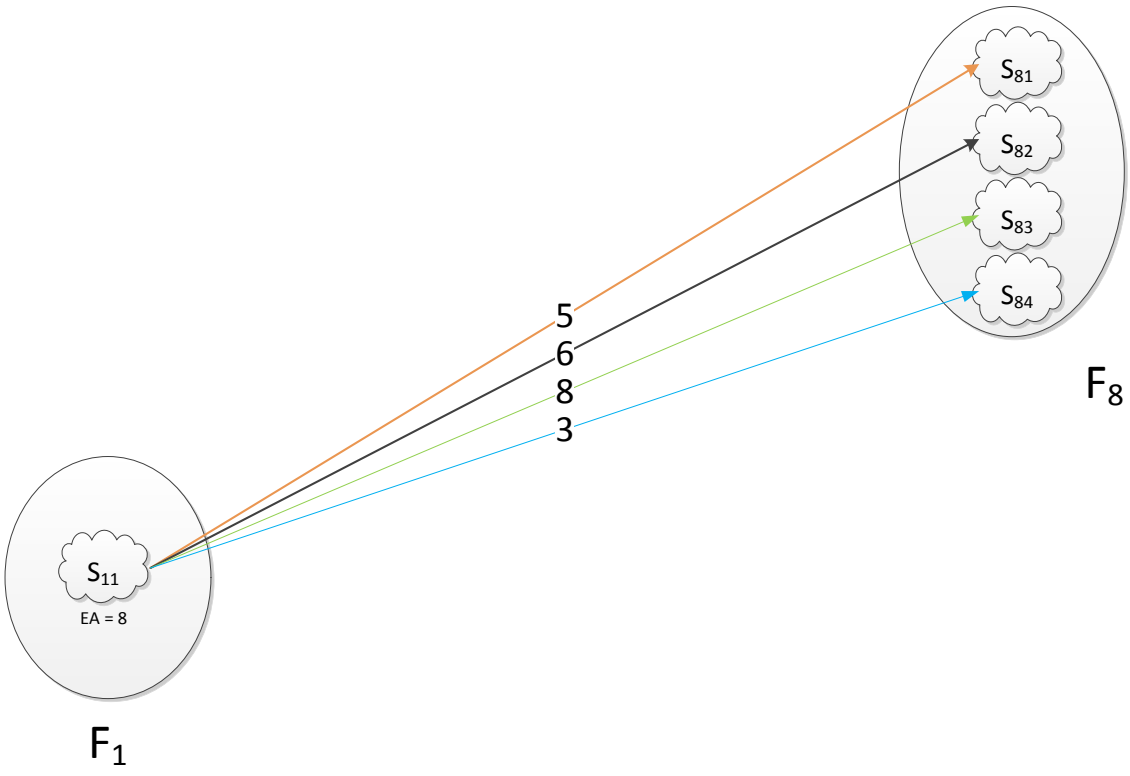


Figure 38: Energy required from one sensor cluster to its successor

4.2. Centralized versus De-Centralized Sensor Cluster Selection

It is desirable to select sensor clusters for a composition that meet quality requirements. Sensor clusters in a WSN can be added, removed or relocated at any time which leads to a dynamic configuration of the WSN. There are two methods for selecting sensor clusters for a CFG. The selection can either be made prior to the execution of the CSI during build time or during the execution of the CSI at bind time. We will examine each approach, identify the advantages and disadvantages and discuss our method for sensor cluster selection.

4.2.1. Centralized Selection

In the case of selection prior to execution, a centralized source is used to select sensor clusters for the composition using a current snapshot of quality metric data [35]. Since there is a delay from the time of selection to the time of execution, it is possible that one or more sensor clusters that were selected may have changed state from available to unavailable by the time they are to be executed. The sensor cluster can change state to unavailable for many reasons, such as sensor cluster relocation, energy depletion or network failure [14]. The disadvantage of this method is that the service selection is inflexible towards re-engineering of the composition during execution which can lead to an increased failure rate of the execution of the service composition [36], [37]. The advantage of this method is that the heavy weight processing necessary to perform sensor cluster selection using the database is handled by the centralized node or Gateway and not at the sensor clusters.

As shown in Figure 39: Centralized Source sensor cluster selection, the gateway receives the request for the CFG. It selects all sensor clusters for the composition from the sets of functionally equivalent sensors and returns the selected group for execution.

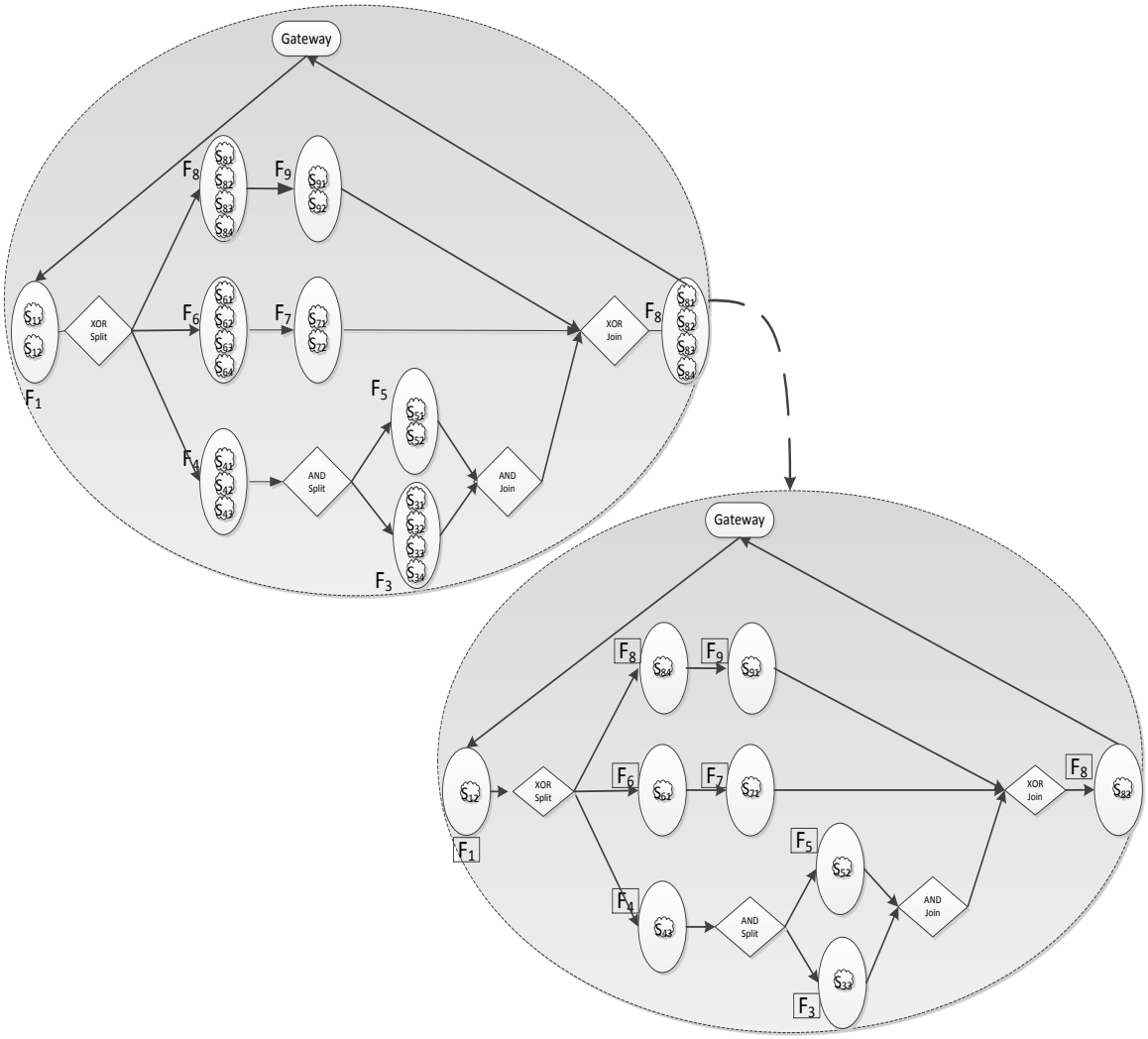


Figure 39: Centralized Source sensor cluster selection

4.2.2. Decentralized Sensor Cluster Selection

In the decentralized case, sensor cluster selection is made during the execution of the service composition using a decentralized selection approach where each sensor cluster in a composition, selects the subsequent sensor cluster that must receive its output based on a set of quality criteria [37]. The selection of sensor clusters occurs at each step of the service composition execution and the quality cost is computed at each selection. The advantage of this approach is that service composition is more resilient to network faults and reaction is quick to the dynamic configuration of the WSN. This will result in higher availability and an increase in reliability of the service composition.

However, the disadvantage of this approach is that each sensor cluster must have information about each of its successor nodes and it must be able to make a selection and compute the quality cost at that moment in time.

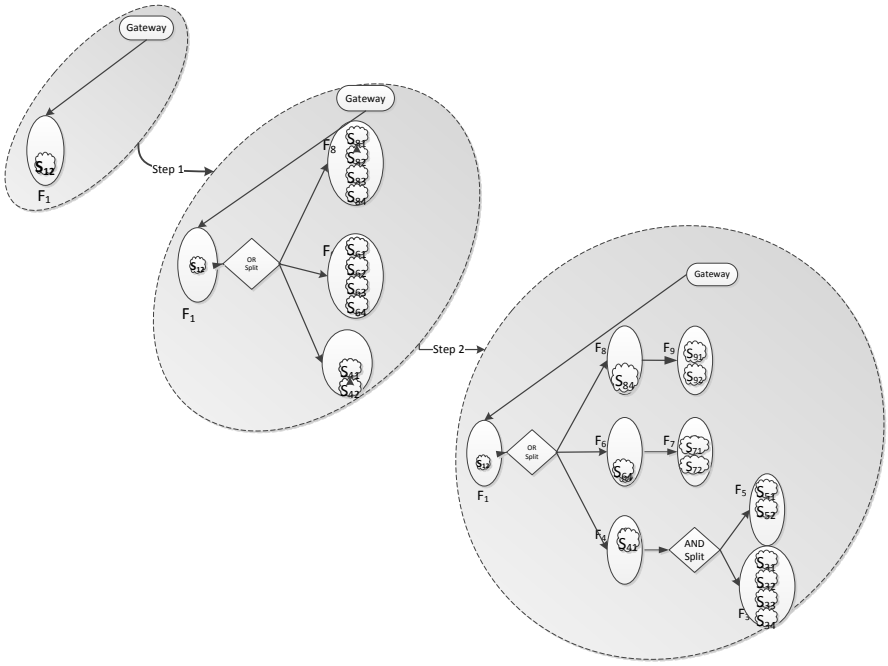


Figure 40: Decentralized Approach to Sensor Cluster Selection - Step 1 & Step 2

In Figure 40: Decentralized Approach to Sensor Cluster Selection - Step 1 & Step 2, sensor cluster F_1 must select the next set of sensor clusters for functions F_8, F_6 and F_4. Next, function F_8 must then select the next sensor cluster for function F_9 and function F_6 must select the next sensor cluster for function F_7 and so on, until all sensor clusters have been chosen as depicted in Figure 41: Decentralized Sensor Cluster Selection - Final Step .

The decentralized approach results in significant communication and processing overhead for each sensor cluster. It has the potential to result in a less cost efficient and energy demanding composition [37]. It is for these reasons that we select a centralized approach to sensor cluster composition through the use of a broker. In our approach, we select sensor clusters at runtime using the centralized method as we assume that changes in the network will be negligible between

selection and execution of the sensor clusters. Our contribution to the research is to develop a sensor cluster strategy that does not require heavy processing at the sensor cluster and promotes energy conservation to help increase the network lifetime, hence the decentralized approach is not used.

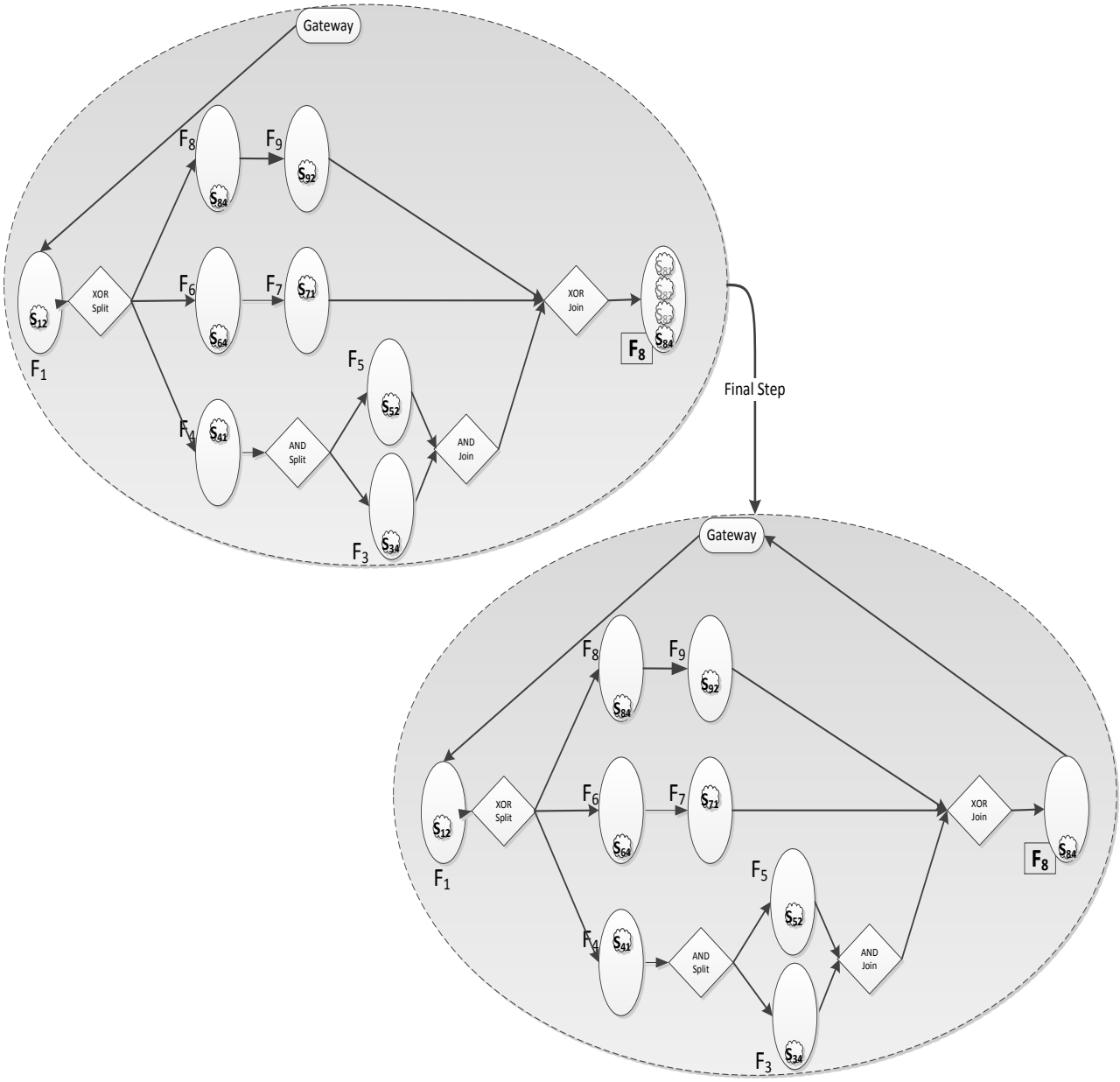


Figure 41: Decentralized Sensor Cluster Selection - Final Step

4.3. QoS Vectors

There are many challenges that affect the quality of a WSN. One challenge is the availability of the sensor clusters of the WSN for a particular service need. WSNs typically have limited

resources such as memory, battery life, communication capabilities, transmission power, buffer size and computing resources [17], [18], [11]. The uncertainty of availability of sensor clusters in a WSN has a rippling effect on other quality characteristics like reliability, time to execute and reusability.

There are four attributes of quality of service that we consider in monitoring and using as a selection criterion in our composite function plans. They are reliability, time, cost and energy required. In addition, we aim to minimize the amount of energy required to execute the composite sensor cluster. When selecting a sensor cluster it will be based on these four metrics forming a QoS reference vector. Each sensor cluster will have this set of QoS parameters associated with it. We present the metrics for reliability, time, cost and energy in the paragraphs that follow. Our vector consist of four elements $(F_{Reliability}, F_{ExecutionTime}, F_{Cost}, F_{Energy})$ for each sensor cluster.

In general for each of the p functions, there are several sensor clusters that can perform that function. For each quality metric, a series of vectors can be used to represent a quality value for all sensor clusters in the composite function plan. Let Q_{ij} represent a generic quality for a sensor cluster in our quality vectors. The vectors consist of a series of quality values for each candidate service and the size of the vector will vary based on the number of services that can perform the function. For each function i, there can be q_i sensor clusters available that can perform that function. A generic quality of the service can be represented as Q_{p,q_p}. If there are m qualities associated with a service, then there will be $m * p * q_p$ quality metrics for the composite function.

4.4. Conclusion

A composite sensor cluster function provides us with the ability to create new processes from existing sensor clusters dynamically. This functionality is extremely powerful with respect to creating new applications on the fly. It requires that we have loosely coupled applications and that we have workflow and other tools to compose the new service in real time. Having more than one service option for a particular function gives flexibility required for service selection that can meet the desired quality constraints. We have discussed the concept of functionally equivalent sensor clusters that provide us with the necessary options for a composite sensor cluster instantiation in section 4.1. In addition we've reviewed the options and types of resources that are required to select a sensor cluster from the set of functionally equivalent services in section 4.2. Finally, in section 4.3 we reviewed QoS vectors and QoS formulas required for managing the quality criteria for each of the sensor clusters in a functionally equivalent space.

Chapter 5

Sensor Cluster Selection Possibilities & Algorithms

In sensor cluster selection, there are three possibilities for the selection of sensor clusters: sensor cluster selection with no constraints, sensor cluster selection with one global constraint and sensor cluster selection with multiple global constraints. We must also take into account the type of workflow of the CFG when performing sensor cluster selection; i.e. workflow with conditional operations or parallel paths of execution which is handled differently than sequential workflow [38] [39] [40]. The selection problem can be mapped to various versions of Knapsack problems. In this chapter we review the sensor cluster selection possibilities and the associated knapsack problems and algorithms. Knapsack problems are well researched combinatorial problems and in section 5.1 we provide a short summary of the relevant research in knapsack problems. In section 5.2 we review sensor cluster selection with no constraints. In section 5.3, we review sensor cluster selection with one global constraint and in section 5.4 we review sensor cluster selection with multiple global constraints. Finally in section 0 we provide our concluding remarks for this chapter.

5.1. Survey on Knapsack Problems

For single and multiple constraint sensor selection, we can map these problems to the set of Knapsack problems which are well-researched problems in combinatorial optimization. Knapsack problems are in the family of NP-Hard problems. The problem statement for all knapsack problems, in general, require that a subset of a group of items be chosen in a way that maximizes a

corresponding profit sum while not exceeding a given limit or capacity to the knapsack. There are many variations of the knapsack problem. We present in Table 1: Family of Knapsack Problems, a subset of the variations of knapsack problems and a survey of research that provides solutions to these problems.

Knapsack Problem	Description	Research
0-1 knapsack problem	There are j items with profit p_j and weight w_j for each that are packed in one or more knapsacks of capacity c Choose a subset of the j items such that the corresponding profit sum is maximized without having the weight sum exceed the capacity c [41]	Horowitz & Sahni [38], Nauss [40], Martello and Toth [39], Pisinger [41]
Bounded knapsack problem	There are j items with profit p_j and weight w_j for each that are packed in one or more knapsacks of capacity c Choose a subset of the j items such that the corresponding profit sum is maximized without having the weight sum exceed the capacity c We have a bounded amount m_j of each item type j, where x_j is the amount of each item type to be included in the knapsack in order to obtain the largest objective value [41]	Martello & Toth [42], Ibarra & Kim [43], Pisinger [41]
Multiple-Choice Knapsack Problem	Items with profit p_j and weight w_j are packed in one or more knapsacks of capacity c Choose a subset of the items such that the corresponding profit sum is maximized without having the weight sum exceed the capacity c	Pisinger [44], Klamroth & Wiecek [45], [46] , [40]
Multiple constraints multiple dimension knapsack problem	In this case, each item j has a profit p_j and q multiple weights $w_j^m \; where \; m = 1, \dots, q$ There are multiple capacity limits c_m for each class i that must be satisfied. Items with profit p_j and weight w_j^m are packed in one or more knapsacks of capacity c_m Choose a subset of the items such that the corresponding profit sum is maximized without having the weight sums of w_j^m exceed the capacity c_m [41]	Sibhi, [47], Mozer [48], [49], [36], [21], [50],

Table 1: Family of Knapsack Problems

Pisinger [41] provides a summary of solutions and their associated algorithms to knapsack problems. There are several techniques used to solve knapsack problems. Branch and bound algorithms provide a complete enumeration where bounds are used for fathoming nodes that cannot lead to an improved solution. Dynamic programming (DP) finds solutions beginning with a small part of the problem and enlarges the problem iteratively until all feasible solutions are identified. DP backtracks through the solution space to find the optimal solution. Breadth first is another technique used to solve knapsack problems through enumeration with the addition of some dominance rules. State space relaxation, is a method that uses dynamic programming relaxation where the coefficients are scaled by a fixed value which leads to efficient approximate algorithms. Another option for solving knapsack problems involves preprocessing where variables may be a priori at their optimal values using some bounding tests to exclude certain values of the solution variables.

5.2. Sensor Cluster Selection with No QoS Constraint

In sensor cluster selection with no constraints the client does not specify a constraint. The selection of sensor clusters is arbitrary in this instance. That is, any sensor cluster that performs the function will be sufficient regardless of the execution time, reliability or cost. We must pick a sensor cluster for each function, however, the choice is arbitrary. This selection type is represented by the formula:

$$CSI(G) = \sum_{i}^{p} x_{ij} * S_{ij} \qquad j \in \{1, \dots, q_i\}$$

EQ [39]

Where $CSI(G)$ is the composite Service G. x_{ij} = 1 if the sensor cluster is selected and 0 otherwise – in this case the selection is arbitrary.

5.3. Sensor Cluster Selection with One Global Constraint

In the sensor cluster selection with one global constraint the client supplies one constraint which must be satisfied by the entire composite sensor cluster instantiation. This one constraint is called a global constraint. The problem can be mapped to a Multiple Choice Knapsack Problem (MCKP).

The MCKP states that given a set of items in several classes where each has a profit and a knapsack with a weight limit, the problem is to select one item from each class to be placed in the knapsack within the weight limit while achieving the highest total profit.

The QoS selection problem for WSNs with one global constraint is to select one sensor cluster for each function to construct a composite sensor cluster instantiation that meets a user's QoS constraint and minimizes the amount of energy required during the execution of the CSI. The efficiency of the CSI is determined by the amount of energy required by each sensor cluster. The objective is to select sensor clusters that use a minimum amount of energy so that the overall requirement to execute the CSI is minimized and potential increases in the lifetime of the WSN can be achieved.

Several researchers have proposed solutions to the single constraint QoS selection problem. Finding exact solutions to MCKP is classified as NP- Hard. In Table 2: Single Global Constraint Multiple Service Selection Problem Survey we provide a brief summary of current research on single constraint QoS Selection problems and solutions.

Solution Name	Solution	Researcher
Discarding Subsets	Backtracking based algorithm that uses search trees which consists of nodes representing possible candidate pairs and a task. The idea behind this algorithm is to cut subtrees representing unfavorable combinations.	Jaeger & Rojec-Goldmann [46]
Bottom-Up Approximation	Maps the single constraint selection problem to a research constrained project scheduling problem which is a project management problem where each task must be assigned to the available workers in order to complete the whole project. Constraints or optimization such a finishing the project quickly introduce complexity to the problem. The bottom up method, sorts candidates y QoS, assigns candidates with the best choice for the constraint at that point then uses the SAW method to refine the choice until no other choices are available.	Jaeger & Rojec-Goldmann [46]
Pattern Based Selection	The algorithm determines the best assignment considering each pattern in a composition separately	Jaeger, Muhl, Golze [33]
Convex Hull	A heuristic algorithm that computes a near-optimal solution by constructing the convex hull [9] of related points for approximation.	Yang et al [51], Vanitha & Palanisamy [16]

Table 2: Single Global Constraint Multiple Service Selection Problem Survey

5.3.1.1. Mapping One Global Constraint Service Selection to MCKP

To correlate the sensor cluster selection problem to a Multi-Choice Knapsack Problem, each function- in a CFG is mapped to a group in the MCKP. One sensor cluster from each function is selected as in MCKP one item from each group is selected to be contained in the knapsack. An example of mapping a global constraint from CFG to MCKP is to map the execution time of each sensor cluster to the weight of an object in MCKP. A client will specify the overall execution time of the CFG which correlates to the total weight in MCKP. The objective is to select sensors clusters for each function in the CFG that do not exceed the clients constraints while using the minimum amount of energy required to execute the CFG. To summarize:

1. The execution time of each sensor cluster is mapped to the weight of the object in MCKP
2. The energy available and the energy required to transmit information from one sensor cluster to another is mapped to the profit in MCKP.
3. A user's end-to-end execution time requirement for the composite service is considered the weight limit of the knapsack
4. The objective is to minimize the total energy required by the composite sensor cluster while meeting the client's execution time constraint. We use the following formula in EQ[40] to calculate the Energy Efficiency for each sensor cluster.

$$E_{ij} = \frac{EnergyAvail_{ij} - EnergyReq_{ij}}{EnergyAvail_{ij}} \quad \text{EQ [40]}$$

Our objective is to maximize the energy efficiency in our solution. The formula presented in EQ[41] represents the single constraint sensor cluster selection. The single QoS constraint in our example is execution time.

$$max\left[\sum_{i,j=1}^{p,q_i} x_{ij} * (E_{ij})\right]$$

$$Client_{ExecutionTimeConstraint} \geq \sum_{i,j=1}^{p,q_i} x_{ij} * \left(F_{ExecutionTime}(S_{ij})\right)$$

$$\sum_{j=1}^{q_i} x_{ij} = 1 \quad , x_{ij} \in \{0,1\}, \qquad i = 1, \ldots, p, \qquad j \in \{1, \ldots, q_i\}$$

EQ [41]

- S_{ij} is the service, i is a function in the composition function plan
- j is a service in the function class
- p is the number of functions in the composite function plan
- q_i is the number of services in the composite function class
- $F_{ExecutionTime}(S_{ij})$= execution time measurement for S_{ij} (weight)

- E_{ij} is the efficiency requirement of the service (utility)
- Where x_{ij} = 1 if S_{ij} is selected for its function and 0 otherwise
- The sum of all execution times of each service in the composition must be less than the $Client_{ExecutionTimeConstraint}$

5.3.2. Sensor Cluster Selection with Multiple Global Constraints

For a composite service that has p service classes in a composite service plan and with m QoS constraints, the service selection can be mapped to a Multi-dimension Multi-choice Knapsack Problem (MMKP). In MMKP, suppose there are N object groups, each group has *i* (1 <= *i* <= N) objects, each object has a profit E_{ij} and requires resource S_{ij},. The MMKP is to select exactly one object from each object group to be placed in the knapsack so that the total efficiency is maximized, while the total resources used are less than the constraint. In general the formula for MMKP is shown in EQ[42].

$$max\left[\sum_{i,j=1}^{p,q_i} x_{ij} * \left(E_{ij}\right)\right] \qquad \text{EQ [42]}$$

Subject to

$$Client^{\alpha}_{QosConstraint} \geq \sum_{i}^{p} F^{\alpha}_{QoS}\left(S_{ij}\right) * x_{ij} \qquad (\alpha = 1, \dots, m)$$

$$\sum_{j=1}^{q_i} x_{ij} = 1 \quad x_{ij} \in \{0,1\} \quad \mathrm{i} = 1, \dots., \mathrm{p} \quad j \in \{1, \dots, q_i\}$$

x_{ij} is 1 if service j is selected for function S_i and 0 otherwise

$F^{\alpha}_{QoS}\left(S_{ij}\right)$ = $[Q_{i_1}, \dots, Q_{i_{qi}}]$ is the QoS resource metric of each service for function i

m is the number of resources

In Table 3: MMKP Problem Survey we provide a summary of research in this area.

Solution Name	Solution	Researcher
Genetic Algorithm (GA)	Uses a fitness function to compute the QoS values of the selected services and compares the results of the fitness function with the global constraints. If the QoS is satisfied and the stop criteria of the GA are met a solution is found.	Confora, Di Penta, Esposito, Villani [52]
BST	Tree based service selection algorithm to select optimal solution. Takes into account the need to re-select services in the event they become unavailable during the selection process.	Oh, Baik, Kang, Choi [36]
Selection on QoS Reference Vector	Two selection algorithms are presented. One for the best choice and the other for a good-enough choice. The optimal choice selection method is called Fast Selection mechanism and it uses integer programming for selecting services without constructing all service combinations.	Wu, Chi, Xu [21]
Flow Based Service Selection	Formulate the service selection problem as a Linear Programming optimization problem that scales as the number of users, request volumes and/or services grow.	Cardellin, et al [53]
DQos	QoS criteria for web service selection are formulated as a Multiple Attribute Decision Making problem. Uses decision matrix and the following decision modes subjective weight mode, single weight mode, object weight mode and subjective-objective weight mode.	Hu, Guo, Wang, Zou [54]
BBLP	A branch and bound method for finding the optimal solution to MMKP by the iterative generation of a search tree. Each node of the tree represents a solution state where some service classes are fixed while others are free.	Khan [55]
WS-Heu	Heuristic algorithm that finds an initial feasible solution, improves the solution by feasible upgrades and then improves the solution by infeasible upgrades followed by downgrades .	Yu [56]
Branch and Bound	A branch and bound algorithm that generates an initial feasible solution as a starting lower bound, and at different levels of the search tree determines an intermediate upper bound obtained by solving an auxiliary problem called $MMKP_{aux}$. Items are fixed at different stages of the program as a strategy.	Sbihi [47]
HEU	A heuristic algorithm that starts with finding a feasible solution then uses an aggregate resource for selecting items to pick. It improves the solution iteratively by exchanging picked items. It finds near optimal solutions by upgrading feasible solutions.	Khan, et al [49]

Table 3: MMKP Problem Survey

5.4. MMKP Mapped Multi Constraint QoS Selection Problem

In mapping MMKP to our selection problem, each function *i* in a composite service is viewed as an object group, each service *j* in a function class is viewed as an item in the object group and each QoS attribute Q of the service is mapped to the resources required by the object in MMKP. The utility (energy efficiency) a candidate is mapped to the profit of the object and the users QoS constraints are the resources to be placed in the knapsack.

Specifically, mapped to our example, the Multi Constraint Service Selection problem is represented by the formula in EQ[43].

$$max\left[\sum_{i,j=1}^{p,q_i} x_{ij} * \left(E_{ij}\right)\right]$$

Subject to

$$Client_{ExecutionTimeConstraint} \geq \sum_{i,j=1}^{p,q_i} F_{ExecutionTime}\left(S_{ij}\right) * x_{ij}$$

$$Client_{CostConstraint} \geq \sum_{i,j=1}^{p,q_i} F_{Cost}\left(S_{ij}\right) * x_{ij}$$

$$Client_{ReliabilityConstraint} \leq \sum_{i,j=1}^{p,q_i} F_{Reliabilityt}\left(S_{ij}\right) * x_{ij}$$

$$\sum_{j=1}^{q_i} x_{ij} = 1 \quad x_{ij} \in \{0,1\} \quad \mathrm{i} = 1, \ldots, \mathrm{p} \quad j \in \{1, \ldots, q_i\}$$

EQ [43]

5.5. Conclusion

The knapsack problems are well researched combinatorial problems that have many applications. We have reviewed several approaches to solve knapsack problems and in the chapters that follow we will discuss our implementation of the global QoS multi-criteria, multi-dimension selection problem for composite function graphs.

Chapter 6

SOA Sensor Cluster Management

In Chapter six we reviewed several topics on sensor cluster selection algorithms and now we would like to review our dynamic programming approach to these problems. However before we discuss our implementation and the algorithms and complexity associated with it, we would like to review the application that was developed that implements our dynamic programming solution.

As discussed in chapter five, we have decided to use a centralized source in the WSN to perform sensor cluster selection. The Sensor Cluster Selection Manager (SCSM) is considered to be the heart of the middleware for the WSN and it is located at the Gateway. The SCSM performs many functions to manage the usage of the sensor clusters in the WSN and it acts as the interface between the client and the WSN. The SCSM accepts requests from the client to find sensor clusters for a composite function that satisfy the client's QoS constraints in the WSN. The function of the SCSM is not limited to selecting sensor clusters for the CFG that meet the client's QoS constraints; it must also maximize the energy efficiency of the WSN to prolong the life of the network. The functions that that the SCSM performs or manages are shown in Figure 42: Sensor Cluster Manager Overview and are as follows:

a) Relational Database Management System

b) Client Request Module

c) Composite Function Graph Manager

d) Sensor Cluster Publish & Subscribe Manager

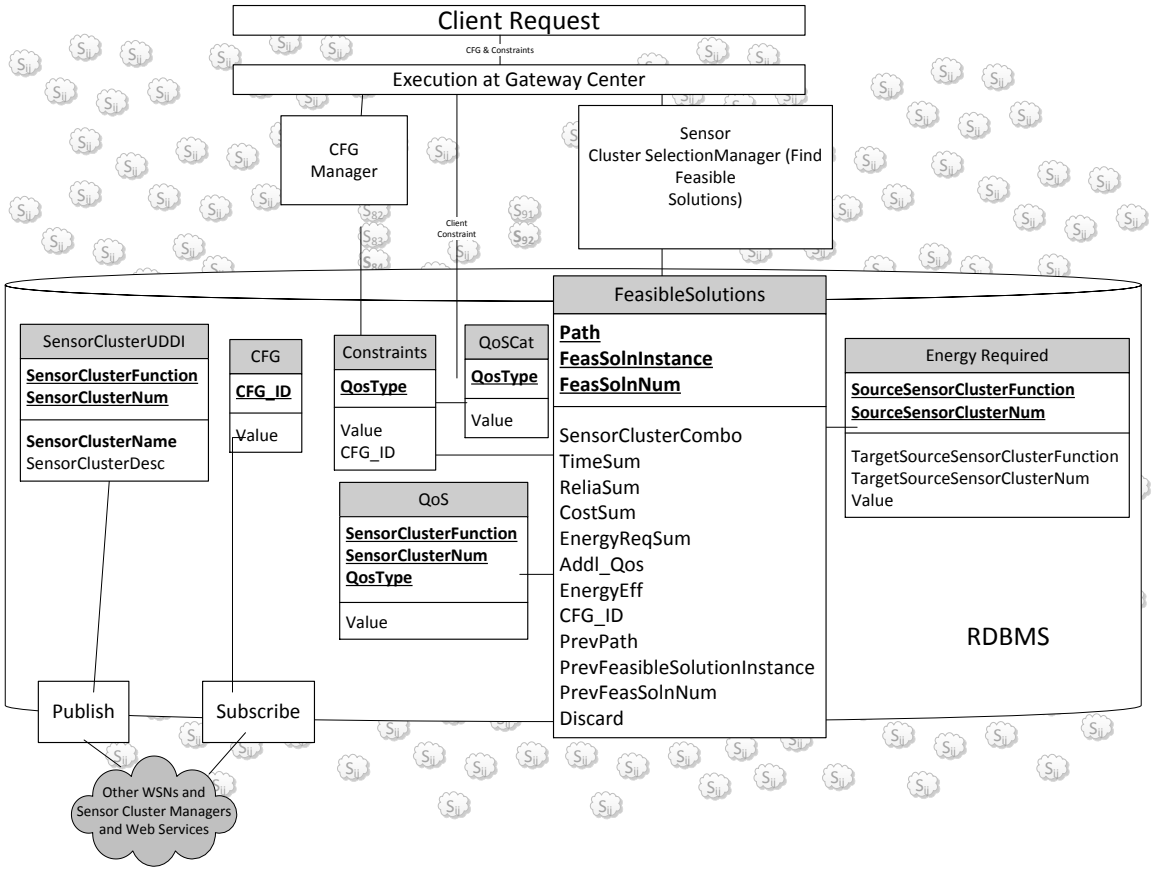
e) Energy Route Manager

f) Sensor Cluster QoS Manager

g) The Processing Center

h) Sensor Cluster Selection Manager

The architecture for the SCSM is presented Figure 42: Sensor Cluster Manager Overview. We have implemented a combination of Java and C++ programs which simulate the role of the SCSM. We provide an overview of each function in the sections that follow.



i)

Figure 42: Sensor Cluster Manager Overview

6.1. **The Relational Database Management System**

The Relational Database Management System **(**RDBMS**)** consists of several tables that are required to manage the sensor clusters in the WSN. The SCSM maintains repositories that contain data and metrics relative to composite sensor cluster selection and execution. The SCSM will share this information with other SCSMs and web services brokers on the network. The SCSM also handles the communication to and from the RDBMS to other functions.

6.2. **Client Request Module**

The Client Request Module is the client interface to the WSN. The client can submit a CFG request along with the required constraints to the SCSM. The SCSM will accept CFG requests from clients through this module. Any communication on the status of the request and other client related communication are performed through this module. For our example, we will assume the client has submitted the CFG request shown in Figure 43: Client Request CFG1. In our example we use five sensor clusters per function.

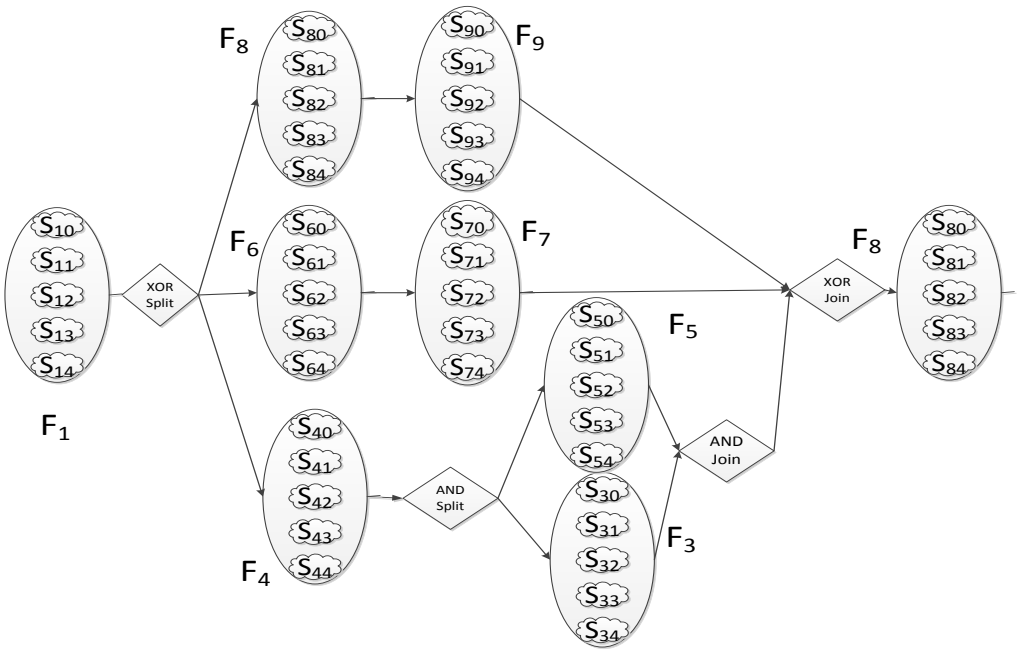


Figure 43: Client Request CFG1

6.3. **The Composite Function Graph Manager**

The Composite Function Graph Manager (CFG) keeps track of and manages a repository of all composite function graphs. The composite function plans are stored in the CFG repository. When a client submits a request for a CFG, the CFG manager will translate the plan into functions and workflow operations. If the CFG does not exist in the CFG table, the CFG manager will create it. The element name Value in the CFG table contains a string representation of the CFG that consists of a combination of SensorClusterFunction, workflow elements and parenthesis. The CFG Value is passed to the Sensor Cluster Selection Manager where it is parsed to identify the functions, workflow and paths of the CFG. The string in Table 4: CFG Table represents the CFG pictured in Figure 8: Composite Function Graph

CFG	
PK	**CFG_ID**
	Value

Table 4: CFG Table

$$1\ or\ ((8,9)(6,7)\big(4\ and\ (5,3)\big))8 \qquad \text{EQ [44]}$$

6.4. Sensor Cluster Publish & Subscribe Manager

To implement SOA in a WSN, it is necessary to provide the "publish, find and subscribe" paradigm for sensor clusters. As each sensor cluster performs a function in the WSN, in order for a client to find the sensor cluster, the details of that sensor cluster must be published in a repository. [1] [18] have proposed WSDL-like documents in a repository similar to the UDDI for a SOA WSN.

In our research, we use a table called SensorClusterUDDI (as seen in Table 5: Sensor Cluster Registry Table) which is a condensed version of the UDDI with custom fields for a WSN. The table contains the following fields:

Energy Required
SourceSensorClusterFunction **SourceSensorClusterNum**
TargetSourceSensorClusterFunction TargetSourceSensorClusterNum Value

Table 5: Sensor Cluster Registry Table

The SensorClusterFunction and SensorClusterNum fields are used to identify the sensor cluster. The SensrClusterName is a concatenated character representation of the SensorClusterFunction and SensorClusterNum fields. The SensorClusterDesc field provides a brief description about the sensor cluster. The active field determines if the sensor cluster is active.

If a new sensor cluster is added to the WSN, the gateway can publish a WSDL like document to the SensorClusterUDDI repository. In addition, other gateway SCSMs can publish WSDL-like documents to the SensorClusterUDDI repository indicating that the sensor cluster exist and may be available for use. If a sensor cluster is removed, the SensorClusterUDDI is updated to reflect that the sensor cluster is no longer available for use. When the sensor clusters are to be considered for a CFG, if the sensor cluster is not active as indicated in the SensorClusterUDDI it will not be considered. This is a simple way to add and remove sensor clusters from the network.

6.5. The Energy Route Manager

The Energy Route Manager maintains the table of sensor clusters and their associated energy requirements to transmit data to their successor sensor clusters in the physical network. This energy route information is taken into account as a part of the selection criteria for the client's request.

The energy required table contains the energy required for each sensor cluster to communicate to their respective successor sensor cluster in the CFG (Table 6: Energy Required Table). The SourceSensorClusterFunction and SourceSensorClusterNum identify the source sensor cluster. The TargetSensorClusterFuntction and TargetSensorClusterNum identify the target sensor cluster. The Value field holds the value of the energy amount required for the source sensor cluster to communicate its response along the physical network route to the target sensor cluster.

Energy Required	
SourceSensorClusterFunction	**SHORT**
SourceSensorClusterNum	**SHORT**
TargetSourceSensorClusterFunction	SHORT
TargetSourceSensorClusterNum	SHORT
Value	LONG

Table 6: Energy Required Table

A partial listing of the energy required data is shown in for Figure 43: Client Request CFG1

Table 7: Partial List of Energy Required Table Values

Source Function	Source Sensor Cluster	Target Function	Target Sensor Cluster	Value
1	0	8	0	2
1	0	8	1	3
1	0	8	2	4
1	0	8	3	4
1	0	8	4	2
1	0	9	0	3
1	0	9	1	2
1	0	9	2	6
1	0	9	3	6
1	0	9	4	4
1	1	8	0	2
1	1	8	1	6
1	1	8	2	2

The data in Table 7: Partial List of Energy Required Table Values represents the energy requirements to transmit data from the sensor cluster in function F_1 to the sensor clusters in function F_8, from sensor cluster F_8 to the sensor clusters in F_9 and from function F9 to the sensor clusters in F_8. In Figure 44: Energy required from one sensor cluster to its successor we illustrate the energy required for sensor cluster S_{11} to the sensor clusters in F_8.

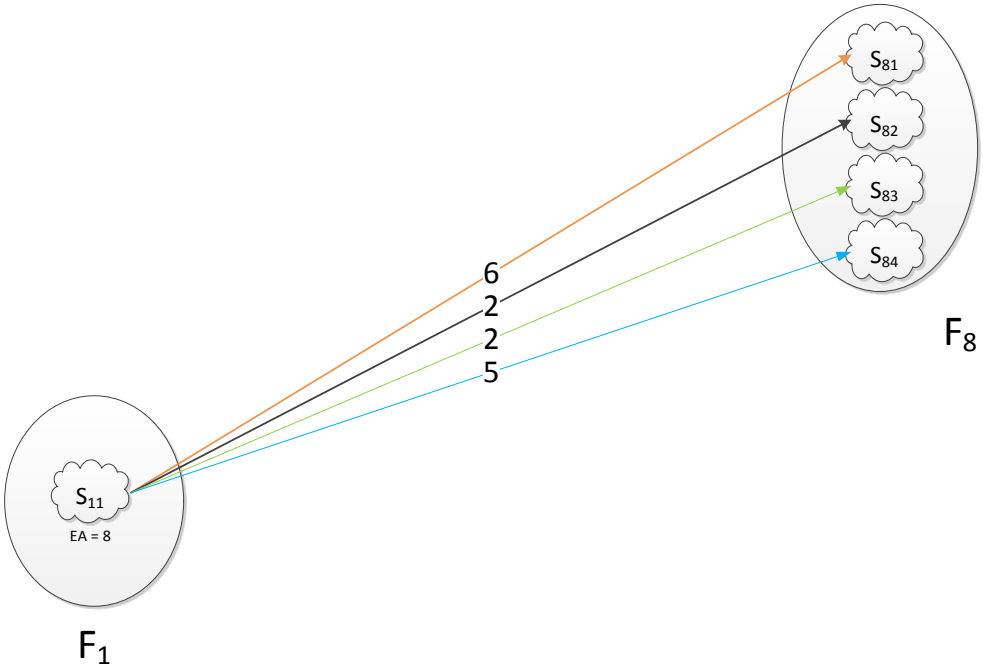


Figure 44: Energy required from one sensor cluster to its successor

6.6. Sensor Cluster QoS Manager

The Sensor Cluster QoS manager maintains QoS metrics for all sensor clusters in the network. It receives QoS information from the sensor cluster heads in the network and maintains this data in the QoS table based on the QoS categories specified in the QoSCat table.

The QoSCat table provides a listing of QoS types that are used for the WSN as illustrated in Table 8: QosCat Table.

QoSCat	
QosType	**SHORT**
Value	CHAR(10)

Table 8: QosCat Table

For example we have four QoSTypes that we are using in our WSN. They are Time, Reliability, Cost and EnergyAvail. These values are represented in Table 9: QoSCat Values:

QosCat	
QosType	Value
0	Time
1	Reliability
2	Cost
3	EnergyAvail

Table 9: QoSCat Values

The advantage of storing the QoS categories in a table is that it provides flexibility in the type of QoS categories that we can use in the WSN. For clarity, in the diagrams in this paper we will use the QosCat Value field in our tables going forward.

In our implementation we could have created a separate table that contains the list of functionally equivalent sensor clusters for each function. For simplicity, we combined the list of functionally equivalent clusters with the QoS values in the QoS table as shown in Table 10: QoS Table. The QoS table contains the current QoS metrics for all functionally equivalent sensor clusters for each function in the CFG. A sample of the data that is stored in this table is illustrated in Table 11: QoS Values.

QoS	
SensorClusterFunction	**SHORT**
SensorClusterNum	**SHORT**
QosType	**SHORT**
Value	LONG

Table 10: QoS Table

Quality metrics will be captured for each sensor cluster. The information that will be stored is as follows:

a) Time – The average time for the sensor cluster to perform its function

b) Reliability – the average that represents the reliable execution of the sensor cluster. This quality metric is derived by the total number of successful sensor cluster executions divided by the total number of sensor cluster executions for that sensor cluster.

c) Cost – represents the price in terms of cents, to execute the sensor cluster

d) EnergyAvail – this number represents the amount of collective energy that the sensor cluster has left. Since the sensor cluster is made up of many sensors, it is possible that one sensor has more energy than another. However, the energy available metric represents the energy available to execute the service that is required by all sensors. For example, let us say that a sensor cluster has 3 sensors within the cluster and one sensor has 6 units of energy while the other two sensors have 0 units each. To execute the service that the sensor cluster performs, 3 sensors are required with a minimum of 2 units of energy each. In this case, while the collective energy is 6 units, the sensor cluster will not execute the service. If the sensor does not have the requisite energy to execute, it will not. We will assume that the energy available metric represents the ability for the sensor cluster to perform the service which is inclusive of the requisite for each individual sensor. Each time the sensor cluster is executed and transmits information a certain amount of energy is depleted from

its energy source. When the sensor cluster no longer has enough collective energy left to execute its function, the sensor cluster will no longer be available for use in the network.

For example, in Table 11: QoS Values, we will review QoS data for the sequential path that consists of F_1, F_8, F_9 and F_8. The sensor clusters have the following metrics associated with them:

Table 11: QoS Values

Sensor Cluster Function	Sensor Cluster Num	Time	Reliability	Cost	Energy Avail
1	0	3	3	4	14
1	1	3	3	1	18
1	2	1	1	6	13
1	3	9	1	2	12
1	4	8	7	3	18
8	0	3	1	3	18
8	1	2	2	7	13
8	2	5	5	6	11
8	3	6	3	6	19
8	4	7	9	7	12
9	0	4	3	7	13
9	1	4	8	9	13
9	2	5	2	7	12
9	3	5	5	9	15
9	4	9	2	2	12

The sensor cluster head is responsible for capturing the QoS metrics of the sensor cluster and reporting the data to the gateway to update the QoS table.

The constraints table contains the client's specified constraints for the chosen CFG. A sample of the data for the table is shown in Table 13: Constraint Table Values.

Constraints	
QosType	**CHAR(2)**
Value	SHORT
CFG_ID	CHAR(10)

Table 12: Constraints Table

Constraints		
QoSType	Value	CFG_ID
Time	25	1
Reliability	25	1
Cost	35	1
Energy Avail	> Energy Required	1

Table 13: Constraint Table Values

The Energy Avail constraint indicates that the sensor cluster must have enough energy to execute its function, i.e. the energy available in the sensor cluster must be greater than the energy required to perform the task. It is a default criteria that the end user does not need to specify.

6.7. **The Processing Center**

The Processing center executes the CSI developed from the CFG. All statistics from the execution of the composite service will be captured and recorded in the RDBMS.

6.8. **The Sensor Cluster Selection Manager**

The Sensor Cluster Selection Manager (SCSM) is one of the core applications of the gateway. The SCSM manages QoS data for sensor clusters in the network and will receive the composite function graph requirements and the QoS requirement for the client's request. The SCSM selects sensor clusters based on clients request and QoS constraints. Once the sensor clusters are selected,

the selection information is sent to the Route Selection manager. We review the features of the SCSM in detail in the next chapter.

6.9. **Conclusion**

In this chapter we provide a high level overview of the functions and tables of the Sensor Cluster Selection Manager. In the next section, as a part of the SCSM, we will cover the details of the sensor cluster selection algorithm and complexity.

Chapter 7

Sensor Cluster Selection Algorithm and Complexity

The SCSM performs the sensor cluster selection for the requested CFG's along with the client's specified constraints. We have provided the overview of the selection problem in Chapter 5 where we mapped the problem to MMKP. In this section we will review our implementation of MMKP for sensor cluster selection.

7.1. Implementing the Sensor Cluster Selection Algorithm for Multiple Constraints

In reviewing approaches to solve MMKP, there are two categories:

a) Exact algorithms to find optimal solutions

b) Approximation algorithms to find near optimal solutions

Finding exact solutions to MMKP is NP-Complete and hence is not practical for large scale problems. We use dynamic programming to find exact solutions to small scale sensor cluster selection problems recognizing that the worst case time complexity will increase exponentially.

Dynamic programming algorithms assess all potential paths to a problem to determine the best possible solution. The dynamic programming approach represents an intelligent, brute-force method for finding the optimal solution to a problem. If the size of the problem makes it feasible to review all possible solutions in a reasonable time, dynamic programming assures that the optimal solution will be obtained.

The general approach to dynamic programming is to find initial feasible solutions to a small part of a problem and then to iteratively enlarge the problem to find new feasible solutions until the whole problem is solved. The dynamic programming approach seeks to solve each sub-problem only once, and to obtain the best feasible solutions. The solutions that are feasible, but are inferior to other solutions are eliminated thus reducing the number of computations. This approach is especially useful when the number of feasible solutions grows exponentially as a function of the size of the input.

In our example of DP we divide the problem into multiple parts and find initial feasible solutions to a small part of the problem. We then iteratively enlarge the problem and use the solutions from the smaller problems until the whole problem is solved. We can then trace back to find the optimal solution of all feasible solutions.

To select the sensors clusters that satisfy the multiple QoS constraints while maximizing the network efficiency, we decompose a composite sensor cluster graph into several inter-related sub-graphs. This results in the decomposition of the larger problem into several inter-related sub-problems.

We first generate feasible solutions for one sub-graph. We then incorporate these values in the next sub-graph to find feasible values for both sub-graphs combined. We store this information in a table and continue this evaluation until we finally include all sub-graphs in the composition and obtain those feasible solutions. We can then backtrack through our table to find the optimal solution of all feasible solutions for the composite sensor cluster graph.

Following is a general overview of how the program works. In the main program, we first load the data into the following tables: CFG, Constraints, QoS and Energy Required tables.

FeasibleSolutions	
Path	**CHAR(10)**
FeasSolnInstance	**CHAR(10)**
FeasSolnNum	**DECIMAL(10,2)**
SensorClusterCombo	CHAR(200)
TimeSum	LONG
ReliaSum	LONG
CostSum	LONG
EnergyReqSum	LONG
Addl_Qos	LONG
EnergyEff	SINGLE
CFG_ID	CHAR(10)
PrevPath	CHAR(10)
PrevFeasibleSolutionInstance	CHAR(10)
PrevFeasSolnNum	LONG
Discard	BIT

Table 14: FeasibleSolutions Table

The table that will hold all of the feasible solutions, Table 14: FeasibleSolutions Table, is shown above. For each function in each path, a new set of feasible solutions will be generated, taking into account the values that were computed before that function.

7.1.1. Dealing with Workflow in DP

If we encounter a workflow operation like an XOR-Split, Or-Split or, an AND-Split then we create a new Path in the FeasibleSolution table. In this case F_1, F_8, F_9 and F_8 is our first path, that will be followed by a new path created by F_1, F_6, F_7 and F_8 and so on.

We enlarge our problem by combing our current solutions with sensor cluster candidates from the next function. We then remove all non-feasible solutions (bad choices) and less optimal solutions (poor choices). We review the outcomes of the feasible solutions for this section of the CFG and label them as "bad" and "poor" choices which are then marked to be eliminated from our feasible solution space. In our example we will mark $S_{11}S_{81}$ as a bad choice if any of the following is true:

- If the sum of execution time for S_{11} and S_{81} is greater than the constraint specified for time, or
- If the product of reliability for S_{11} and S_{81} is less than the constraint specified for reliability, or
- If the sum of cost for S_{11} and S_{81} is greater than the constraint specified for cost, or
- If the energy required for S_{11} or S_{81} is greater than the energy available for either S_{11} or S_{81}

We label $S_{11}S_{81}$ as a poor choice if the sum of execution time, the product for reliability and the sum of cost for S_{11} and S_{81} is worse than any another solution (S_{10} and S_{80}) or (S_{10} and S_{81}) or (S_{10} and S_{82}) and so on, in the feasible solution table.

We continue the comparisons for the other feasible solutions in our table to identify any additional poor choice solutions. We record the new set of feasible solutions in our table (see Table 15: Feasible Solutions for the first two functions in CFG), and we continue to enlarge the problem by combining our existing feasible solutions with new functions in the path.

Path	Instance	Number	Combo	Time	Relia	Cost	Energy Req
1	0	0	F1S0 F8S0	6	3	2	2
1	0	1	F1S0 F8S1	5	6	3	3
1	0	2	F1S0 F8S2	8	15	4	4
1	0	3	F1S0 F8S3	9	9	4	4
1	0	4	F1S0 F8S4	10	27	2	2
1	0	5	F1S1 F8S0	6	3	2	2
1	0	6	F1S1 F8S1	5	6	6	6
1	0	7	F1S1 F8S2	8	15	2	2
1	0	8	F1S1 F8S3	9	9	5	5
1	0	9	F1S1 F8S4	10	27	5	5
1	0	10	F1S2 F8S0	4	1	4	4
1	0	11	F1S2 F8S1	3	2	6	6
1	0	12	F1S2 F8S2	6	5	3	3
1	0	13	F1S2 F8S3	7	3	2	2
1	0	14	F1S2 F8S4	8	9	6	6
1	0	15	F1S3 F8S0	12	1	5	5
1	0	16	F1S3 F8S1	11	2	3	3
1	0	17	F1S3 F8S2	14	5	3	3
1	0	18	F1S3 F8S3	15	3	6	6
1	0	19	F1S3 F8S4	16	9	5	5
1	0	20	F1S4 F8S0	11	7	6	6
1	0	21	F1S4 F8S1	10	14	3	3
1	0	22	F1S4 F8S2	13	35	6	6
1	0	23	F1S4 F8S3	14	21	3	3
1	0	24	F1S4 F8S4	15	63	5	5

Table 15: Feasible Solutions for the first two functions in CFG 1

A new feasible instance is created for function F_9 and we combine our previous solutions with the sensor clusters of function F_9. We continue this process until all functions and sensor clusters in the path have been examined.

7.1.2. Final Solution Table & Selecting the Best Answer

After we have examined all functions and their associated sensor clusters we review our feasible solutions table to determine if we have found a solution. It is possible that no solutions can be found that satisfy the given constraint. In addition, it is also possible that more than one solution can be found and in that case, we want to select the best of all feasible solutions that will optimize our energy utilization. As an example, for the final FeasibleSolutions table for this path we have 16 feasible solutions.

Path	Instance	Number	Combo	Time	Relia	Cost	Energy Req
1	2	0	F1S0 F8S1 F9S0 F8S0	12	18	11	8
1	2	1	F1S0 F8S1 F9S0 F8S1	11	36	9	13
1	2	2	F1S0 F8S1 F9S0 F8S3	15	54	12	6
1	2	3	F1S2 F8S0 F9S0 F8S0	11	3	11	9
1	2	4	F1S2 F8S0 F9S0 F8S1	10	6	9	14
1	2	5	F1S2 F8S0 F9S0 F8S3	14	9	12	7
1	2	6	F1S2 F8S0 F9S1 F8S0	11	8	10	10
1	2	7	F1S2 F8S0 F9S1 F8S1	10	16	8	15
1	2	8	F1S2 F8S0 F9S2 F8S0	12	2	11	10
1	2	9	F1S2 F8S0 F9S2 F8S1	11	4	9	15
1	2	10	F1S2 F8S0 F9S2 F8S3	15	6	12	8
1	2	11	F1S2 F8S1 F9S0 F8S0	10	6	14	11
1	2	12	F1S2 F8S1 F9S0 F8S1	9	12	12	16
1	2	13	F1S2 F8S3 F9S3 F8S0	15	15	8	7
1	2	14	F1S2 F8S3 F9S3 F8S1	14	30	6	12
1	2	15	F1S2 F8S3 F9S3 F8S3	18	45	9	5

Table 16: Final feasible solution table

We now review these 16 solutions to find the best solution based on the amount of energy required in relationship to the amount of energy available for each sensor cluster. That is for each feasible solution we compute the energy efficiency using the following formula in EQ[45] for each sensor cluster in that solution:

$$\frac{EnergyAvail - EnergyReq}{EnergyAvail}$$ EQ [45]

We then take the sum of those values to obtain an energy efficiency value for each feasible solution in the final set. The feasible solution with the greatest value is the best answer and will be returned as the final result. That is, the answer that uses the least amount of energy and has the best energy available to energy required ratio.

The optimal answer to the solution for path 1 is F1S2 F8S3 F9S3 F8S0 and that value would be returned to the processing center to execute the CSI.

The general flow for the sensor cluster selection process in the main program that finds the best solution for the CFG is illustrated in Figure 45: General Flow for Selecting Sensor Clusters for a CFG

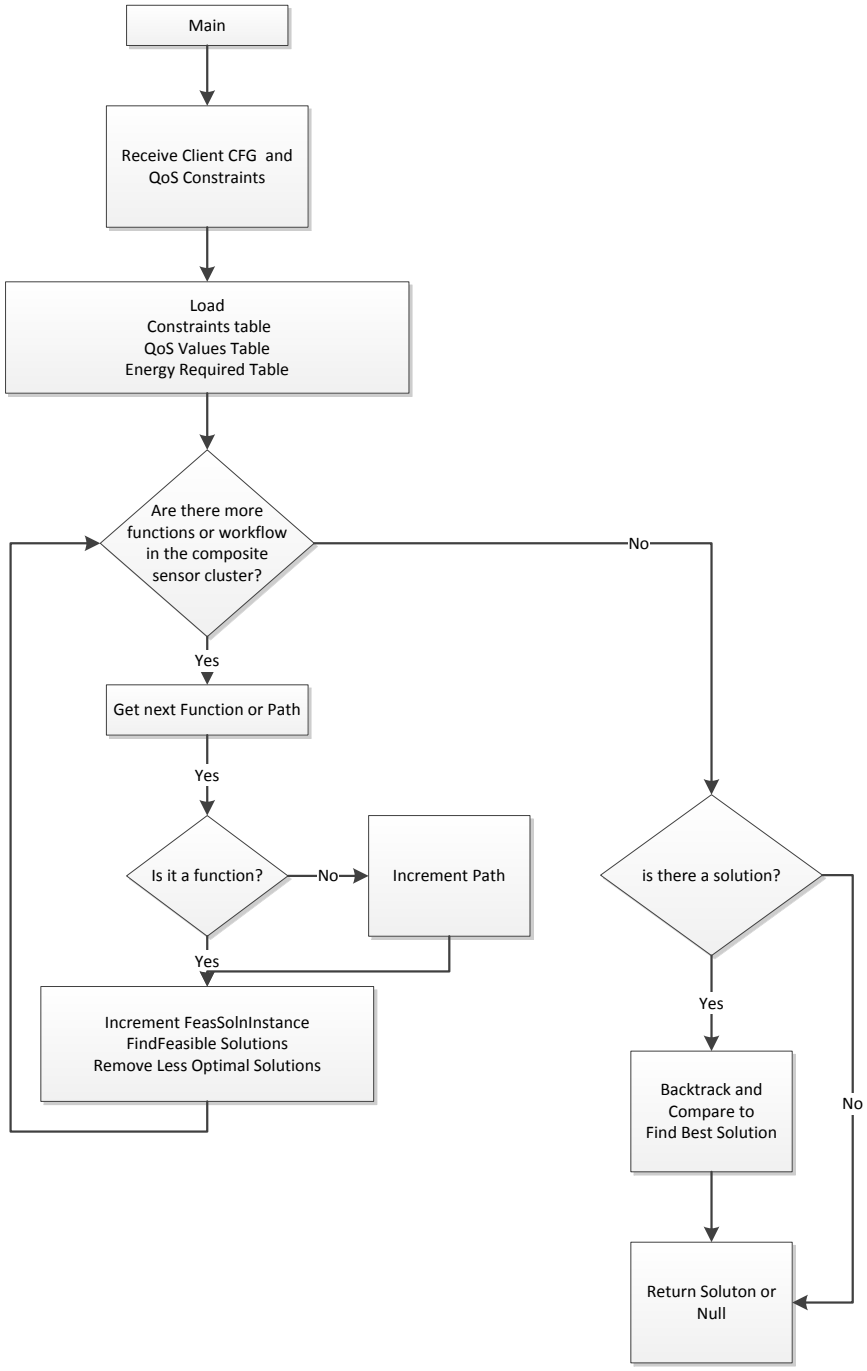


Figure 45: General Flow for Selecting Sensor Clusters for a CFG

7.2. **Analysis of the MMKP Algorithm**

Let us review the case where there are no eliminations of feasible choices and we call it the brute force method. In this case the complete solution space is defined and the selection of the optimal choice is performed at the end.

For Sequential Paths

i – index for the function

w_i- is the number of sensor clusters for function i

N is the number of functions in the sequential path

The number of solutions in a sequential path can be represented by the algorithm

$$G = \prod_{i=1}^{N} w_i \qquad \text{EQ [1]}$$

The complexity of this formula is $O(w_1 * w_2 * \dots * w_N)$

Let P represent the number of solutions for paths in an XOR or AND operation in a CFG so, $p_z = (w_{z_1} * w_{z_2} * \dots * w_{N_z})\ where\ z = (1,2,\dots,m)$ and $P = (p_1 + p_2 +, \dots, p_m)$. Then the formula can be represented as:

$$P = \sum_{z=1}^{m} \prod_{i=1}^{N_z} w_{z_i} \qquad \text{EQ [2]}$$

And the complexity for this formula can be represented by

$$O\left(\left(w_{1_1} * w_{1_2} * \ldots * w_{1N_1}\right) + \left(w_{2_1} * w_{2_2} * \ldots * w_{2N_2}\right) + \ldots + \left(w_{m_1} * w_{m_2} * \ldots * w_{mN_m}\right)\right)$$

We have shown how to determine the complexity for sequential paths and for parallel splits in a CFG. If we view a CFG as a graph containing sequential and parallel splits, a pattern of workflow emerges.

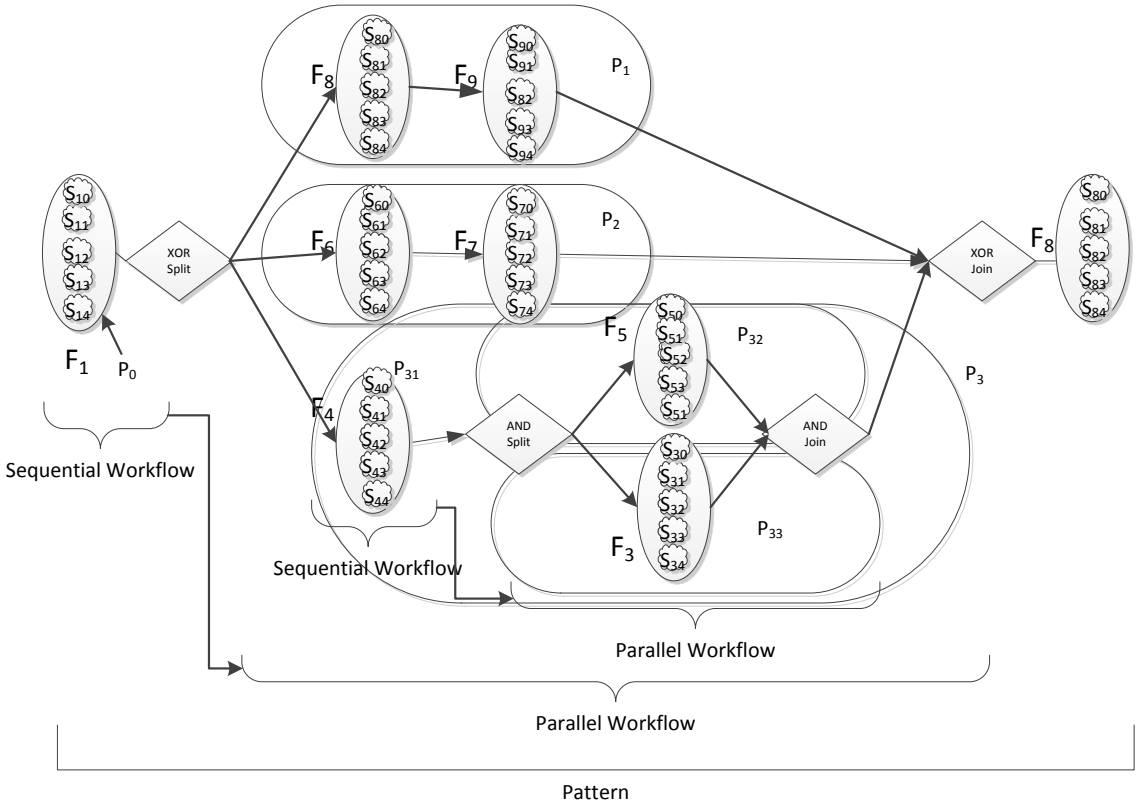


Figure 46: Sequential-Parallel Workflow Patterns

For each sequential path followed by a parallel split, we must multiply the number of solutions in the sequential path by the number of solutions for each path in the parallel split. For example, in Figure 46: Sequential-Parallel Workflow Patterns, the first sequential path is represented by p_0. The sequential paths of the OR-Split directly following p_0 are p_1, p_2, p_3. We must multiply the

solutions in p_0 by the solutions in p_1, p_2 and p_3 resulting in the formula $p_0(p_1 + p_2 + p_3)$. In fact, parallel split p_3 consist of a sequential-parallel workflow pattern ($p_{31} + p_{32} + p_{33}$) and this pattern must be evaluated to obtain a result for p_3. Hence the formula becomes $p_0(p_1 + p_2 + (p_{31}(p_{32} + p_{33}))$. In general the *Possible Number of Solutions* is a factor of the number of sequential-parallel workflow patterns. As the number of sequential-parallel workflow patterns increases the *Possible Number of Solutions* increases exponentially.

Once we have completely enumerated the *Possible Number of solutions*, in order to find the optimal solution we must compare each solution to other members of the solution space. The comparisons also grow based on the number of QoS dimensions.

We will use the following scenario to illustrate the growth for the brute force method:

Let the number of functions in the CFG = 10, the number of QoS categories =5, and number of functionally equivalent sensor clusters for each function in the CFG = 5. For the brute force method we have the following growth in the number of feasible solutions:

Number of Functions in the CFG	Enumeration of Possible Solutions
0	5
1	25
2	125
3	625
4	3125
5	15625
6	78125
7	390625
8	1953125
9	9765625

7.3. Analysis of the Dynamic MMKP Algorithm - Eliminating poor choices at each function

Dynamic programming is used to reduce the number of possible feasible solutions in the solution space. We do so by eliminating "bad" or "poor" choices at each function evaluation.

Since a better choice exists, the poor choice will not result in an optimal solution, hence this choice can be eliminated. If we eliminate bad and poor choices early on in our selection process, we are able to reduce the number of solutions in our possible solution space significantly.

In our test, we simplify the problem by using a small sample of functions per CFG, a constant number of sensor clusters per function and no parallel paths. We capture results of the program for iterations of 100, 1000, 5000 and 10000 times and compare it to the brute force method in Table 17: Final Set of Solutions.

FUNCTION	100	1000	5000	10000	Avg # of Feasible Solutions	Brute Force # of Feasible Solutions
0	5	5	5	5	5	5
1	25	25	25	25	25	25
2	125	125	125	125	125	125
3	185	193	196	192	192	625
4	468	475	492	470	476	3125
5	1078	1037	1078	1025	1054	15625
6	2037	1847	1925	1829	1909	78125
7	2283	2190	2263	2163	2225	390625
8	1500	1390	1481	1421	1448	1953125
9	388	380	420	416	401	9765625

Table 17: Final Set of Solutions

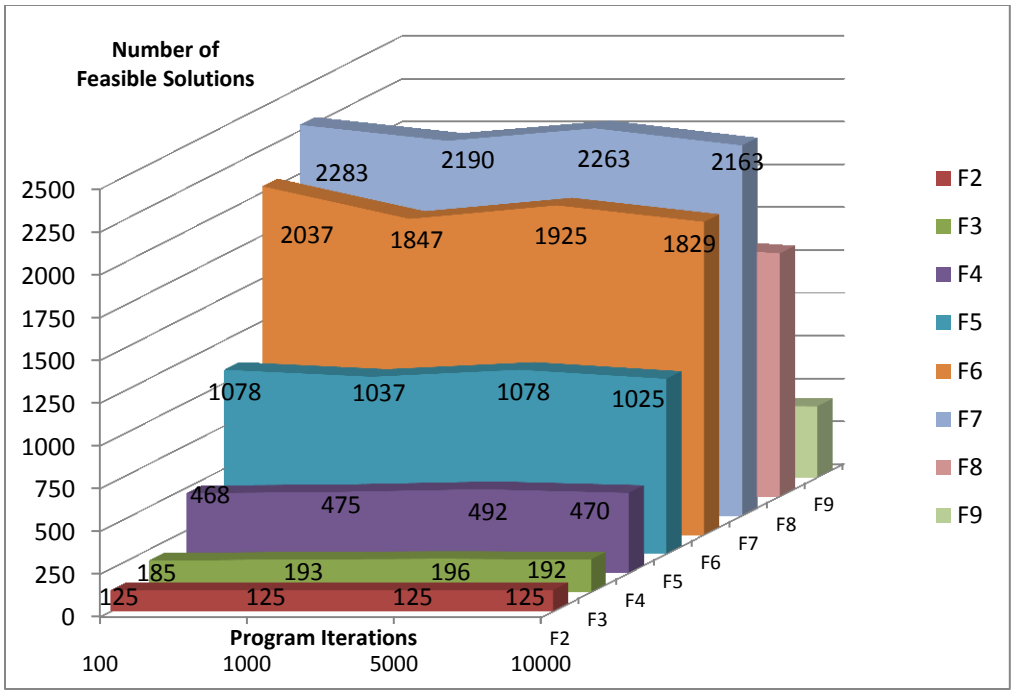


Figure 47: MMKP Average Execution Steps

Our results show that there was a significant reduction in the number of operations required to find the optimal solution when compared to a brute force exhaustive search.

The rate of growth of the Brute Force Method is exponential for each function compared to linear-like growth in the dynamic programming method.

7.4. Analysis of energy consumed using arbitrary selection versus MMKP selection

The objective is to select an optimal feasible solution that maximizes the energy efficiency. At the end of the analysis, we were presented with 16 solutions to the selection problem. Of the 16 solutions, we selected the one with the best energy available to energy required solution. If we continue to select sensor clusters on this basis for our CFG's not only do we select resources that

meet the client's quality constraints, we also select sensor clusters that utilize energy in an efficient and balanced way in the network. This balanced utilization of energy has already been proven to be effective by [16].

7.5. **Conclusion**

We developed a dynamic program to implement the sensor cluster selection algorithm for small scale problems which can be mapped to MMKP. MMKP is NP-Complete. Our program improved the solution space dramatically and reduced the number of solutions significantly.

In addition, we provided a formula to maximize the energy efficiency in the WSN by selecting the most energy efficient sensor cluster combination for a CFG.

Chapter 8

Conclusion and Future Direction

QoS and network life time management are key challenges in WSNs that can be improved by applying SOA methodologies to a WSN. What our contributions to the research are as follows:

1) We identify the primary objectives of quality for a SOA WSN in order to define the standard for operation. The primary objectives are:
    a) Loose coupling of application and sensors to promote ease of application modification and re-usability
    b) Improve reliability and availability of the network
    c) Improve network lifetime
2) We define the conceptual model of a service in a WSN using Sensor Clusters. Sensor clusters provide the service in the WSN
3) We apply the concepts of service orchestration through Composite Services, Composite Function Plans and Logical & Physical graphs to a WSN.
4) We provide formulas for Calculating QoS with Workflow Operations using QoS aggregation methods.
5) We apply dynamic Sensor Cluster selection to a WSN to select sensor clusters that will achieve our objectives of quality.

6) We developed a dynamic program that implements MMKP for Sensor Cluster selection as a proof of concept to determine if the objectives were achieved.

This research differs from other research in the following areas:

1) We use a unique method of applying SOA to WSN using dynamic Sensor Cluster composition
2) We have shown that the application of Dynamic Sensor Cluster composition can provide the benefits that we are looking for in a WSN by:
    a) Increasing the energy efficiency of the network to potentially extend network lifetime
    b) Improving the usage of sensors by selecting the best sensors that meet specific constraints
    c) Improving reliability and availability on the network through sensor cluster selection that meets specific reliability and availability constraints
    d) Increasing the manageability of the network through the use of a centralized repository that keeps track of active and inactive sensors
    e) Creating new applications in the WSN quickly through the use of dynamic sensor cluster selection

Our future directions in this research are to:

1) further define the quality objectives in a WSN, identify methods to achieve them and the quality metrics to measure them.
2) explore how to handle sensor mobility and network reconfiguration in a WSN with dynamic sensor cluster selection

3) examine how reconfiguration with the sensor cluster (head and sensors) can provide benefit to sensor cluster lifetime

4) potentially, expand the research to other areas that would include Radio Frequency Identification (RFID) and other technologies. A relatively new paradigm called the "Internet of Things" [57] promotes the concept of a pervasive presence around us of a variety of "things" such as RFID tags, sensors, actuators and mobile phones all working in concert over the internet to provide some functionality. Expanding our research to explore the protocols and standardization requirements for interoperability across many technologies which include WSN is a goal for our future direction.

There is a great deal of opportunity for research in the area of interoperability and WSNs. The basic concepts of achieving loose coupling and re-usability will improve the usage and lifetime of WSNs. In addition, standardization across many technologies will help to bring paradigms like the internet of things to reality. This research is a step in that direction. Future research will expand on these ideas.

Bibliography

[1] F. Delicato, F. Protti, J. Ferreira de Rezende, L. Rust and L. Pirmez, "Appliction-Driven Node Managemet in Multihop Wireless Sensor Networks," Springer, vol. 3420, pp. 569-576, 2006.

[2] A. Meads, "A Holistic Approach to Mobile Service Provisioning," Auckland, New Zealand, 2009.

[3] L. Liu, "Delivering Sustainable Capability on Evolutionary Service-oriented Architecture.," in IEEE International Symposium on Object/Component/Service Oriented Real-Time Distibuted Computing. , 2009.

[4] J. Bih, "Service Oriented Architecture (SOA) A New Paradigm to Implement Dynamic E Business Solutions," in Ubiquity (An ACM IT Magazine and Forum), 2006.

[5] E. Ort, "Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools," April 2005. [Online]. Available: http://java.sun.com/developer/technicalArticles/WebServices/soa2/. [Accessed 28 September 2011].

[6] W. Emmerich, M. Aoyama and J. Sventek, "The Impact of Research on Middleware Technology," ACM SIGSOFT Engineering Notes, vol. 32, no. 1, pp. 21-46, January 2007.

[7] Webservices.org, "UDDI Members," Webservices.org, 14 April 2003. [Online]. Available: Finding Web services with UDDI - Registry-UDDI - Technology - Categories - WebServices_Org.htm. [Accessed 13 August 2007].

[8] O. Ezenwoye and M. Sadjadi, "Composing Aggregagte Web Services in BPEL," in ACM Southeast Regional Conference, Proceedings of the 44th Annual Southeast Regional Conference, New York, 2006.

[9] J. M. Prinsloo, C. L. Schulz, D. G. Kourie, W. M. Theunissen, T. Strauss, R. V. D. Heever and S. Grobbelaar, "A Service Oriented Architecture for Wireless Sensor and Actor Network Applicaitons," in Proceeding of SAICSIT 2006, Pretoria, South Africa, 2006.

[10] Ali, Z.; Shahzad, W., "Analysis of Routing Protocols in AD HOC and Sensor Wireless Networks Based on Swarm Intelligence," International Journal of Networks and Communications, pp. 1-11, 2013.

[11] M. Yu, H. Mokhtar and M. Merabti, "A Survey of Network Mangement Architecture in Wireless Sensor Network," in In in Proc. of the Sixth Annual PostGraduate Symposium on The Convergence of Telecommunications, Netowrking & Broadcasting (2006), Liverpool, 2006.

[12] W. W. W3C Group, "QoS for Web Services: Requirements and Possible Approaches," 23 November 2003. [Online]. Available: http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/#2. [Accessed 15 November 2010].

[13] L. Taher, R. Basha and H. Khatib, "QoS Information and Computation (QoS-IC) Framework for QoS Based Discovery of Web Services," Upgrade, vol. VI, no. 4, pp. 55-66, August 2005.

[14] D. Chen and P. Varshney, "QoS Support in Wireles Sensor Networks - A Survey," in In MSWiM '07: Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems, 2007.

[15] H. Wu, Q. Luo, J. Li and A. Labrinidis, "Quality Aware Query Scheduling in Wireless Sensor Networks," Proceeding of the Sixth International Workshop on Data Management for Sensor Networks, 2009.

[16] V. Vanitha and V. Palanisamy, "A Global QoS Aware Service Composition in Wireless Sensor Networks," vol. 1, no. 19, 2010.

[17] F. Delicato, F. Protti, J. Ferreira de Rezende, L. Rust and L. Pirmez, "Application-Driven Node Managemet in Multihop Wireless Sensor Networks," Springer, vol. 3420, pp. 569-576, 2006.

[18] K. Pandey and S. Patel, "A Novel Design of Service Oriented and Message Driven Middleware for Ambient Aware Wireless Sensor Network," International Journal of Recent Trends in Engineering, Vols. 1, No. 1, pp. 313 - 317, May 2009.

[19] M. Marin-Perianu, N. Meratnia, P. Havinga, L. de Souza, J. Muller, P. Spiess, S. Haller, T. Riedel, C. Decker and G. Stromberg, "Decentralized Enterprise Systems: A Multi-Platform Wireless Sensor Networks Appraoch," in Wireless Communications, IEEE , 2007.

[20] J. King, R. Bose, H.-I. Yang, S. Pickles and A. Helal, "Atlas: A Service-Oriented Sensor Platform," in 29th Annual IEEE International Conference on Local Computer Networks (LCN'04), Gainesville, FL, 2006.

[21] B. Wu, C. Chi and S. Xu, "Service Selectoin Based on QoS Reference Vector," IEEE Congress on Services, pp. 270-277, 2007.

[22] J. Kulik, R. B. Heinzelman and H. and Balakrishnan, "Negotiation-based protocols for disseminating information in Wireless Sensor Networks," in ACM Wireless Networks 2000, Available in: http://citeseer.nj.nec.com/ 335631.html, 2000, 2000.

[23] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000), Boston, MA, 2000.

[24] W3C (World Wide Web Consortium) Note, WAP Binary XML Content Format.

[25] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura and E. Jansen, "The gator tech smart house: A programmable pervasive space," Computer, pp. 50-60, 2005.

[26] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in In PLDI '03: Proceedings of the ACM SIGPLAN 2003 conference on Programming language, New York, NY, 2003.

[27] P. Levis, N. Lee, M. Welsh and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos application," in SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems., New York, NY, 2003.

[28] L. O'Brien, L. Base and P. Merson, "Quality Service Attributes and Service Oriented Architectures," Carnegie Mellon University, Pittsburg, 2005.

[29] M. P. Papazolglou and W. Jan van den Heuvel, "Service Oriented Design and Development Methodology," International Journal of Web Engineering and Technology (IJWET), pp. 1-17, 2006.

[30] O. Younis, M. Krunz and S. Ramasubramanian, "Node Clustering in Wireless Sensor Networks: Recent Developments and Deployment Challenges," IEEE Networks, pp. 20-25, 2006.

[31] Z. Zhou, S. Das and H. Gupta., "Connected K-Coverage Problem in Sensor Networks," in Computer Communications and Networks, 2004.

[32] C. Ortega, T. Brown, J. Ibbotson and R. Hancock, "Improving WSN Application QoS and Network Lifetime Management using SOA Strategies," in Military Communications Conference, 2011 MILCOMM, Baltimore, MD, 2011.

[33] M. C. Jaeger, G. Muhl and S. Golze, "QoS-aware Composition of Web Services: A Look at Selection Algorithms," IEEE Conference on Web Services, pp. 1-2, 2005.

[34] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski and A. Barros, "Workflow Patterns," Distributed and Parallel Databases, pp. 5-51, 2003.

[35] M. Fluegge, I. Garcia dos Santos, N. Tizzo and E. Madeira, "Challenges and Techniques on the Road to Dynamically Compose Web Services," in Proceedings of the 6th International Conference on Web Engineering, Palo Alto, CA, 2006.

[36] M. Oh, J. Baik, S. Kang and H. Choi, "An Efficient Approach for QoS Aware Service Selection Based on a Tree-Based Algorithm," in Seventh IEEE/ACIS International Conference on Computer and Information Science, 2008.

[37] S. Geyik, B. Szymanski, P. Zerfos, D. Verma, J. Wright and C. Vincent, "Dynamic Composition of Services in Sensor Networks and Its Implementation under Sensor Fabric," in Annual Conference of ITA, 2010.

[38] E. S. S. Horowitz, "Computing partitions wiht applications to the Knapsack Problem," Journal of the ACM, pp. 21, 277-292, 1974.

[39] S. Martello and P. Toth, Knapsack Problems: Algorithms and Computer Implementations, Chichester, England: Wiley, 1990.

[40] R. Nauss, "An Efficient Algorithm forthe 0-1 Knapsack Problem," Management Science, pp. 23, 27-31, 1976.

[41] D. Pisinger, "Algorithms for Knapsack Problems," Copenhagen, 1995.

[42] S. Martello and P. Toth, "Branch and Bound Algorithms for the Solution of General Uni-Dimensional Knapsack Problems," in Advances in Operations Research, North Holland, 1977, pp. 295-301.

[43] O. Ibarra and E. Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," Journal of the ACM (JACM),, pp. 463-468, 1975.

[44] D. Pisinger, "A minimal algorithm for the Multiple-Choice Knapsack Problem," European Journal of Operational Research , pp. 394-410, 1995.

[45] K. Klamroth and M. Wiecek, ""Dynamic programming approaches to the multiple criteria knapsack problem."," Naval Research Logistics, pp. 57-76., 2000.

[46] M. Jaeger and G. Rojec-Goldmann, "SENECA - Simulation Algorithms for the Selection of Web Services for Compositions," TU Berlin, Institute of Telecommunication Systems, Berlin, 2005.

[47] A. Sbihi, "A best first search exact algorithm for the Multiple-choice Multidimensional Knapsack Problem," Journal of Combinatorial Optimization, pp. 337-351, 8 December 2007.

[48] M. Moser, D. Jokanovic and N. Shiratori, "An Algorithm for the Multidimensional Multiple-Choice Knapsack Problem," IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer, pp. 582-589, 1997.

[49] S. Khan, K. Lleric, G. Manning and M. Akbar, "Solving the Knapsack Problem for Adaptive Multimedia Systems," Stud. Infom., pp. 157-178, 2002.

[50] G. Yeom, T. Yun and D. Min, "A QoS Model and Testing Mechanism for Quality Driven Web Services Selections," in Proceedings of the Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems and Second International Workshop on Collaborative Computing, Integration and Assurance, 2006.

[51] Y. Yang, S. Tang, Y. Xu, W. Zhang and L. Fang, "An Approach to QoS-aware Service Selection in Dynamic Web Service Composition," Third International Conference on Networking and Services(ICNS'07), 2007.

[52] G. Confora, M. Di Penta, R. Esposito and M. L. Villani, "QoS Aware Replanning of Composite Web Services," ICWS, 2005.

[53] V. Cardellin, E. Casaliccho, V. Grassi and F. Lo Presit, "Flow-Based Service Selection for Web Service Composition Supporting Multiple Qos Classes," ICWS 2007. IEEE International Conference on Web Services, pp. 743-750, 2007.

[54] J. Hu, C. Guo, H. Wang and P. Zou, "Quality Driven Web Services," in IEEE International Conference on e-Business Engineering ICEBE, 2005.

[55] M. Khan, "Quality adaptation in a multisession multimedia system: Model, algorithms and architecture, Doctoral Dissertation," University of Victoria,, 1998.

[56] T. Yu., "Quality of Services in Web Services: Model, Architecture and Algorithms - Dissertation," University of California, California, 2006.

[57] L. Atzori, A. Iera, G. Morabito, "The Internet of Things: A Survey," Calabria, Italy, 2010.

[58] J. L. Hill, "System architecture for wireless sensor networks," University of California, Berkeley, Berkeley, 2003.

[59] C. Perkins, "“Service Location Protocol White Paper".

[60] M. Marin-Perianu, N. H. P. Meratnia, L. M. S. de Souza, J. Muller, P. Spieß and G. Stromberg, "Decentralized enterprise systems: a multiplatform wireless sensor network approach.," Wireless Communications, IEEE,, pp. 57-66, 2007.

[61] Y. Tao, Y. Zhang and K. Lin, "Efficient algorithms for Web services selection with end-to-end QoS constraints.," ACM Transactions on the Web (TWEB), p. 6, 2007.

[62] Y. Wang and J. Vassileva, "A Review of Trust and Reputation for Web Service Selection," in 27th International Conference on Distributed Computing Systems Workshops, 2007.

[63] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell and K. Nahrstedt, "GAIA: a middleware platform for active spaces," SIGMOBILE Mobile Computing Communications, pp. 65-67, 2002.