MINI-SOA/ ESB DESIGN GUIDELINES AND SIMULATION FOR WIRELESS SENSOR NETWORKS

By

JONGYEOP KIM

Bachelor of Science in Computer Science

Korea National Open University

Seoul, Korea

1996

Submitted to the Faculty of the

Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of

MASTER OF SCIENCE
May, 2009

MINI-SOA/ ESB DESIGN GUIDELINES AND SIMULATION FOR WIRELESS SENSOR NETWORKS

Thesis Approved:

Dr. Johnson P. Thomas

Thesis Adviser

Dr. Nohpil Park

Dr. Xiaolin Li

Dr. A. Gordon Emslie

Dean of the Graduate College

ACKNOWLEDGMENTS

First of all, it has been an honor and privilege to have Johnson P. Thomas, who has provided me with the direction for the research, as my thesis advisor. His enthusiasm for research greatly inspires me. I am deeply grateful for his support and suggestions.

I gladly express my gratitude to Dr. Nohpil Park, graduate coordinator, Department of Computer Science, for providing all the valuable support to carry out the work. I would like to make a special thanks to Dr. Xiaolin Li for his valuable advice during the work.

I extend my sincere thanks to the Korean Government and the Military Manpower Administration for the overseas study support.

Finally, I am also thankful to my wife, Kyeongsook, whom I truly love and respect. I am thankful to my two boys, Seongsoo and Seonghyeon, who mean more than anything to me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Motivation and objective of research

Wireless Sensors Network(WSN) devices have been commercially used to gather information from the physical environment to transmit to the external cyber world application domains for the purpose of monitoring medical devices, process automation, transportation system automation, structural health monitoring, and many more applications[1,2,3,4].

One of the major challenges in WSN is the difficulty of efficient interaction between different WSN application domains because there is no open standard for supporting various types of sensors that are produced by many senor manufacturers [1,5,6]. To solve this problem, a Service Oriented Architecture (SOA) concept has been proposed to improve the interoperability between WSNs which are composed of heterogeneous sensor devices[6]. The SOA's three required factors for service interaction are Service Registry, Service Requestor and Service Provider. The Enterprise Service Bus is the core component of SOA. It supports message exchange between service providers and service consumers, which are loosely coupled.

This ESB needs to support Security, Message transformation, Reliability, Transaction management, Orchestration of Service, etc.[7]. There are many ESB products now widely available on the market, such as IBM WebSphere ESB, JAVA based OPEN ESB, BEA AquaLogic Service Bus, Mule, Cape Clear6, etc.

However, as those products focus on large enterprise services, it is difficult to apply WSN integration because of limited hardware and software capabilities. Therefore, this thesis discusses the important design issues of SOA/ESB, and proposes design guidelines for a mini-SOA Enterprise Service Bus (ESB) for WSN as an open standard for the management and application development of different WSN domains.

1.2 Research Contributions

In this thesis, we propose a mini-SOA/ESB for wireless sensor networks as an open standard.

The contribution of this thesis are as follows:

- The major design requirements, such as Transformation, Interoperability, Flexibility, Security, Quality-of-Service for a mini SOA/ESB.
- Mini-SOA/ESB Architecture to support WSNs.
- A sensor-UDDI structure to support Quality of Service.
- An simulation program is implemented with an Alternative Service List to increase service availability and keep service Consistency between sensor-UDDIs.

1.3 Organization of Thesis

Chapter 2 is a review of the literature This chapter will discuss:

1. Interconnection Issues and Integration issues in the wireless sensor networks.
2. Overview of Service Oriented architecture and Enterprise Service Bus
3. Commercial product review (IBM web Sphere) and OASiS frame work.

Chapter 3 proposes a mini-SOA/ESB for wireless sensor networks applications

Chapter 4 presents simulations of availability and consistency of mini-SOA/ESB.

CHAPTER II

REVIEW OF LITERATURE

2.1 Wireless Sensor networks

Advances in micro-electromechanical systems(MEMS) have enabled the development of small, inexpensive, low power, sophisticated sensors[8]. These sensors are connected with wireless communication technologies and are widely used for environmental monitoring, indoor climate control, habitat monitoring, transportation system automation, etc. Because of WSN's diversity and heterogeneity, previous research has focused on reducing energy consumption, wakeup strategies, time-synchronization, data aggregation, etc[8].This chapter discusses the characteristics of WSNs, and provides a brief review of interconnection issues and integration issues.

2.1.1 Sensor Node Architecture

A sensor node is composed of four major units: the processing unit, sensing unit, transceiver unit and power unit.

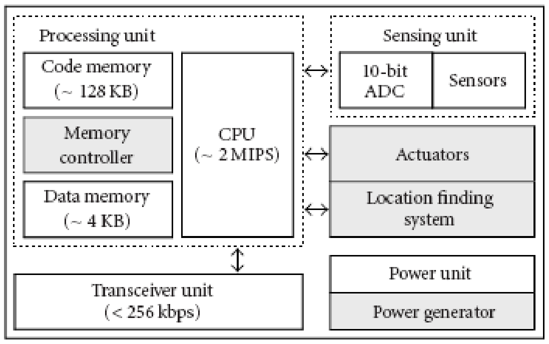


Figure 2.1 Mica2 Mote architecture [9]

A sensing unit senses analog data and converts it to digital data. A transceiver unit's role is to communicate with other nodes. A power unit supplies power to the node. An actuator perform location finding functions to a moving node.[9].

2.1.2 Interconnection Issues of WSNs

Research for sharing the sensor data over the Internet using Interconnecting WSNs methodologies are outlined below:

Interconnection Issues	Methodologies	Advantages	Disadvantages
Direct Interconnection Using IP protocol	Implement IP protocol stack on Sensor Node	Internet host can directly Send Command to particular nodes in Sensor Network	Sensor node is required enough processing capability
Overlay Indirectly Interconnection	Sensor networks protocol is deployed over the TCP/IP	Easy to integrate into a virtual sensor Network	Protocol overhead to TCP/IP network
Bridge for indirect interconnection	Different protocol in both networks are translated in application layer	The communication protocol used in the sensor networks may be chosen freely, and internet users cannot directly access any special sensor node	Single point of failure
Gateway for indirect Interconnection	A different protocol in both networks are translated by the application layer	The communication protocol used in the sensor network may be chosen freely	Internet users cannot directly access any special sensor node

Table 2.1 Advantages and disadvantages of Interconnection Issues[21].

2.1.3 Integration

For integration issues, three approaches have been proposed, namely, the Server-client Approach, Peer-to-peer Approach, and sensor network sharing.

2.1.4 Server-client Approach

This approach employs a central system which requires data owners to register their data sources with a central server. These sensing resources are updated at intervals to let the server know the availability When an application submits a query to search for a service, the central server analyzes the query and finds the appropriate sensor networks, and then produces a response [12].

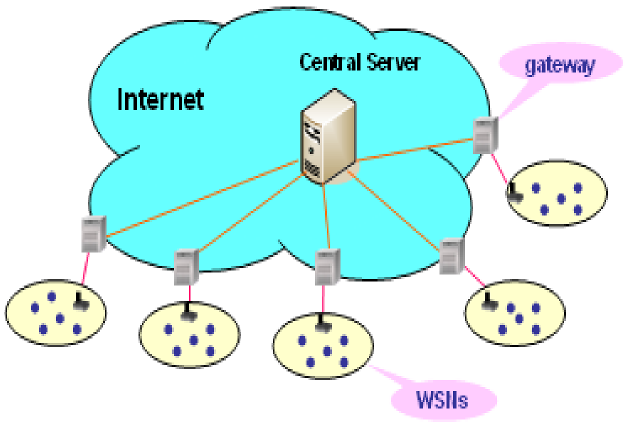


Figure 2.2 Server-Client Approach[12]

2.1.5 Peer-to-Peer Approach

Adopting P2P techniques, each WSN with a gateway acts as a peer. The main goal of P2P overlay is to treat the underling heterogeneous WSNs as a single unified network, in which users can send queries without considering the details of the network. [12]

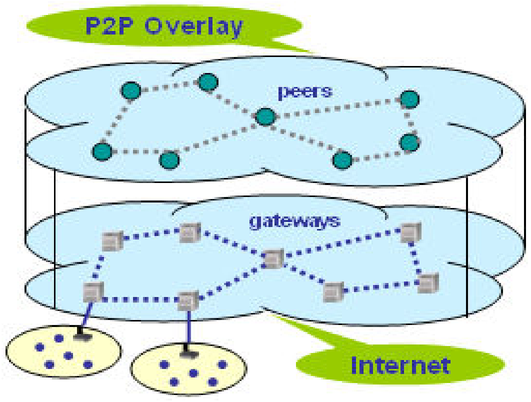


Figure 2.3 Peer-to-peer Approach[12]

2.1.6 SensorBase.org - Centralized repository to Slog

SensorBase.org was created for the purpose of sharing and managing a specific domain for sensor network data on Internet. It also serves as a search engine that provides users the ability to query for specific data sets based on geographic location, sensor type, range of time, and patterns in the sensor signals.[23]

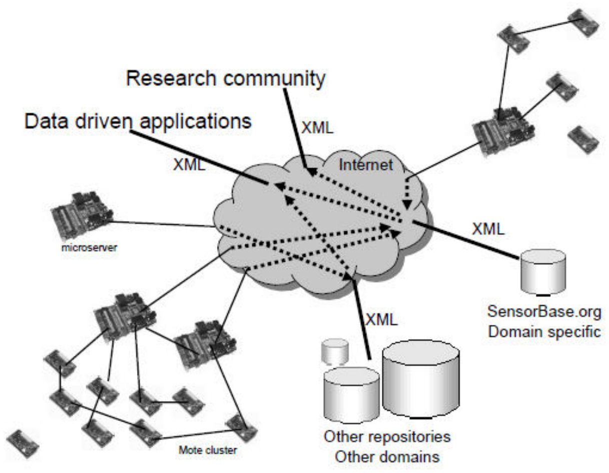


Figure 2.4 Sensor Network Data Sharing Overview[23]

2.1.7 World wide Sensor Web Framework Overview

The world wide Sensor Web is distributed over the Internet, and contains separate components which provide accessible services that are capable of networking sensing devices on a global scale. These components are composed as follows: the Query Handler, Sensor Register, Sensor Interface, Sensor Data Store, Functionality Register, and User Register[24].

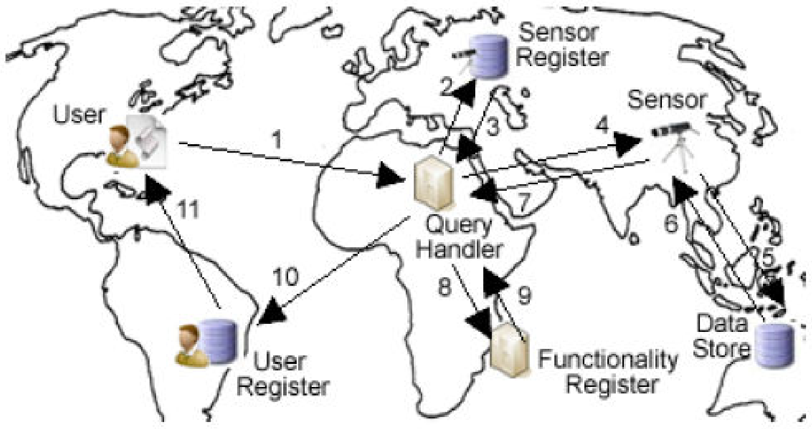


Figure 2.5 Relationship between components[24]

2.2 Service Oriented Architecture Overview

2.2.1 What are services ?

A Service includes every resource in a company or organization. It could be business logic, a data base system, file structure , documents, files , application processes, transactions, and anything that can be accessed via a network [14].

2.2.2 SOA definition

An SOA "provides methods for system development and integration where systems group functionality around business process, and packages these as interoperable services[7]."
There are three major essential elements for SOA. These are the service requestor, service provider and service registry.

- Service provider: The Service provider is responsible for publishing the service on the web with specific details and protocols to guide the service requestor's use.
- Service Registry: The Service registry assists the service requestors in searching the correct services with UDDI data structure.
- Service requestor: The service requestor finds the right service from the service registry that is published by the service provider. After the correct service is found, the requestor and provider negotiate the format of the request, along with other protocol issues. Finally, the requestor can access and invoke the service of the provider.

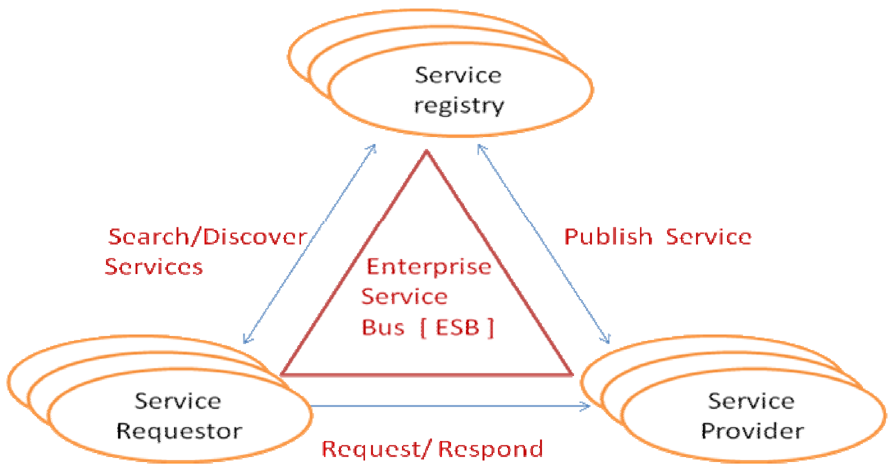


Figure 2.6 Major elements of SOA(Requestor, Registry , Provider)

2.3 SOA standard

2.3.1 XML

XML was developed as a general-purpose specification by the W3C to support dynamic content creation and overcome the limitations of HTML. Using XML we can define any content of an element in a meaningful way[25]. The example below describes not only each element of the attributes, but also the informational structure for the data.

```
<University>
       <UniversityName region = “US”>
         Oklahoma State University
       </UniversityName>
<Student>
   <StudentName> Joy  Kim </StudentName>
   <StudentAddress>124 Brumley apt #200 Stillwater </StudentAddress>
   <StudentCollege> Oklahoma State University</StudentCollege>
   <StudentPhone> 403-334-1343 </StudentPhone>
   <Gpa> 3.7 </Gpa>
 </Student>
</University>
```

2.3.2 SOAP

Simple Object Access Protocol(SOAP) is a specification for common format message structured by XML for communication over HTTP, between service provider , service consumer, and service registry [14].

SOAP is composed of three major blocks: the envelope, the header and the body. The header is noncompulsory, and can include one or more header blocks carrying the attribute of the message or defining the qualities of service for the message. Headers are intended to carry contexts or any application defined information related to the message, such as security tokens, transaction identifiers, and message correlation mechanisms. The body is essential and contains one or more body blocks, encompassing the message itself [25].

- SOAP Envelope: The SOAP envelop symbolizes the start and the end of the message, so that the receiver knows when an entire message has been received. The SOAP envelope solves the problem of knowing when you're done receiving a message, and are ready to process it. The SOAP envelope is therefore basically a packing mechanism.

- SOAP Header: The headers are the main mechanisms by which SOAP can be extended to include additional features and functionality, such as security, transactions, and other quality-of-service attributes associated with the message. The header is encoded as the first immediate child element of the SOAP envelope.

- SOAP Body: The SOAP body contains the application-defined XML data being exchanged in the SOAP message. The body must be contained within the envelope and must follow any headers that might be defined for the message. The body is defined as a child element of the envelope, and the semantics for the body are defined in the associated SOAP schema.

2.3.3 WSDL

The Web Services Description Language (WSDL) is a standard way to describe a Web service. It describes and publishes the protocol and format [25].

- Data Types: in the form of XML schemas of some other possible mechanism – to be used in messages.
- Message: an abstract definition of the data, in the form of a message presented either as an entire document, or as arguments to be mapped to a method invocation
- Operation: the abstract definition of the operation for a message, such as naming a method, message queue, or business process, that will accept and process the message
- Port type: an abstract set of operations mapped to one or more end points, defining the collection of operations for a binding, the collection of operations. Since these operations are abstract, they can be mapped to multiple transports through various bindings.
- Binding: the concrete protocol and data formats for the operations and message defined for a particular port type.
- Port: a combination of a binding and a network address providing the target address of the service communication
- Service: a collection of related end points encompassing the service definitions in the file. The services map the binding to the port and include any extensibility definitions.

2.3.4 UDDI

The UDDI registry accepts information describing a business, including the web services it offers, and allows interested parties to perform online searches and downloads of the information. UDDI information is often described as being divided into three main categories of business information[25].

White page	Business name and address, contact information , Web site name, and data Universal Numbering
Yellow page	Type of business, location, products, geographical location, industry type, business ID
Green page	Technical information about business service

2.3.5 Problems in UDDI data Structure

The UDDI data structure provides so many options and extensions, that it’s almost impossible to predict the level of consistency that will be achieved among entries for different businesses. In other words, it may be very difficult to predict the type of detail available for a given entry. If UDDI is ever to succeed, the data will have to be normalized and regularized a good deal more than it is [25].

2.4 Enterprise Service BUS

2.4.1 Overview of ESB

The word “bus” is a reference to the physical bus that carries bits between devices in a computer. In the Service Oriented Architecture, the Enterprise Service Bus(ESB) refers to the construct of a software architecture that is implemented using middleware infrastructure, which supports standard-based event driven message exchange engine between complex service architectures [13].

ESB is the core component of SOA, it supports Transport protocol management, Message transformation, Security, Reliability, Management, Transaction, Orchestration of service[7].

As shown below, there are four different kinds of Service Requestors and four service providers developed on different platforms, ESB allows the exchange of a standard set of message between Service requestors and Service providers.

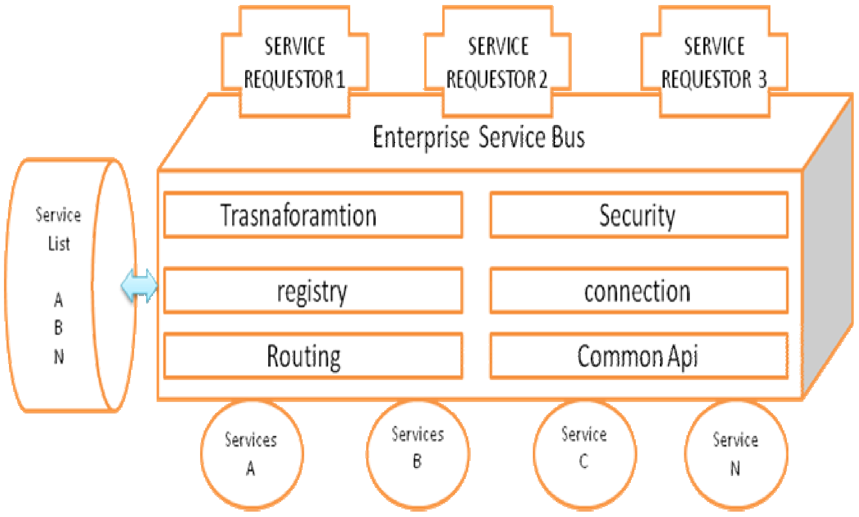


Figure 2.7 Architecture of enterprise Service Bus(ESB)

2.4.2 IBM WebSphere's ESB and SOA

This is a concept diagram of ESB/SOA developed by IBM WebSphere research group. They proposed an "ESB hub" architecture to support routing, transformation, mediations, security etc[14].

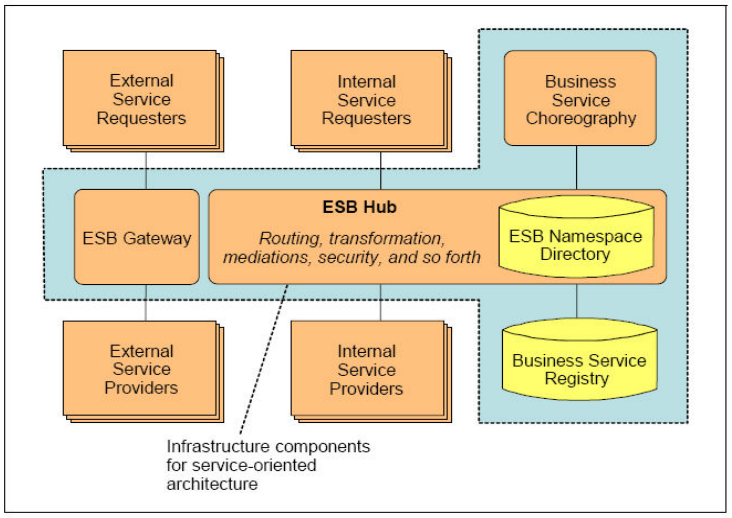

External Service Requesters
Internal Service Requesters
Business Service Choreography
ESB Gateway
ESB Hub
Routing, transformation, mediations, security, and so forth
ESB Namespace Directory
External Service Providers
Internal Service Providers
Business Service Registry
Infrastructure components for service-oriented architecture

Figure 2.8 ESB and SOA [14]

2.4.3 ESB capabilities

IBM WebSphere ESB capabilities are as follows[14] :

- Communication: An ESB should provide event-oriented middleware over HTTP infrastructure and service interaction over various protocols.
- Service Interaction: An ESB supports declaration of service operation and interaction and message correction.
- Integration: An ESB supports heterogeneous environmental technologies such as EAI technologies, JDBC, FTP, EDI, J2EE connector architecture, client API for various

languages and platforms.

- Management: An ESB enables the monitoring and control of services and interacts with system management software.
- Quality of Service: An ESB provides different qualities of service for integrity of data.
- Security: An ESB should support security infrastructures, identification and authentication, access control, confidentiality of data, security management and any other security related aspects.
- Service Level: An ESB enables handling business service level agreements.
- Message processing: An ESB has the capability of integrating message, object, and data models among the application components of an SOA.
- Modeling : An ESB should support the use of development tools and be capable of identifying different models for inter and external services and processes.
- Infrastructure intelligence : An ESB supports autonomic pattern recognition.
- Management and autonomic : An ESB supports autonomic self-healing, self-configuring, and dynamic routing.

2.4.4 WebSphere Enterprise Service BUS

The WebSphere ESB infrastructure enables connecting applications that have standards-based interfaces as described in the WSDL file. WebSphere Enterprise Service Bus adds the following values to the application server :

- Provides built-in meditation(centralizes logic, routing, transformation, data handling) to create integration logic for connectivity.

- Offers support for J2EE Connector Architecture.

2.4.5 Structure of WebSphere Enterprise bus

A service interaction in SOA defines both service consumers and service providers. The role of WebSphere ESB is to intercept the request of service consumers and fulfill additional tasks in mediations in order to support loose coupling.

Mediation tasks include :

- Centralizing the routing logic, which provide transparency of the services.
- Acting as a façade in other to provide different interfaces between service consumers and providers.
- Interfaces are defined in a WSDL document.

2.4.6 Broker

The broker is a set of application processes that host and run message flows. When a message arrives at the broker from a business application, the broker processes the message before passing it on to one or other business applications. Execution groups enable message flows within the broker to be grouped together. Each broker contains a default execution group.

2.5 OASiS

OASiS is an Object-Centric, Ambient-aware, Service-oriented sensor net programming model and middleware implementation for WSNs application, proposed by Vanderbilt University. OASiS is a lightweight framework which avoids the use of XML-based messages found in Web Servcie3 standards [25].

The OASiS programming model is composed of a Finite state machine, Node Manager, Object manager, Dynamic Service Configurator, and WWW Gateway. The Gateway resides on a sensor network base station and provides access to web services by translating node-base byte sequence messages. There are three types of messages handled by the Node manager : service discovery message, service binding messages, service access messages[25].

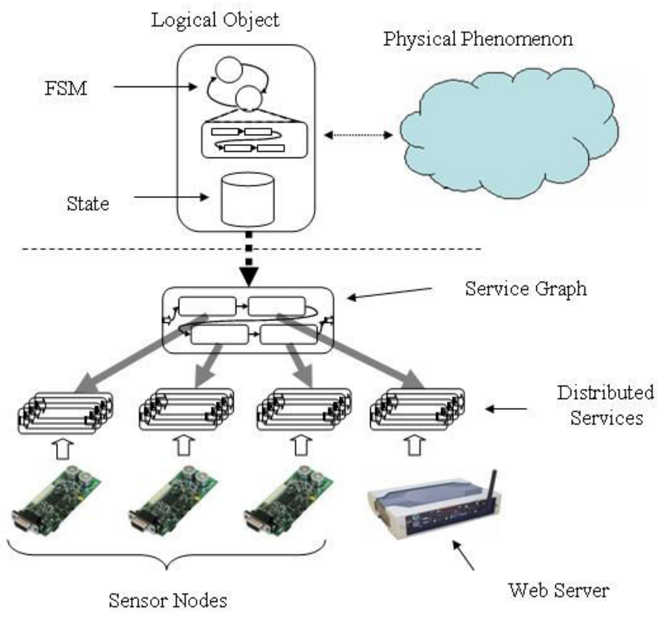


Figure 2.9 OASiS Programming Model [25]

The OASiS is a very useful Architecture in developing WSN applications. The mini-SOA/ESB and OASiS comparison is shown below.

Comparison	OASiS	Mini-SOA/ESB
Goal	- Provide SOA for Sensor networks - Propose a programming model and middleware implementation for WSN.	- Provide the possibility of integrating WSN applications as an open standard frame work.
Proposed Model	Logical Model	Logical Model
Key Idea	- Service graph concept is used for connection between two services - The WWW gateway resides on a sensor network base station and provides access to Web services. - A gateway application is developed on a base station - The middleware services include a Node Manager, Object manager, and Dynamic Service Configurator.	- Enterprise Service Bus concept used - A mini-SOA/ESB Service Engine supports a common interface of sensor network platforms. - Sensor Web Domain used for sharing information about sensor service applications among Service providers and consumers. - The mini-ESB includes a message broker, service transformer, consistency monitor and service publisher.
Implementation	- Scalability analysis using Prowler - The feasibility and effectiveness of OASiS was evaluated using a simple tracking application.	- Service availability with Alternative Service List - UDDI consistency

Table 2.2 OASiS[25] vs. mini-SOA/ESB

CHAPTER III

PROPOSED MINI-SOA/ESB FOR WSN

A general approach to desegregate sensor nodes into the sensor Grid is to choose the Grid Standard and APIs. The Open Grid Services Architecture (OGSA) is based on the major technology of SOA standards like XML, SOAP, and WDSL. If Sensor data is accessible in the OGSA framework, it is easy to share data and services developed by various service providers. However, since sensor nodes have restricted computing power and processing capacity, it may not be possible for sensor data to be encoded in XML format within SOAP envelops or transported using internet protocol to applications. Grid services are also complex in order to be implemented directly on most simple sensor nodes[10].

Therefore, we propose a new concept of SOA/ESB architecture for WSNs, called "mini-SOA/ESB," to address these design issues.

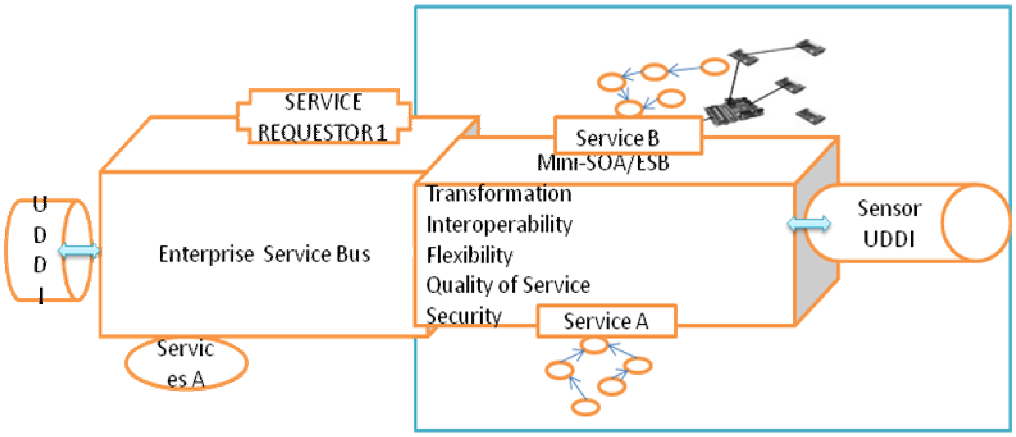


Figure 3.1 Mini-SOA with Service Oriented Architecture

3.1 Relationship between SOA and mini-SOA

How are SOA and mini-SOA related? SOA focuses on the integration of the Enterprise service, whereas mini-SOA focuses on the interoperability between different kinds of WSN applications. Let us assume the fire department has a "fire monitoring system." This system consists of two different parts: mini-SOA and SOA.

When a fire happens, a fire department needs information such as the best route, ambulance information, location of the fire and the spread of fire, in order to dispatch firefighters and ambulances.

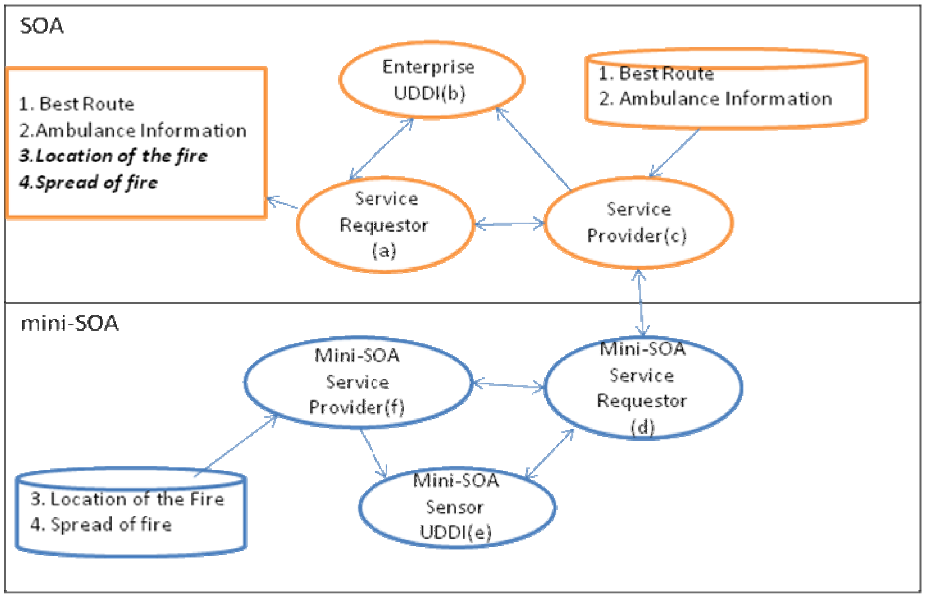


Figure 3.2 Relationship SOA and mini-SOA

In this case, how do we get information from the "fire monitoring system?" The processing steps are as follows:

The SOA’s Service requestor (a) finds the right service from the SOA’s Enterprise UDDI (b). After the correct service is found, the SOA’s service provider (c) checks information from SOA’s Service requestor (a). If SOA’s service provider (c) does not have enough information, the service provider (c) sends a requests for information to the mini-SOA’s Service requestor.

The mini-SOA’s Service requestor (d) finds the right service from the mini-SOA’s Sensor UDDI (e). After the correct service is found, the mini-SOA’s Service provider (f) provides information detailing the location of the fire (3) and spread of fire (4). The mini-SOA’s Service requestor (d) receives information from the mini-SOA’s Service provider (f), and transfers information to the SOA’s service provider (c).

The SOA’s service provider (c) combines information pertaining to the location of the fire (3), spread of fire(4), Best route (1) and Ambulance information (2), then transfers the message to the SOA’s Service requestor (a). Finally, SOA’s Service requestor (a) gets all the information that he requested.

3.2 Design

In order design the mini-SOA/ESB architecture for WSNs, we need to consider a number of features. As shown in Table 3.2, design issues for integration have been proposed[10][15][12][16].

Based on these integration concepts, the mini-SOA/ESB design guidelines are categorized by Transformability, Interoperability, Flexibility, Security and Quality of Service and Management.

Proposed Architecture	Design Consideration	Reference
Proxy Software Architecture	-Data Management -Information Services -WSN Connectivity -Power Management -Security -Availability -Quantity of Service -Grid Interface, WSN Scheduler, WSN Management	[10]
IP-enabled	-IP over sensor network Technologies -Ad hoc Networking -Gateway discovery -Service Discovery -Mobility Management -Security	[15]
Server-Client Approach & Peer-to-peer Approach	-Heterogeneity -Scalability -Publishing and discovering sensor resources -Query aggregation -Interconnection -Integration -Data Collection and data storage -API for high-level application	[12]
Tiny Web Services	-Interoperability -Improves the programmability -Easy to integrate with enterprise system via Internet -Providing Multiple gateways for converting between each sensor manufacturer and the application	[16]

Table 3.1 Design Considerations, depending on Architecture

Another consideration for the design requirements of mini-SOA is the possibility of supporting various kinds of sensor application platforms, such as OS-based architecture, VM-based architecture, Middleware architecture and Stand-alone protocols [9]. See table 3.2.

Prototype platform	Proposal	Code
OS-based architecture	TinyOS BerthaOS EYE OS MOS	OS-1 OS-2 OS-3 OS-4
VM-based architecture	Sensorware MagnetOS Mate’	VM-1 VM-2 VM-3
Middleware architecture	MiLAN Cluster-based Middleware in Qos-aware Middleware in SINA TinyDB Cougar LIME MARE RSCM	MA-1 MA-2 MA-3 MA-4 MA-5 MA-6 MA-7 MA-8 MA-9 MA-A
Stand-alone protocols	GSD Task migration in	SA-1 SA-2
Others		O-1

Table 3.2 Prototype platform

3.3 Requirements for mini-SOA/ESB

3.3.1 Transformability

Transformability is the ability to message transformation, which combines messages between service provider and service consumer.

Assume a service provider publishes services Service1, Service2, where each service consists of Room1 and Room2 ‘s Temperature and Node power. After publication, these services can be used by the service consumer.

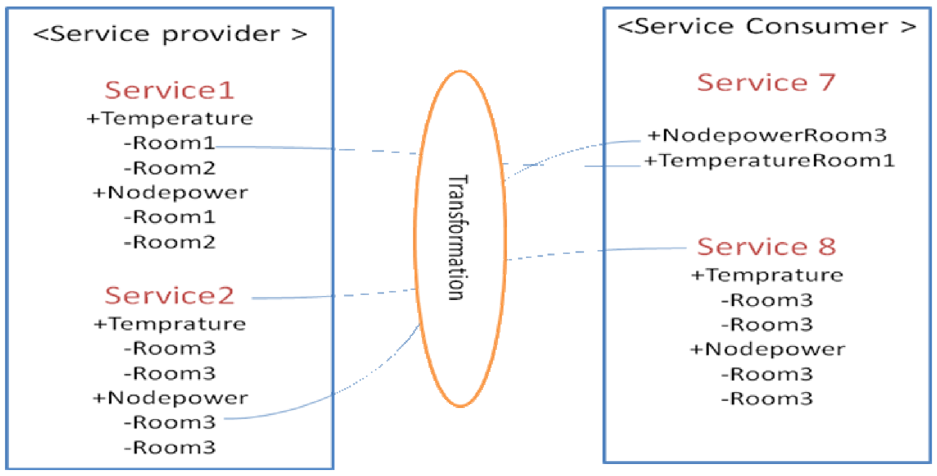


Figure 3.3 Service Transformation

In order to use these services, the service consumer needs to create a new service. As shown in Figure 3.3, Service7 and Service8 are created. Service8 is generated from Service2, with the same compositional format Temprature(Room3, Room4) and NodePower(Room3,Room4), but with a different name.

Service7 is made of Service2's NodePower(Room3) and Service1's Temperature(Room1) with a different format. In this case, the mini-SOA/ESB provides the mechanism to format mapping functions between the Service provider and the Service Consumer.

3.3.2 Interoperability

In order to share sensing resources on the Web, an appropriate interconnection approach must be introduced, which is spatially deployed in different locations[21]. Interoperability is a key factor in supporting communication interfaces of different sensor platforms, like OS-based, VM-based and Middleware-based architectures.

3.3.3 Flexibility

Flexibility is the ability to interface between WSN applications and Enterprise level application services. A mini-SOA/ESB should keep SOA's major open standards, for example, XML, SOAP , WDSL, BEPL(Business Process Execution Language), and UDDI. In order to interact with enterprise level applications that are not tied to a specific vender, Mini-SOA/ESB should automatically generate a XML format message to support the SOAP protocol, which is a highly-distributed architecture.

3.3.4 Security

Wireless sensor networks are prone to security problems, such as the compromising and tampering of sensor nodes, eavesdropping of sensor data and communication, and denial of attacks[10]. To make a secure mini-SOA/ESB model, it is to necessary to ensure the protection of sensor networks from attackers.

3.3.5 Quality of Service

Sensor nodes have restricted battery power and processing capability. If some services are not available, the mini-SOA/ESB needs to have a failure of recovery plan or an Alternative Service Selection.

3.4 mini-SOA/ESB Architecture

Figure 3.4 is the proposed architecture of a new concept for integrating mini-SOA/ESB with WSNs. Mini-SOA/ESB is composed of a Mini-SOA/ESB Server Engine, Mini-SOA/ESB, Mini-SOA Orchestrator and Sensor UDDI. Mini-ESB has a Message broker, service transformer, consistency monitor and service publisher.

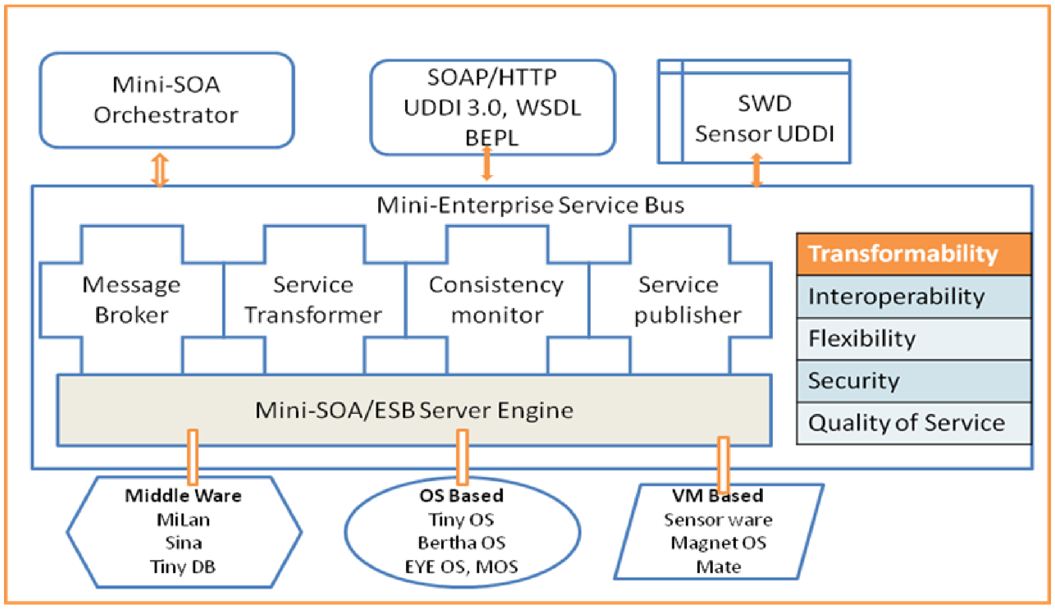

Mini-SOA Orchestrator
SOAP/HTTP UDDI 3.0, WSDL BEPL
SWD Sensor UDDI
Mini-Enterprise Service Bus
Message Broker
Service Transformer
Consistency monitor
Service publisher
Mini-SOA/ESB Server Engine
Transformability
Interoperability
Flexibility
Security
Quality of Service
Middle Ware MiLan Sina Tiny DB
OS Based Tiny OS Bertha OS EYE OS, MOS
VM Based Sensor ware Magnet OS Mate

Figure 3.4 Architecture of mini-SOA/ESB

3.4.1 Mini-SOA Orchestrator

The mini-SOA Orchestrator provides a user-convenient GUI, which interacts with the Service transformer, message broker and sensor UDDI. A good GUI design not only relates to the system architecture, but is also one of the most important factors for increasing the productivity of application development and management. The mini-SOA Orchestrator requirements are as follows :

- Visual display function of published service information.
- Easy to create sensor application processes.
- Provides an active service monitoring function.

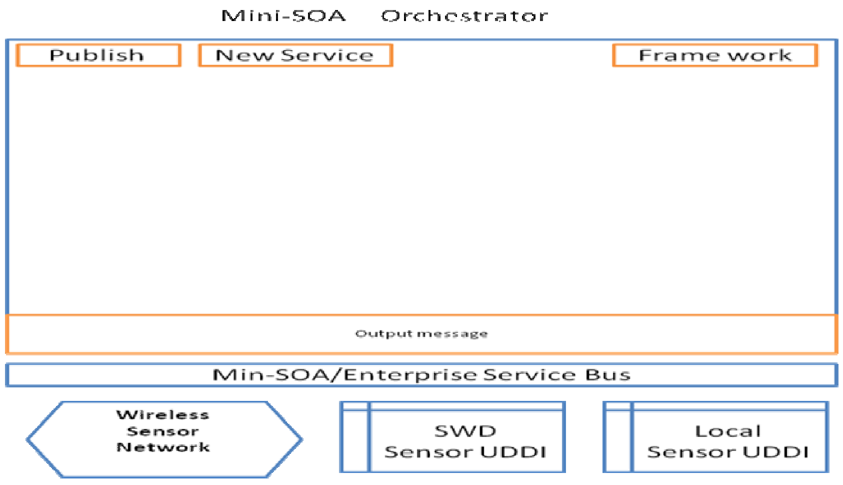


Figure 3.5 mini-SOA Orchestrator

3.4.2 Mini-SOA/ESB Service Engine

The Mini-SOA/ESB Service Engine is the heart of the new mini-SOA/ESB architecture. This service engine supports common interfaces of various kinds of sensor network platforms, for instance, middleware based (Milan, Sina, Tiny DB), OS based(Tiny OS,Bertha OS) and VM based platforms.

3.4.3 Message Broker

The message broker controls all of the interacting messages between the Message broker, Service transformer, Consistency monitor and Service publisher. When a message transfers from the service requestor, the broker passes the message to the Service transformer and service consistency monitor.

3.5 Sensor Web Domain

Sensor Web Domain (SWD) is the web site for sharing information about sensor service applications among Service providers and Service consumers, for example Google or Yahoo search engines. This site presents every published fact that the service has on file, for example published service List, contact information, service creation time, service reliability rating and alternative service list. This is the proposed site map of the web site.

The name of SWD's URL(Uniform Resource Locator) will be " www.sensorUDDI.org".

As shown in Figure 3.6, this site is specially designed for sharing sensor data in the form of sensor applications centric on the Web, and the user can also access this site via the Mini-SOA Orchestrator.

- □ User Login
  - + user authentication
- □ Sensor-UDDI
  - + register UDDI
  - + search published service
  - + publish service
- □ Service Level Management
  - + Service level category
  - +Service Authentication
- □ mini-SOA/ESB management
  - + software download
- □ Contact Information
- □ How to get Authentication

Figure 3.6 Site map of SWD web site

3.6 Sensor-UDDI structure

The general UDDI data structure has so many selections and extensions that it is almost impossible to maintain a level of consistency[7]. Therefore, we propose a new model of UDDI that is aimed for WSNs.

Any sensor application should publish to the sensor UDDI in the mini-Sensor-UDDI.org domain and service publish domain itself. There are three sensor-UDDI domains: at the Service provider, Service consumer and sensor-UDDI.org domains.

Column Name	**Description**
BusinessKey	AREA-XXXX-XXXX
AuthenticationStep	Approved, processing,denied
ServiceName	A1
Service List	{a1,a2,a3,a4,a5},
Alternative Service List	{a1:a2, a2:a3, a4:a5}
Service Level	0,1
ServiceCreationDateTime	YYYY-MM-DD 13:00
EffectiveServiceDateTime	YYYY-MM-DD 24:00
LastConsistencyCheck	YYYY-MM-DD 13:00

Figure 3.7 UDDI elements

- BusinessKey: Business key is the unique key in a Service.
- AuthenticationStep: When publishing a service, AuthenticationStep is processing(0), If a Service is approved, it will be changed to approved(1), if denied, to denied(2).
- ServcieName: Name of the service.
- ServiceList: Published by a Service name, lists the set of processor names.

- Set of Processor names, which is published by a service Name.
- Alternative Service List(ASL): The set of lists can be replaceable. An ASL can be created at the time of service and published as an optional requirement.
- Service Level: Sever Level 0 - These are the basic services published on one platform. Service Level 1 – this is a combination of Level 0 Services composed of services from different platforms.
- ServiceCreationtime: Service publishing time.
- EffectiveServiceDateTime: Service expiration time.

The new type of data structure for the mini-SOA , called "sensorUDDI", is composed of a Service name, Service list and Alternative service List. This sensorUDDI is specially designed for increasing QoS, defined in terms of service availability and consistency.

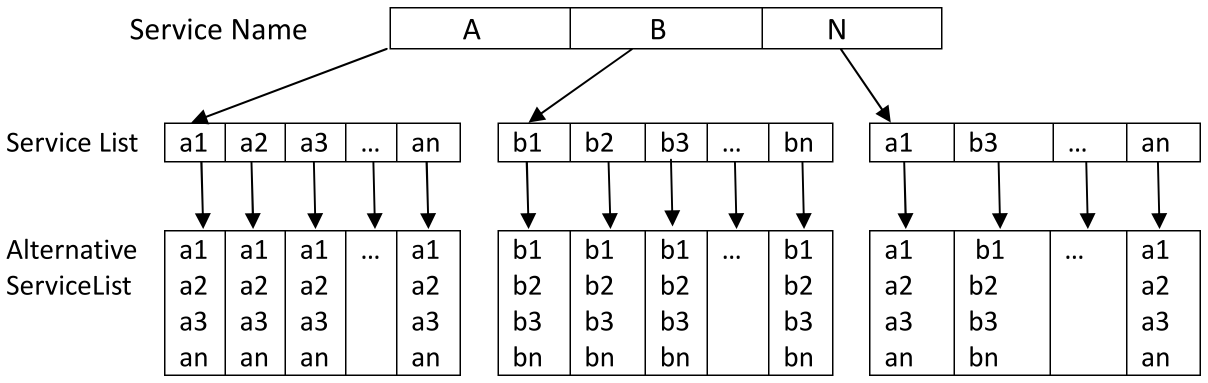


Figure 3.8 Architecture of Service list of UDDI

The alternative service list(ASL) is created with its Service Name at the time of service publication. The list of ASL is sorted by availability of service.

Let’s assume, if we have a service list as follows :

a1:{a2, a3, a4}

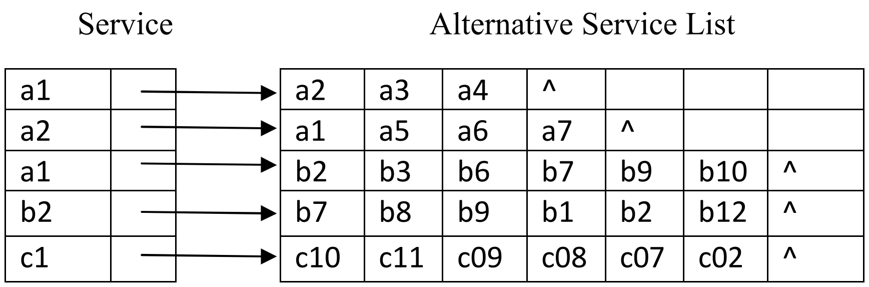
a2:{a1, a5, a6, a7}

b1:{b2, b3, b6, b7, b9, b10}

b2:{b7, b8, b9, b1, b2, b12}

c1:{c10, c11, c09, c08, c07, c02}

They can be described as:



a1	a2	a3	a4	^			
a2	a1	a5	a6	a7	^		
a1	b2	b3	b6	b7	b9	b10	^
b2	b7	b8	b9	b1	b2	b12	^
c1	c10	c11	c09	c08	c07	c02	^

Figure 3.9 Alternative Service List (ASL)

3.7 Operation sequence of mini-SOA/ESB

The operation of the Mini-SOA/ESB follows the general SOA steps of service find, service bind, service and request/respond. The difference lies in the management and reference of UDDI information. The mini-SOA/ESB contains three different sensor-UDDIs, at sensorUDDI.org, service provider and service consumer. These three sensor-UDDIs maintain the same structure as shown in Figure 3.10, but hold different service lists with different contents.

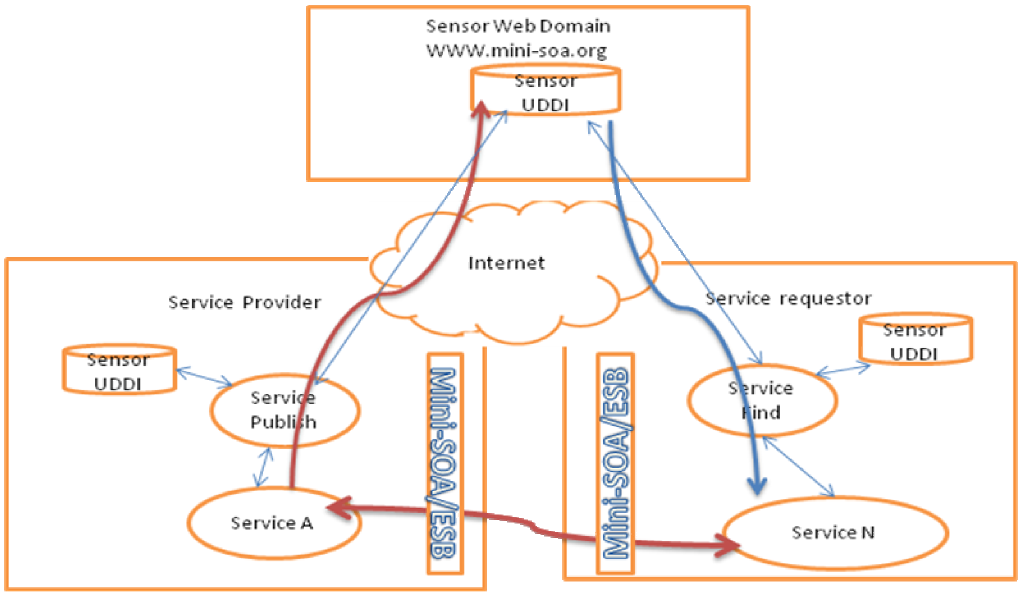


Figure 3.10 Service find, bind, request/respond procedure

A Service provider needs to publish a service at its local site and Sensor Web Domain(SWD, Figure 3.6). When a service requestor requests a service, the service finder finds the correct service from the service requestors site, not from the SWD. This is because the SWD contains the complete information related to sensor applications, whereas each service requestor site has copied or duplicate information from the SWD.

Why do we need three different UDDIs at these three places? The main reason for keeping sensor-UDDI information separate is to assure the Quality of Service. If the SWD site breaks down accidently, then every service provider and consumer has to wait until its recovery. As long as the information is kept at each node's own sensor-UDDI, the service can run without interruption if the SWD site fails. As a result of this approach, QoS in the min-SOA/ESB infrastructure will be greatly improved.

3.7.1 Service Publication procedure

The service publisher is responsible for publishing services in a specific format, including service level, business key, discovery URL, service Creation date, effective service time, service info and alternate service list.

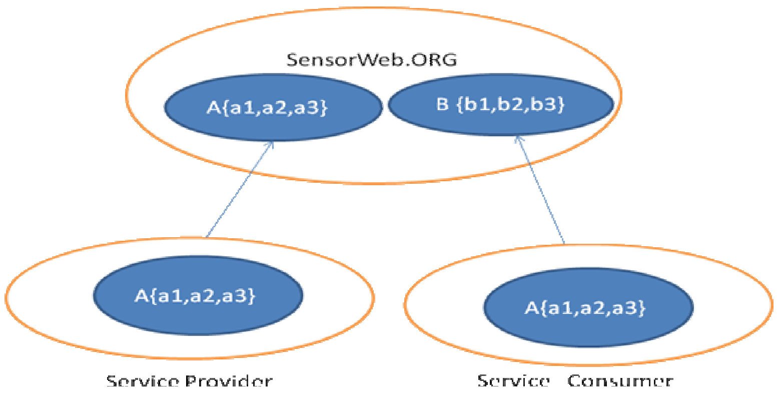


Figure 3.11 Service Publish A,B

When publishing a service, the mini-SOA Orchestrator acts as a user interface. By manipulating the interface, the user develops a service without learning specific details, such as Tiny Os, Sensorware, MilAN, TinyDB, etc. The Mini-SOA/ESB Service Engine provides an Application Programming Interface to interact with any kind of platforms made by different manufacturers.

The first step in publishing a service is selecting the working platform. The min-SOA Orchestrator screen displays the platforms lists. For instance, if you are working on crossbow motes, you should select TinyOS tab (OS-1). After that, the user defines a main service name and alternative service, and presses the publish button. The service publisher makes a process at a service provider's site, and writes to the sensor-UDDI and sensor web site's UDDI. When a service is published successfully, the screen displays the messages "Service Successfully Created."

3.7.2 New Service Creation

Once services are published at the sensor.org, the Service consumer who registered at Sensor.org can use the services. Using a Mini-SOA Orchestrator, we can create a new service out of a composite of different services. The Graphical User Interface provides detailed information on the published services.

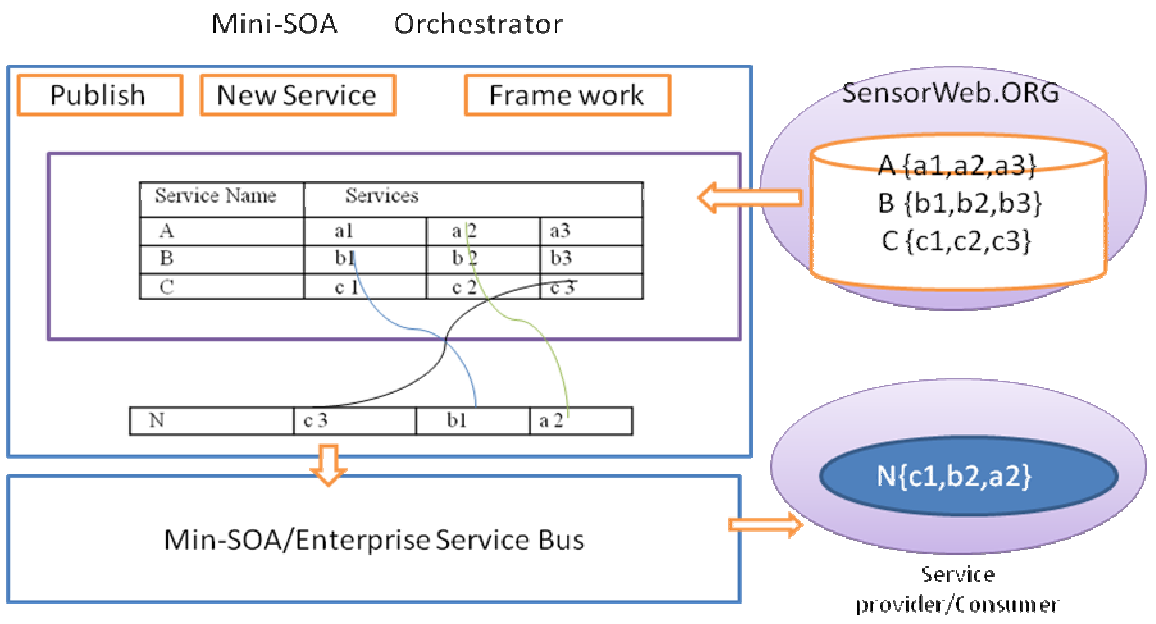


Figure 3.12 New Service Creation procedure

Assume Service “A” is published with its Service processors { a1, a2, a3 }, Service “B” with {b1, b2, b3}, and Service “C” with {c1, c2, c3}. The New creation steps are as follows:

```
Step 1:  for the published Service,
            Searches required services from the SWD.
Step2 :  Mini-SOA Orchestrator displays information,
            Makes new service name and chooses services based on displayed information
           // Example , Service name  “N”  and  its Services c3, b1, a2
Step3 :  chooses code of framework to create
           // Table  3.2   Prototype platform
Step4:  Clicks publish button.
          // New service processors are created, named “N”, with its own processors.
Step5:  updates sensor-UDDI information
          // Service provider/consumer site and SWD.
```

3.7.3 Service Availability

Service availability is the ability to maintain services without errors or suspension of services over a period of time.

There are two cases of Availability,

Case 1. Without Alternative Services:

$$A(t) = \left(\sum \text{Up}\ \{\text{Services}\} / \left(\sum \text{Up}\{\text{Services}\} + \sum \text{Down}\{\text{Services}\}\right)\right) * 100$$

Up = Up time of all published services
Down = Down time of published services

If we publish the service name “N” and its Services { a1, a2, …, an }, then the Service Availability is described as

$$A(t) = \left(\sum \text{Up}\ \{a1, a2, \ldots, an\} / \left(\sum \text{Up}\{a1, a2, \ldots, an\} + \sum \text{Down}\{a1, a2, \ldots an\}\right)\right) * 100$$

Case 2. With Alternative Services:

In this case, the calculation of Service availability is different from Case1. This is because as long as we have ASL, even though a Service is down, this service can be replaced by another service among the ASLs. Then this service is regarded as running.

$$A(t) = \left(\sum \text{Up}\ \{\text{Services}\} / \left(\sum \text{Up}\{\text{Services}\} + \sum \text{Alt}\{\text{Down}(\text{Services}) - \text{up}(\text{ALT Servvices})\}\right)\right) * 100$$

Up = Up time of all published services
Alt = Down time – Alternative Service time

To increase availability of the service, apply alternative service select algorithms:

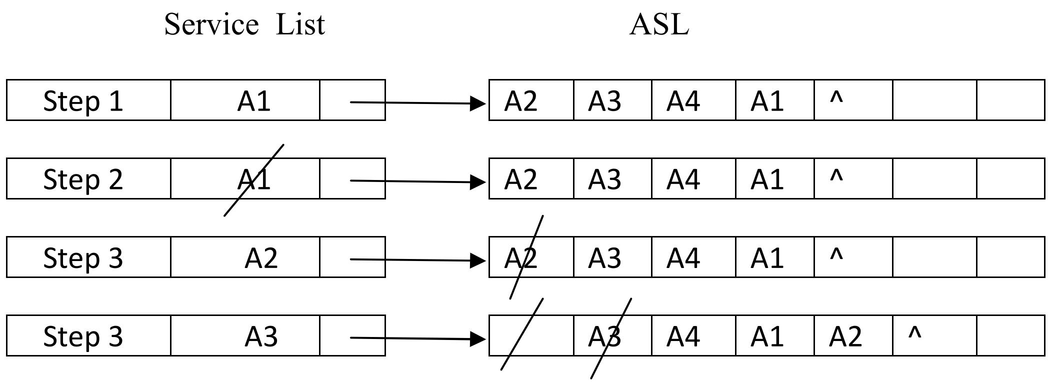


Figure 3.13 Alternative Service Selection

Step 1: Service is suspended due to errors,

Service A1 put at the end of the ASL order.

Step 2: Removes Service A1 from the Service List

Step 3: Searches an available service from the ASL.

Step 4: If a service is found from ASL

The service is moved from ASL to the Service List

The selected service is removed from ASL

The services are shifted left one by one, in the ASL

Step 5:if not found, then go to Step 3.

3.7.4 UDDI Consistency

Service consistency is the ability to maintain consistent sensor-UDDI information between the service provider's sensor-UDDI and the sensor-UDDI at SWD. For example, a service was published, but due to problems at the local web site, such as a node's new power battery, or because published services are not working, the local sensor-UDDI's information is changed by the service publisher. This information should be updated to the sensor-UDDI at SWD.

To maintain accurate sensor-UDDI information, a Consistency Check monitor checks at regular time intervals. If mismatched services are detected, corrections are made.

$$C(t) = (\sum \mathrm{MatchCount}/(\sum \mathrm{MatchCount} + \sum \mathrm{MisMatchCount})) * 100$$

t = measurement time
MatchCount = number of count of match case
MisMatchCount = number of count of mismatch

CHAPTER IV

SIMULATION

4.1 Objective of the simulation

The aim of the simulation is to validate the proposed approach to Quality-of-Service in mini-SOA/ESB. To increase the QoS, this simulation measures two aspects, Service availability and UDDI consistency. To determine the Service availability, we used the ASL list as a test set of services, and tried to affect the number of available ASLs. For the UDDI consistency, we compared two sensor-UDDIs between the Sensor Web Domain and service provider domain.

4.2 Development tools and programming languages

All the experiments are conducted on a AMD Truion[tm] 64 * 2 CPU with 1.61Ghz of RAM and Microsoft Windows XP professional Version-2002 Service Pack3. We implemented our algorithms in Eclipse SDK 3.4.1.

4.3 Assumption

The mini-SOA/ESB architecture is composed of various components, such as the service orchestrator, Engine, enterprise service bus, etc. It is beyond the scope of a master is thesis to implement all of the components of mini-SOA/ESB. Therefore, we implemented a service consistency check procedure, which is a small part of the mini-SOA/ESB related to Quality-of-Service.

4.4 Simulation of service availability and consistency

The test environment is composed of four processors. For the Service availability test, we used a Failure Control processor, Failure Recovery processor, and Sensor Monitor processor. For the UDDI consistency, a Consistency checker is used to compare sensor-UDDI between SWD and Service provider/Consumer.

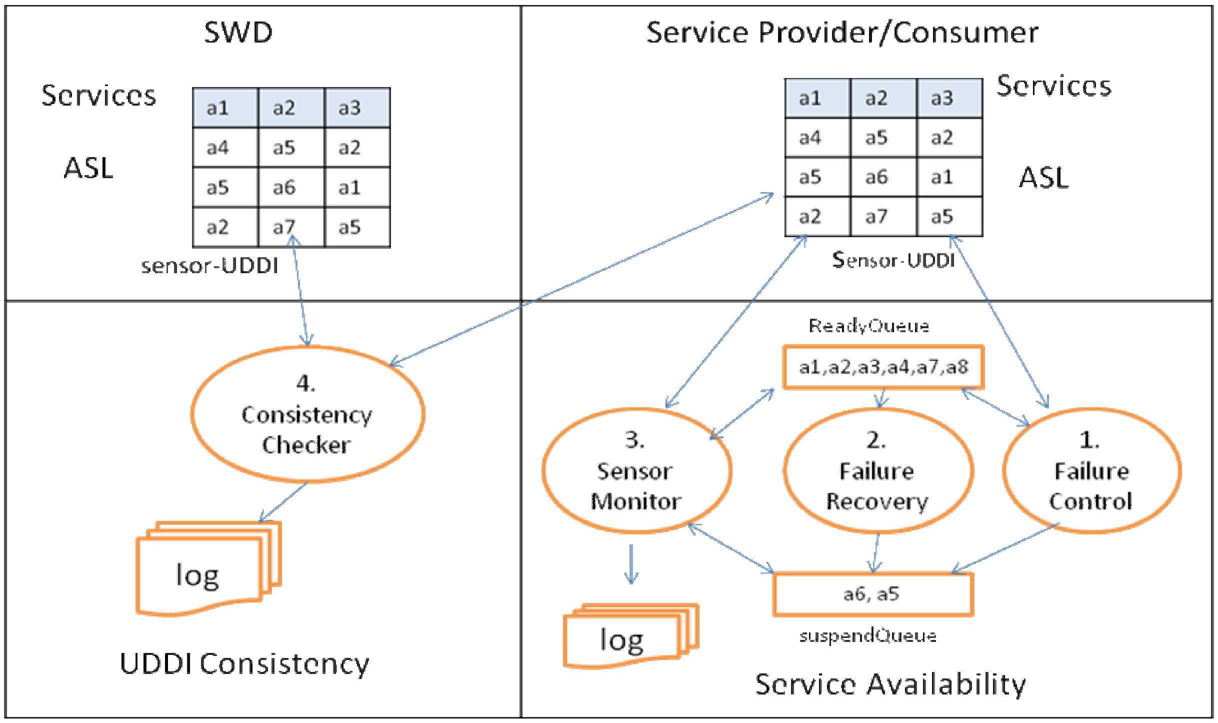


Figure 4.1 DFD of Service Availability and Consistency simulation

4.4.1 Service Availability processors

(1) Failure Control processor : As shown Figure 4.1, Services are published at the SWD and Service provider/Consumer domains with their ASLs. For instance, Service “a1” is published with its ASL {a4,a5,a2}. If service “a1” fails, then the Failure Control processor selects a service from ASL and replaces the failed service.

Algorithm 4.1 : Failure Control

```
Step 0:  Puts all published services in the readyQueue
          // ReadyQueue   ← sensor UDDI

Step1 : While ( Time Periode )

          1.Generates magicNumber to pick one service from the readyQueue

          2. puts readyQueue's Service to the suspendQueue
             // suspendQueue = readyQueue(magicNumber)

          3. puts readyQueue's Servce to the tail of the ASL

          4. finds an available Service among ASL

             4.1  if  an available service is found from the ASL, then

                   4.1.1 Moves a Service to Service List

                   4.1.2 Removes selected service from the ASL
                         //The services are shifted left one by one, in the ASL

             4.2 if available service is not found from ASL,
                 Perform 4.
```

(2) Failure recovery processor: This processor is responsible for recovering a Service which was detected to be failed by the Failure Control processor.

Algorithm 4.2 : Failure recovery

```
Step 0:  for all services in the suspendQueue
          // ReadyQueue ← sensor UDDI

Step1 : While ( Time Period  )

          1. Generates magicNumber to pick one service from the suspendQueue

          2. puts suspendQueue's Service to the  readyQueue
```

(3) Sensor Monitor processor: The Sensor-Monitor processor's role is to check the status of Services and Alternative services at regular time intervals. Table 4.1 shows a partial result

generated by the sensor monitor processor. In this table, service "A_A2" is suspended at the time interval 40, but this service is replaced by one of the Alternative services from ASL.

Service Name	***Time (Second)***	***Service Status***	***ASL Status***
Servie: A_A2	Interval: 24	Status : 0	ASL Status : 1
Servie: A_A2	Interval: 28	Status : 1	ASL Status : 1
Servie: A_A2	Interval: 32	Status : 1	ASL Status : 1
Servie: A_A2	Interval: 36	Status : 1	ASL Status : 1
Servie: A_A2	Interval: 40	Status : 0	ASL Status : 1
Servie: A_A2	Interval: 44	Status : 0	ASL Status : 1
Servie: A_A2	Interval: 48	Status : 0	ASL Status : 1
Servie: A_A3	Interval: 4	Status : 1	ASL Status : 1
Servie: A_A3	Interval: 8	Status : 1	ASL Status : 1
Servie: A_A3	Interval: 12	Status : 1	ASL Status : 1

Table 4.1 Service and Alternative services status

Algorithm 4.3 : Sensor Monitor

```
Step 0:  Puts all published services in the aMonitorV (monitor vector)

Step1 : While ( Time Periode, timeInterval )

        1. if aMovitorV is in  readyQueue
           Sets aServcie status = 1

          else
           Sets aServcie status = 0

           1.1 checks if  its alternative Service is available
               Sets aAltStatus status = 1
           else
               Sets aAltStatus status = 0

        2. writes to the logfile with ( serviceName, Timeinterval,
                                  servicestatus, AltServiceStatus)
```

4.4.2 UDDI consistency processor

(4) Consistency Checker: To ensure UDDI Consistency, the consistency checker checks inconsistencies between the service provider's sensor-UDDI and SWD. If any inconsistency is found, update information is sent from the sensor-UDDI to the SWDs.

Algorithm 4.4 : Failure Control

```
Step 0:  for the service provider's sensor-UDDI
Step1 : While ( Time Periode, , timeInterval )
        1. compares (aService.ServiceProvider <> aService.SWD)
          aMisMatch++; // Increases mismatch counter
          1.1 compares alternativeService.ServiceProvider <> alternativeService.SWD
              altMisMatch ++;
        2. writes to the logfile with ( serviceName, aMisMatch,
                                        timeInterval, altMisMatch )
```

4.5 Experimental Results

4.5.1 Service Availability

Test Case 1 : ASL (n = 4)

Test conditions are as follows : number of ASL = 4 , Test time period =200 seconds. Each processor's time interval is as follows :

Failure Control (4 seconds), Recovery Control (5 seconds), Sensor Monitor(4 seconds)

service Name	Service A				
Service List	A_A1	A_A2	A_A3	A_A4	A_A5
ASL	A_A2 A_A3 A_A4 A_A5	A_A3 A_A4 A_A1 A_A5	A_A4 A_A5 A_A1 A_A2	A_A5 A_A3 A_A4 A_A1	A_A1 A_A2 A_A3 A_A2

Table 4.2 Test set ASL (n = 4)

Test Case 2 : ASL (n = 2)

Test conditions are follows: number of ASL = 4, Test time period=200 seconds. Each processor's time interval is as follows :

Failure Control (4 seconds), Recovery Control (5 seconds), Sensor Monitor(4 seconds)

service Name	Service A				
Service List	A_A1	A_A2	A_A3	A_A4	A_A5
ASL	A_A2 A_A3	A_A3 A_A4	A_A4 A_A5	A_A5 A_A3	A_A1 A_A2

Table 4.3 Test set ASL (n = 2)

Test Result Case 1 : ASL (n = 4)

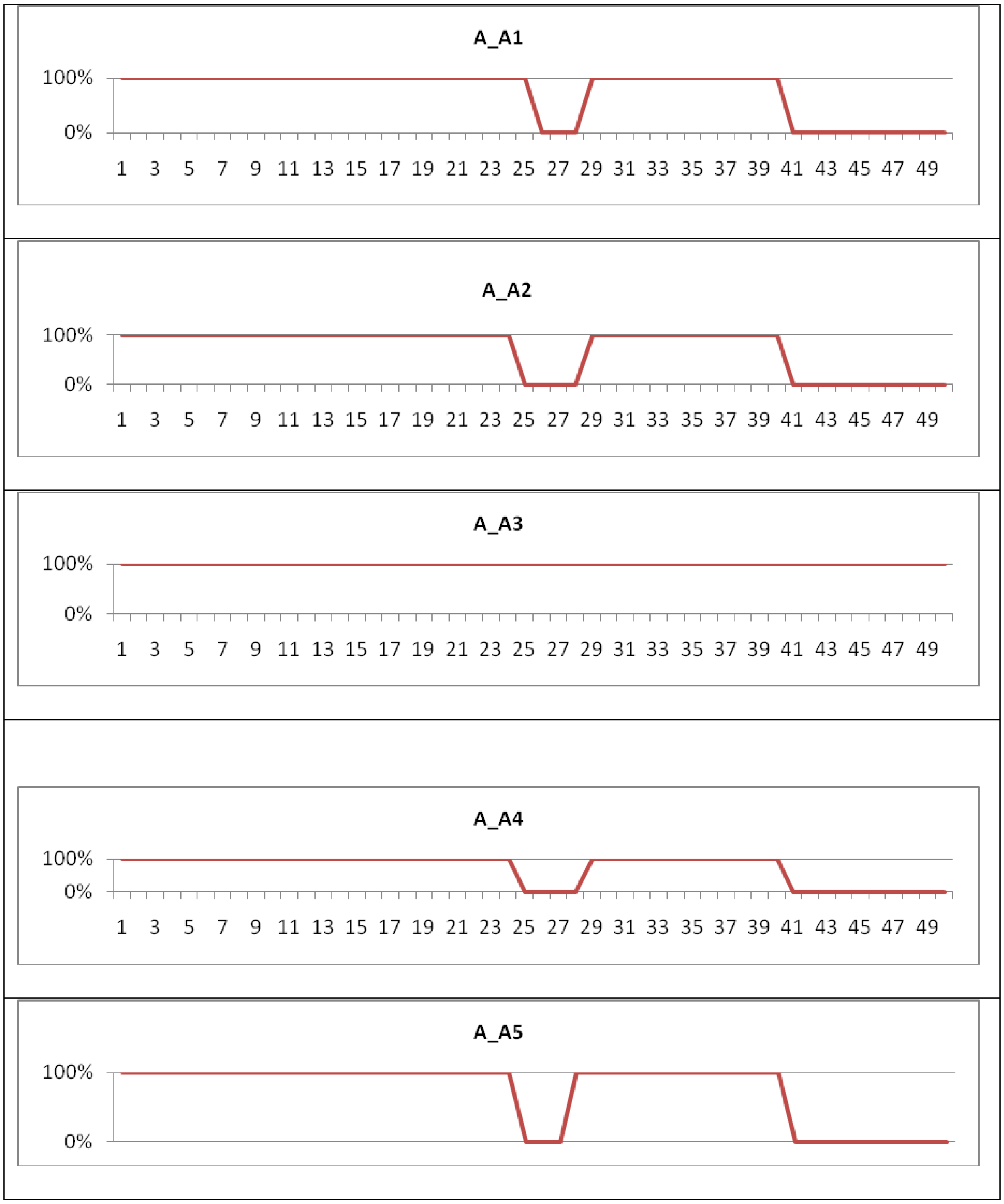


Figure 4.2 Availability graph (n = 4)

Test Result Case 1 : ASL (n = 2)

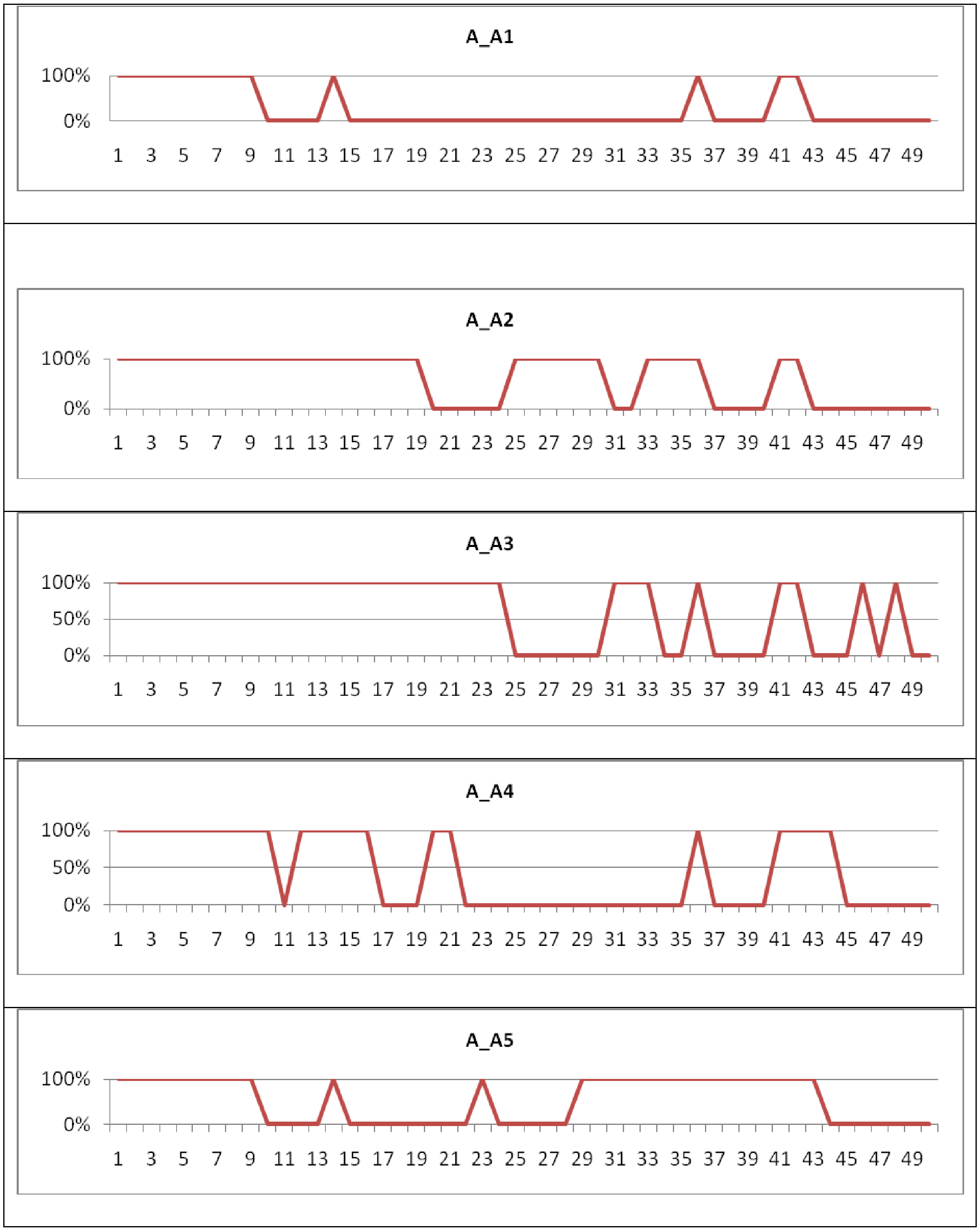


Figure 4.3 Availability graph (n = 2)

4.5.2 Availability Analysis

Case1 : ACL (number of ASL : 4)

service Name	Service A				
Service List	A_A1	A_A2	A_A3	A_A4	A_A5
Up time	17	24	50	29	17
DownTime	33	26	0	21	33
Alternative Service	20	12	0	7	20
Availability (%)	56%	63%	100%	67%	57%

Table 4.4 Availability (n = 4, unit of time : Second)

Case2 : ACL (number of ASL : 2)

service Name	Service A				
Service List	A_A1	A_A2	A_A3	A_A4	A_A5
Up time	13	24	26	20	21
DownTime	37	26	24	30	29
Alternative Service	0	7	6	2	4
Availability (%)	26%	56%	59%	41%	45%

Table 4.5 Availability (n = 2)

From the simulation result shown in Table 4.4 and Table 4.5, in case of ASL(n =2), the average Service availability is 45.4%. In case of ASL(n=4), the average availability is 68.6%. The percentage of availability is increased by 23.2 %, as the number of ASLs doubles from 2 to 4. In other words, a larger number of ASL increase a service. Therefore, number of ASLs is determines the Quality of Service.

4.5.3 UDDI Consistency

To ensure UDDI Consistency, the consistency check monitor has to find inconsistencies and make corrections. In this test, we examine how many mismatches have occurred based on the same test set that was used at the service availability test.

Service Name	ASL = 4	Mismatch Count	match	Service Name	ASL = 2	Mismatch Count	match
A_A1	A_A2	10	30	A_A1	A_A2	6	34
	A_A3	10	30		A_A3	5	35
	A_A4	10	30				
	A_A5	10	30				
A_A2	A_A3	14	16	A_A2	A_A3	0	40
	A_A4	14	16		A_A4	0	40
	A_A1	14	16				
	A_A5	14	16				
A_A3	A_A4	5	35	A_A3	A_A4	8	32
	A_A5	5	35		A_A5	6	34
	A_A1	5	35				
	A_A2	5	35				
A_A4	A_A5	7	23	A_A4	A_A5	8	22
	A_A3	3	27		A_A3	1	39
	A_A4	7	23				
	A_A1	7	23				
A_A5	A_A1	15	15	A_A5	A_A1	6	34
	A_A2	15	15		A_A2	5	35
	A_A3	15	15				
	A_A2	15	15				
Sum		200	480			45	345

Table 4.6 UDDI Consistency check

From the simulation result shown in Table 4.4 and Table 4.5, ASL (n = 4) Consistency = (480 / (200+480)) * 100 = 70.5% , ASL(n=2) , Consistency = (345/ (345+45)) * 100 = 88.4 %

In this experiment, UDDI consistency is decreased by increasing number of ASLs.

4.5.4 Service Availability vs. UDDI Consistency

To increase the service availability, the number of ASLs should be increased. Whereas, to increase UDDI consistency, the number of ASLs should be decreased.

Number of ASL	n= 2	n = 4
Service Availability	45.4%	68.6%
UDDI Consistency	88.4%	70.5%

Table 4.7 Service Availability vs. UDDI Consistency

As shown in Figure 4.4 below, the vertical axis stands for the rate of Service availability and UDDI consistency, wheras the horizontal axis stands for number of ASLs. The graph of service availability increased significantly whereas- the graph of UDDI consistency declined. From the simulation, we have found how ASL affect availability and consistency. It can be seen when n=4 the consistency and availability match. If availability is more important, then n > 4 is preferable whereas if consistency if more important then n = 2 is better.

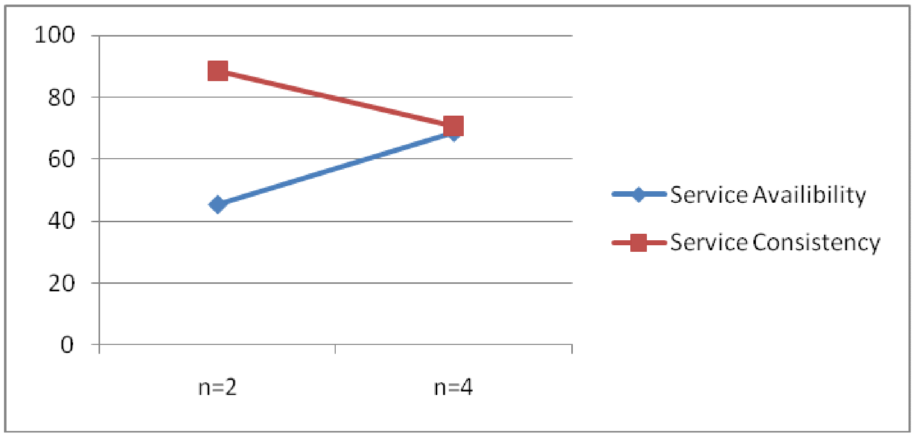


Figure 4.4 Service Availability vs. UDDI Consistency

CHAPTER V

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

The characteristics of heterogeneous sensor devices and various kinds of platforms make them difficult to integrate among WSN applications. To solve this problem, two kinds of different approaches are possible. One is that all manufactures produce powerful sensors, followed by specific open standard Architecture, such as the same type of sensor node, and the same Operating system.

However, this approach needs to consider the cost of powerful sensor nodes. The other approach is to create new types of standard platforms to support all kinds of platforms that are OS-based, VM-based, Middleware-based Architecture. We have proposed the mini-SOA/ESB architecture as an open standard that aims to provide the integration of wireless sensor network applications developed on various different platforms, and we have identified the major requirements of a mini-SOA/ESB, such as Transformability, Flexibility, Security and Quality of Service.

To support QoS, we proposed a modified concept of UDDI structure and its operational algorithms. Furthermore, we simulated a service availability using ASL service select algorithms and consistency monitoring. From the experiment, increasing the number of ASLs affects service availability and UDDI consistency. The proposed mini-SOA/ESB will provide the possibility of integrating wireless sensor network applications as an open standard frame work.

5.2 Future work

We believe that proposed mini-SOA/ESB Architecture will be a basic building block of WSN integration. To make this architecture as an acceptable framework, the requirements are as follows:

First, we need to propose a more specific design consideration of Transformability, Interoperability, Flexibility, Security, and QoS. Second, simulations and operations of mini-SOA/ESB will be implemented on the WSN platforms that are currently used. Finally, to integrate with common SOAs, we will perform feasibility tests for integration..

REFRENCES

[1] F.L Lewis, “Wireless Sensor Networks”,
http://arri.uta.edu/acs/networks/WirelessSensorNetChap04.pdf [last accessed - Jan 10, 2009]

[2] Subhas C. Mukhopadhyay, Anuroop Gaddam and Gourab S. Gupta., “Wireless Sensors for Home Monitoring – A Review”, Recent Patents on Electrical Engineering, 2008,Vol.1,No.1

http://www.bentham.org/eeng/samples/eeng%201-1/Mukhopadhyay.pdf

[3] Laurent Gomez, Annett Laube, Alessandro Sorniotti,. “Design Guidelines for Integration of Wireless Sensor Networks with Enterprise Systems”, Proceedings of the 1st international conference on Moblie Wireless Middleware, Operating Systems, and Applications, 2008, Vol. 278, Article No. 12
http://portal.acm.org/citation.cfm?id=1361507

[4] Borzooo, Bonakdarpour., “Challenges in transformation of existing real-time embedded systems to cyber-physical systems”, Special issue on the Real Time Systems Symposium (RTSS) forum on deeply embedded real-time computing, 2008, Vol.5, Article No. 11
http://portal.acm.org/citation.cfm?id=1366294

[5] Woochul Kang, SangH.Son, “The Design of an Open Data Service Architecture for Cyber-physical Systems”, Special issue on the Real Time Systems Symposium (RTSS) forum on deeply embedded real-time computing, Vol 5, issue 1, No.3, 2008
http://portal.acm.org/citation.cfm?id=1366283.1366286&coll=portal&dl=ACM

[6] Jaco M. Prinsloo, Christian L. Schulz, Derrick G. Kourie, W.H. Morkel Theunissen,Tinus Strauss, Roelf Van Den Heever, Sybrand Grobbelaar.,”A Service Oriented Architecture for Wireless Sensor and Actor Network applications, Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, 2006 Vol.204, pp. 145-154.
http://portal.acm.org/citation.cfm?id=1216278

[7] Setrag Khoshafian, Alan Trefler., Service Oriented Enterprise, Auerbach Publications, 2007, pp 37~41

[8] Yingshu Li, My T.Thai, Weili Wu.,Wireless Sensor Networks and Applications, Springer,2007

[9] Mauri Kuorilehto, Marko Hannikainen, Timo D, Hamalainen.,”A Servey of application distribution in wireless Sensor networks”, EURASIP Journal on Wireless Communications and Networking, Volume 5, issue 5, pp. 774-788, 2005
http://portal.acm.org/citation.cfm?id=1115486.1115500

[10] Hock Beng Lim, Yong Meng Teo, Protik Mukherjee, Vinh The Lam, Weng Fai Wong, Simon See.,"Sensor Grid: Ingernation of Wireless Sensor Networks and Grid", Proceedings of the IEEE Conference on Local Computer Networks (LCN'05), 2005 http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01550845

[11] Wouter Horr´e, Sam Michiels,Nelson Matthys, Wouter Joosen, Pierre Verbaeten., "On the Integration of Sensor Networks and General Purpose IT Infrastructure Proceedings of second international workshop on Middleware for Sensor Networks MidSens'07, November 30, 2007 http://www.cs.kuleuven.be/~wouterh/talks/midsens2007.pdf [last accessed - Jan 10,2009]

[12] Lei Shu, Manfred Hauswirth, Long Cheng, Jian Ma, Vinny Reynolds, Lin Zhang.,"Sharing Worldwide Sensor Network", Proceedings International Symposium on Applications and the Internet (SAINT 2008), pp. 189-192, 2008. http://www.google.com/search?hl=ko&q=Sharing+Worldwide+Sensor+Network&lr= [Last accessed – Jan 2,2009]

[13] Enterprise Service Bus – wikipedia http://en.wikipedia.org/wiki/Enterprise_service_bus [Accessed - Dec 20,2008]

[14] Rufus Credle, Jonathan Adams, Kim Clark, Yun Peng Ge, Hatcher Jeter, Joao Lopes, Samir Nasser, Kailash Peri.,"Patterns:SOA Design Using WebSphere Message Broker and WebSphere ESB", IBM Red Books, July 2007 [SG24-7369-00]

[15] Karl Mayer and Wolfgang Fritssche,"IP-enabled Wireless Sensor Networks and their integration into the Internet ",Proceedings of the first international conference on Integrated internet ad hoc and sensor networks Vol. 138, Article No.5, 2006 http://portal.acm.org/citation.cfm?id=1142687

[16] Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao,"Tiny Web Services: Design and implementation of Interoperable and Evolable Sensor Networks", Proceedings 6th ACM Conference on Embedded Networked Sensor Systems SenSys'08, pp. 253-266, 2008,

[17] Self-Describing Sensor Networks Using a Surrogate Architecture , http://www.icta.ufl.edu/projects/publications/Sensor-platform-paper2.pdf [Last accessed -Jan 10,2009]

[18] Fire alarm system, http://www.fire-monitoring.com/faqs.htm#14 [Last accessed – Feb 5,2009]

[19] NetBeans IDE 6.1 Informaion , http://www.netbeans.org/community/releases/61/ [Last accessed Feb 10,2009]

[20] Implenting SOA with the java EE 5 SDK , http://java.sun.com/developer/technicalArticles/WebServices/soa3/ [Last accessed Feb 11, 2009]

[21] Sharing worldwide Sensor Network, http://lei.shu.deri.googlepages.com/swdmnss2008CameraReady.pdf [Last accessed Feb 15, 2009]

[22] OASIS(Organization for the Advancement of Structured Information Standards), http://www.oasis-open.org/ [Last accessed Mar 31,2009]

[23] SensorBase.org- A Centralized Repository To Slog Sensor Network DATA, http://research.cens.ucla.edu/people/estrin/resources/conferences/2006jun-Chang-Estrin-Yau-Centralized.pdf [Last accessed April 10,2009]

[24] Worldwide Sensor Web Framework Overview, http://www.cms.livjm.ac.uk/pgnet2008/Proceeedings/Papers/2008063.pdf [Last accessed April 10, 2009]

[25] Koutsoukos, X., M. Kushwaha, I. Amundson, S. Neema, andSztipanovits, "OASiS: A Service-Oriented Architecture for Ambient-Aware Sensor Networks", Vanderbilt University, 2007.
http://www.isis.vanderbilt.edu/sites/default/files/LNCS2007.pdf

APPENDIX

Simulation Code List

1. mini_SOA_ESB.java

```
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.*;

public class mini_SOA_ESB {
private function  function = new function();
private int sizeOfCol = 5;
private int sizeOfRow = 5;
private int businessKey = 0;
private int serviceLevel = 0;
public  int timeInterval = 0;

private String[][] ServiceList = new String[sizeOfCol][sizeOfRow];

/* UDDI defination at Sensor Web Domain and Service Provider */
public static Vector<sensorUDDI>A_sensorUDDI =new Vector<sensorUDDI>();
public static Vector<sensorUDDI>W_sensorUDDI =new Vector<sensorUDDI>();

/* All of the published services put into the readyQue */
public static Vector<readyQueue>readyQueueList =new Vector<readyQueue>();

  /* Logging file for the Availibility and Consistency */
public  static PrintWriter  availibility_out;
public  static PrintWriter  consistency_out;

  /* To make logging file for Availibility.txt */

public void createFileAvailbility() {
  try {
      availibility_out = new PrintWriter(new BufferedWriter (new FileWriter("Availibility.txt")));
                } catch (FileNotFoundException e) {
                                System.exit(1);
                } catch (IOException e) {
  }
}

  /* To make logging UDDI consistency Consistency.txt */
public void createFileConsistency() {
  try {
      consistency_out = new PrintWriter(new BufferedWriter
                      (new FileWriter("Consistency.txt")));
                 } catch (FileNotFoundException e) {
                              System.exit(1);
                         } catch (IOException e) {
             }
}

/* To keep pending processors in the Queue */
```

```
public static Vector<suspendQueue>suspendQueueList=new Vector<suspendQueue>();

public static Vector <aMonitor>  aMonitorV  = new Vector<aMonitor>();

public void setAserviceList ( String[] nodeinfo, int col, int row) {
   for( int i = 0; i < row  ;i++ ) {
                ServiceList[col][i] =  nodeinfo[i];
                }
}

   /* Servcies and their Alternative service Setting */
public void setArrayA() {
  int col=0, row = 3; //Number of services
  col = 0;
  String[] AserviceList  = {"A_A1","A_A2","A_A3","A_A4","A_A5"};
  setAserviceList( AserviceList,col, row );

  col = 1;
  String[] AserviceList1 = {"A_A2","A_A3","A_A4","A_A1","A_A5" };
  setAserviceList( AserviceList1,col, row );
  col = 2;
  String[] AserviceList2 = {"A_A3","A_A4","A_A5","A_A1","A_A2"};
  setAserviceList( AserviceList2,col, row );
  col = 3;
  String[] AserviceList3 = {"A_A4","A_A5","A_A3","A_A4","A_A1"};
  setAserviceList( AserviceList3,col, row );

  col = 4;
  String[] AserviceList4 = {"A_A5","A_A1","A_A2","A_A3","A_A2" };
                setAserviceList( AserviceList4,col, row );
                function.Verify_Aservice(ServiceList,sizeOfCol,sizeOfRow );

} // end of Set Array

public void setUDDIA() {

   createFileAvailbility(); //Creat a Logfile for Availbility //
   createFileConsistency(); //Creat a Logfile for Consistency //

   String serviceNameA = "FireMonitorA";
   //Save a service Name //
   sensorUDDI mService = new sensorUDDI();
   sensorUDDI wService = new sensorUDDI();
   //readyQueue readyQueue1 = new readyQueue();
   businessKey = 0;
   //for save Service //
   mService.setUDDI( businessKey,serviceNameA);
   wService.setUDDI( businessKey,"FireMonitorB");
   A_sensorUDDI.add(mService);
   W_sensorUDDI.add(wService);

   sensorUDDI getMainService  = A_sensorUDDI.get(0);
   sensorUDDI getMainServiceW = W_sensorUDDI.get(0);

   Vector<aService> getAservice  = getMainService.getaServiceVector();
   Vector<aService> getAserviceW  = getMainServiceW.getaServiceVector();

   for ( int i = 0; i < sizeOfCol ;i++ ) {
                aService aService1 = new aService();
      aService1.setAservice(1,ServiceList[i][0],0);
      getAservice.add(aService1);
```

```
     aService aService2 = new aService();
     aService2.setAservice(1,ServiceList[i][0],0);
     getAserviceW.add(aService2);

     aService gotAservice = getAservice.get(i);
     Vector<aAltService> getAltService = gotAservice.getaAltServiceVector();

     aService gotAserviceW = getAserviceW.get(i);
     Vector<aAltService>getAltServiceW = gotAserviceW.getaAltServiceVector();

    /* Monitor Vector : Setting for monitoring a Services that published
     * with aMonitorID, aMonitorName, serviceName                    */

     aMonitor aMonitor1 = new aMonitor();
     aMonitor1.setAmonitor(i, ServiceList[i][0],serviceNameA);
     aMonitorV.add(aMonitor1);

     // Ready Queue Setting //
     readyQueue readyQueue1 = new readyQueue();
     serviceLevel = 0;

     readyQueue1.setReadyQueue( serviceLevel,
       businessKey, i,ServiceList[i][0]);
       readyQueueList.add(readyQueue1);

     for(int j = 1; j < sizeOfRow ; j++) {
             aAltService aAltService1 = new aAltService();
                aAltService1.setaAltService2(3,ServiceList[i][j],0);
                getAltService.add(aAltService1);

                aAltService aAltService2 = new aAltService();
                aAltService2.setaAltService2(3,ServiceList[i][j],0);
                getAltServiceW.add(aAltService2);
     }
   }
} //
public void verify_ServiceA () {
   // get first item from vector V_sensorUDDI //

   for( int k = 0; k < A_sensorUDDI.size();k++) {
               sensorUDDI getMainService = A_sensorUDDI.get(k);
               System.out.println ("\n\n service Name " +
                                        getMainService.getServiceName());
   // Type conversion for the get values Ve//
   Vector<aService> getAservice  = getMainService.getaServiceVector();
   for( int i = 0; i < getAservice.size(); i++ ) {
               aService gotAservice = getAservice.get(i);

               Vector<aAltService> getAltServie = gotAservice.getaAltServiceVector();
               System.out.print("Service Name" + gotAservice.getAserviceName());
               for( int j = 0; j < getAltServie.size(); j++ ) {
                  aAltService gotAltservice = getAltServie.get(j);
                //gotAltservice.getAserviceName();
                          System.out.print(" -- " + gotAltservice.getAltserviceName());
               }
                System.out.println(" ");
             } //for
  } //for
}//verify_ServiceA ()
```

```
public void verify_ServiceW () {
// get first item from vector V_sensorUDDI //
   for( int k = 0; k < W_sensorUDDI.size();k++) {
                sensorUDDI getMainService = W_sensorUDDI.get(k);
                System.out.println ("\n W service  " + getMainService.getServiceName());
                // Type conversion for the get values Ve//
                Vector<aService> getAservice  = getMainService.getaServiceVector();
                for( int i = 0; i < getAservice.size(); i++ ) {
                            aService gotAservice = getAservice.get(i);
                            //gotAservice.getAserviceName();
                            //System.out.println(" ");
                            System.out.print  ("\n aService  " + gotAservice.getAserviceName());
                            Vector<aAltService>getAltServie=gotAservice.getaAltServiceVector();
                            for( int j = 0; j < getAltServie.size(); j++ ) {
                               aAltService gotAltservice = getAltServie.get(j);
                              //gotAltservice.getAserviceName();
                                        System.out.print(" -- " + gotAltservice.getAltserviceName());
                                       }//for
                            }//for
   } //for
}//verify_ServiceW ()

public void init_Array () {
   for ( int i = 0; i < sizeOfCol; i ++)
      for ( int j = 0; j < sizeOfRow ; j ++)
                  ServiceList[i][j] = " ";
}

public void verify_ReadyQ() {
   System.out.println("\n");
   for( int k = 0; k < readyQueueList.size();k++) {
      readyQueue getReadyQvalue = readyQueueList.get(k);
                System.out.println(" ReadyQ  " + getReadyQvalue.getServiceName());
   }
}

public void verify_SuspendQ() {
   System.out.println("mini_SOA_ESB: 326 SUSPEND QUEUE  \n");
   for( int k = 0; k < suspendQueueList.size();k++) {
      suspendQueue getReadyQvalue = suspendQueueList.get(k);
                System.out.println(" Suspend Q  " + getReadyQvalue.getServiceName());
  }
}

public static void  main(String[] args) {
   mini_SOA_ESB ESB  = new mini_SOA_ESB();
   failureRecovery failure = new  failureRecovery();
   consistencyCheck consistencyCheck1 = new consistencyCheck ();

                /* Initialize data */
   ESB.init_Array();
   ESB.setArrayA();
   ESB.setUDDIA();
   ESB.verify_ServiceA();
   ESB.verify_ServiceW();
   ESB.verify_ReadyQ();
   failure.failureRecovery1();
   consistencyCheck1.consistencyCheck1();

   ESB.verify_SuspendQ();
}//main(String[] args)
```

```
}//End of Program
```

2. suspendQueue.java

```
public class suspendQueue {
   private int businessKey = 0;
   private int serviceID = 0;
   private String serviceName = " ";
   public void setSuspendQueue (int businessKey,int serviceID,String ServiceName){
                this.businessKey = businessKey;
      this.serviceID = serviceID;
                this.serviceName = ServiceName;
   }

   public int getBusienssKey() {
                return businessKey;
   }

   public int getServiceID( ){
      return serviceID;
   }

   public String getServiceName() {
      return serviceName;
   }
}//SuspendQueue
```

3. sensorUDDI.java

```
import java.util.*;
public class sensorUDDI{
private int                  businessKey          =0 ;//AREA-XXXX-XXXX
   private int     authenticationStep = 0;// 0, 1, 2
   private String  discoveryURL              = "WWW.Sensor.org";
   private String  serviceName = " ";
   private String  serviceCreationTime = " " ;
   private String  effectiveServcieDateTime = " ";
   private String  lastConstistencyCheck = " ";
   //-- Operation --//
   public void setUDDI(int businessKey, String serviceName ) {
                   this.businessKey = businessKey;
                            this.serviceName = serviceName;
     }

   public int getBusinessKey( ) {
                return businessKey;
                }

   public String getServiceName () {
                return serviceName;
   }

   public Vector<aService> aServiceVector = new Vector<aService>();

   public Vector<aService> getaServiceVector() {
                return aServiceVector;
   }
}
```

4. readyQueue.java

```
public class readyQueue {
   private int businessKey =0;
   private int serviceID = 0;
   private int serviceLevel = 0;
   private String serviceName = " ";
   public void setReadyQueue (int businessKey,int serviceLevel,
                                           int serviceID,String ServiceName){
               this.serviceLevel = serviceLevel;
      this.businessKey = businessKey;
               this.serviceID = serviceID;
               this.serviceName = ServiceName;
   }
   public int getServiceLevel () {
               return serviceLevel;
   }
   public int getBusinessKey () {
               return businessKey;
   }
   public int getServiceID( ){
      return serviceID;
   }
   public String getServiceName() {
      return serviceName;
   }
}//class readyQueue
```

5. finction.java

```
import java.io.*;
import java.util.Date;
import java.util.Timer;
import java.util.TimerTask;
import java.util.Vector;
import java.io.BufferedWriter;
import java.text.DateFormat;
import java.text.SimpleDateFormat;

public class function  {
  //For reporting availibility and consistency //
             public void verify_Monitor () {
               for ( int k = 0; k < mini_SOA_ESB.aMonitorV.size();k++) {
                 aMonitor getMonitor = mini_SOA_ESB.aMonitorV.get(k);
                 Vector<aAltMonitor> aAltMonitorVector =
                           getMonitor.getAltMonitorVector();
                           for ( int i = 0; i < aAltMonitorVector.size(); i++ ) {

                           aAltMonitor gotAltMonitor = aAltMonitorVector.get(i);
                           //Logging to the file "
                           mini_SOA_ESB.availibility_out.println
                           ( " Servie: " + getMonitor.getMonitorName() +
                             " Interval:  " + gotAltMonitor.getTimeInterval()+
                             " Status  :  " + gotAltMonitor.serviceStatus()  +
                             " ASL Status :  " + gotAltMonitor. altServiceStatus());
                 }
               }
              }
             // Verify service for ServiceA //
```

```
            public void verify_ServiceA () {
                        // get first item from vector V_sensorUDDI //
                        for ( int k = 0; k < mini_SOA_ESB.A_sensorUDDI.size();k++) {
                        sensorUDDI getMainService = mini_SOA_ESB.A_sensorUDDI.get(k);
                        System.out.println ("\n service A " +
                                                getMainService.getServiceName());
                        // Type conversion for the get values Ve//
                        Vector<aService> getAservice  = getMainService.getaServiceVector();
                        for ( int i = 0; i < getAservice.size(); i++ ) {
                          aService gotAservice = getAservice.get(i);
                                  //gotAservice.getAserviceName();
                                  //System.out.println(" ");
                          mini_SOA_ESB.consistency_out.print("\n  Service A " +
                                                      gotAservice.getAserviceName());
                                  System.out.print  ("("+ gotAservice.getAserviceMishmatch()+")");
                                  Vector<aAltService> getAltServie =  gotAservice.getaAltServiceVector();
                                    for ( int j = 0; j < getAltServie.size(); j++ ) {
                                            aAltService gotAltservice = getAltServie.get(j);
                                            //gotAltservice.getAserviceName();
                                            mini_SOA_ESB.consistency_out.print(" * * " +
                                                          gotAltservice.getAltserviceName());
                                            mini_SOA_ESB.consistency_out.print("("+
                                                          gotAltservice.getAltserviceMismatch ()+")");
                                          }
               }
                        mini_SOA_ESB.consistency_out.println("  " );
              }
             }

             public void verify_ServiceW () {
                        // get first item from vector V_sensorUDDI //
                        for ( int k = 0; k < mini_SOA_ESB.W_sensorUDDI.size();k++) {
                        sensorUDDI getMainService = mini_SOA_ESB.W_sensorUDDI.get(k);
                        System.out.println ("W service  " + getMainService.getServiceName());
                        // Type conversion for the get values Ve//
                        Vector<aService> getAservice  = getMainService.getaServiceVector();
                        for ( int i = 0; i < getAservice.size(); i++ ) {
                          aService gotAservice = getAservice.get(i);
                                  //gotAservice.getAserviceName();
                                  //System.out.println(" ");
                                  System.out.print  ("\n ====== " + gotAservice.getAserviceName());
                                  Vector<aAltService> getAltServie =
                 gotAservice.getaAltServiceVector();
                                    for ( int j = 0; j < getAltServie.size(); j++ ) {
                                            aAltService gotAltservice = getAltServie.get(j);
                                            //gotAltservice.getAserviceName();
                                            System.out.print(" -- " +
                      gotAltservice.getAltserviceName());
                                          }
               }
              }
             }
    public String getDateTime() {
               DateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd  HH:mm:ss");
               Date date = new Date();
               return dateFormat.format(date);

            }//getDataTime

    //This method is to check published service //
    public int confirmAltService(String aService) {
            int readyQ = 0;
        // Get first element of the Vector A_sensorUDDI //

                    sensorUDDI getMainService = mini_SOA_ESB.A_sensorUDDI.get(0);
                            Vector<aService> getAservice  = getMainService.getaServiceVector();
                            aService gotAservice = getAservice.get(0);
                            readyQ = readyQchcek(gotAservice.getAserviceName());

                return readyQ;
            }//getDataTime

    // Checking for Reday Queue //
    public int readyQchcek(String aService) {
                int i = 0;
                int qsize = mini_SOA_ESB.readyQueueList.size();
                            for ( int q = 0; q < qsize ;q++) {
                             readyQueue getReadyQvalue = mini_SOA_ESB.readyQueueList.get(q);

                            if ( getReadyQvalue.getServiceName() == aService )
                                        i = 1;
                            }
                return i;
    }
        //Checking for the suspend Queue //
    public int suspendQueueCheck(String aService) {
                int j = 0;
                int qsize = mini_SOA_ESB.suspendQueueList.size();
                            for ( int q = 0; q < qsize ;q++) {
                             suspendQueue getSuspendQvalue =
                mini_SOA_ESB.suspendQueueList.get(q);

                            if ( getSuspendQvalue.getServiceName() == aService )
                                        j = 1;
                            }
                return j;
    }

    public void Verify_Aservice(String[][] Array, int sizeOfCol, int sizeOfRow ){
                for (int i = 0; i < sizeOfCol; i ++ ) {
                            System.out.println("\n" );
                            for (int j = 0; j < sizeOfRow ; j ++) {
                                        System.out.print( "  "+ Array[i][j]);
                            }
        }
    }// Veryfy_Aservice

    public class timeDelay {
                 Timer timer;
                 public timeDelay(int seconds) {
                    timer = new Timer();
                    timer.schedule(new RemindTask(), seconds * 1000);
                 }
                 class RemindTask extends TimerTask {
                   public void run() {
                     System.out.println("Time's up!");
                     timer.cancel(); //Terminate the timer thread
                   }
                 }
    }
} // End of function
```

6. failureRecovery.java

```
import java.util.Random;
import java.util.Vector;

public class failureRecovery extends mini_SOA_ESB  {

   private sensorUDDI getMainService =  new sensorUDDI();
   private function  functionx = new function();
   public void failureRecovery1( )   {
      new FailureControl (200  ,4).start();  //  3 second
      new RecoveryContorl(200  ,5).start();  //
      new SensorMonitor  (200  ,4).start();
 // new RecoveryContorl (20 , 5).start();
   }
              /*--------------------------------------------------------------*/
   public class FailureControl extends Thread {

   private int delayTime; // delay time second
   private int executionTime;
   private int numberOfLoops;
   private int businessKey =0;
   private int serviceID = 0;
   private String serviceName = " ";

   public  FailureControl(int executionTime, int delayTime) {
      this.delayTime = delayTime;
      this.executionTime = executionTime;
      numberOfLoops = this.executionTime / this.delayTime;
      }

   public void run() {
      try {
              for( int i = 0 ; i < numberOfLoops; i ++ ) {
                  //sensorUDDI newUDDI = new sensorUDDI();
    sleep(delayTime  *  1000 );  // wait until next time
    functionx.getDateTime();   // Time display on the monitor
    RandomFailureGenerte();    // Random number generate
    System.out.println("0-FailureControl  :" + functionx.getDateTime()+"\n");
}
} catch (InterruptedException e) {
   return; // end this thread;
  }
}

//---Random failure generate ----//

   public synchronized void  RandomFailureGenerte( )   {
   // count element of the Ready Queue //
  Random magicNumber = new Random();
  int randomInt=0;
  int businessKey = 0;

  int Qsize =  readyQueueList.size();
  System.out.println(" Size of the array queue" + Qsize );

      if ( Qsize > 0 )
         {
        //Random number generate depending on the array //
 for (int idx = 0; idx <  1; ++idx){ // Number of magic number
 randomInt = magicNumber.nextInt(Qsize);
 System.out.println("Generated : " + randomInt);
 }
//readyQueue value to Suspend value

     readyQueue getReadyQvalue = readyQueueList.get(randomInt);

   //1- get businessKey , serviceID , serviceName from ReadyQ to move SuspendQ//
    businessKey =  getReadyQvalue.getBusinessKey();
    serviceID =  getReadyQvalue.getServiceID();
    serviceName = getReadyQvalue.getServiceName();

    System.out.println("ReadyQ's Element  : " + serviceName);


   suspendQueue suspendClass = new suspendQueue();
   suspendClass.setSuspendQueue( businessKey,
                                        serviceID  ,
                                        serviceName );
 //2- add suspendQ and delete from ready Q//
      suspendQueueList.add(suspendClass);
      readyQueueList.remove(randomInt);



      if (businessKey == 0 )
           getMainService = A_sensorUDDI.get(0);
      else getMainService = A_sensorUDDI.get(0);

     //Get Value from Vector //
   System.out.println ("A Main service  " + getMainService.getServiceName());
   Vector<aService>    getAservice  = getMainService.getaServiceVector();
   // a services move to the end of the alternative Services //

   for ( int k = 0; k < getAservice.size(); k++ ) {
        aService gotAservice = getAservice.get(k);

       Vector<aAltService> getAltService = gotAservice.getaAltServiceVector();

       //3- Compare Ready Q value and Service Name //
    if (gotAservice.getAserviceName().equals(serviceName)) {

       //4- Move aService to the end of the ASL list with it's values//
       aAltService aAltService1 = new aAltService();
                 aAltService1.setaAltService2( gotAservice.getAserviceID(),
                                                    gotAservice.getAserviceName(),
                                                    gotAservice.getAserviceMishmatch());
                 getAltService.add(aAltService1);

                 functionx.verify_ServiceA();
                 int setService =0; //If one service is selected no more loop required //

                 //5- Choose one service among the A Service and move to the A Service //
                 for( int j = 0; j < getAltService.size()   ;j ++) {
               aAltService gotAltService = getAltService.get(j);

               for ( int i = 0; i < readyQueueList.size();i++) {
                      getReadyQvalue = readyQueueList.get(i);

                      // 6- Check ASL List and pick one of available service //
```

```
                    if ( gotAltService.getAltserviceName() ==
                            getReadyQvalue.getServiceName()  ) {
                           // 7- Checking service selected //

                             if (setService == 0 ) {
                                // 8- Remove from the ASL //
                                getAltService.remove(j);

                                // 9- backup the value to keep ASL //
                                Vector<aAltService> getAltTemp =
                                         gotAservice.getaAltServiceVector();

                                aService aServiceAdd = new aService();

                                //10 - Set A services among ASL List at location K //
                                aServiceAdd.setAservice
                                    ( serviceID,gotAltService.getAltserviceName(),
                                              gotAltService.getAltserviceMismatch ());
                                getAservice.set(k,aServiceAdd);

                                aService gotAserviceF = getAservice.get(k);
                            Vector<aAltService> getAltServiceF =
                                          gotAserviceF.getaAltServiceVector();

                            //11- Depending on change services //
                                for ( int t=0;t < getAltTemp.size(); t++){

                                         aAltService gotAltTemp= getAltTemp.get(t);

                                         aAltService aAltService2 = new aAltService();
                                             aAltService2.setaAltService2
                                                ( gotAltTemp.getAltserviceID(),
                            gotAltTemp.getAltserviceName(),
                                   gotAltTemp.getAltserviceMismatch ());
                                             getAltServiceF.add(aAltService2);
                                }

                                setService = 1;
                               /* AFTER CHANGING */
                               System.out.println("\n===after changing\n");

                                             functionx.verify_ServiceA();
                     }
             } //for
            } //Unitil find new services
      }//for
   }
  }//for
} //if

  }//random failure generate

  } //FailureControl
  /*-----------------------------------------------------------*/
  public class RecoveryContorl extends Thread {
  private int businessKey =0;
  private int serviceID = 0;
  private String serviceName = " ";
  private function  function = new function();
  private int delayTime; // delay time second
  private int executionTime;
```

```
   private int numberOfLoops;

   public RecoveryContorl(int executionTime, int delayTime) {
               this.delayTime = delayTime;
     this.executionTime = executionTime;
               numberOfLoops = this.executionTime / this.delayTime;
   }
    public void run() {
       try {
              for ( int i = 0 ; i < numberOfLoops; i ++ ) {
            sleep(delayTime*1000); // wait until next time
     function.getDateTime();
     RandomFailureRecovery( );  // Random number generate
     System.out.println("2-Recovery control  :" + function.getDateTime()+"\n");
              }
} catch (InterruptedException e) {
return; // end this thread;
   }
}
//--- Failure Recovery Part ---/
public synchronized void  RandomFailureRecovery( )   {

   Random magicNumber = new Random();
   int randomInt=0;
   int Qsize = suspendQueueList.size();
   System.out.println(" Size of the Suspend queue" + Qsize );
   if ( Qsize > 0 )
     for (int idx = 0; idx <  1; ++idx){ // Number of magic number
     randomInt = magicNumber.nextInt(Qsize);
     System.out.println("Generated : " + randomInt);

   suspendQueue
                      getSuspendQvalue =  suspendQueueList.get(randomInt);
                        //get businessKey , serviceID , serviceName from ReadyQ//

   businessKey = getSuspendQvalue.getBusienssKey();
   serviceID = getSuspendQvalue.getServiceID();
   serviceName = getSuspendQvalue.getServiceName();

   System.out.println("Suspend Q's Element : " + serviceName);
 //copy SuspendQ's --> ReadyQ's value //
   readyQueue readyClass = new readyQueue();
   readyClass.setReadyQueue(  businessKey,
                                                    0, //Service Level
                                     serviceID  ,
                                     serviceName );
        // add suspendQ and delete from ready Q//
                 readyQueueList.add(readyClass);
                 suspendQueueList.remove(randomInt);
     }
    }
   } // RecoveryContorl
/*------------------------------------------------------------*/
public class SensorMonitor extends Thread {
   private function  function = new function();
   private int delayTime; // delay time second
   private int executionTime;
   private int numberOfLoops;
   private int serviceStatus = 0;
   private int altServiceStatus = 0;
   private String aService = " ";
```

```
  public SensorMonitor(int executionTime, int delayTime) {
               this.delayTime = delayTime;
     this.executionTime = executionTime;
               numberOfLoops = this.executionTime / this.delayTime;
  }

  public void run() {
      try {
              for ( int i = 0 ; i < numberOfLoops; i ++ ) {
          sleep(delayTime*1000 ); // Delay time interval for the processor
    function.getDateTime();
    System.out.println("3-SensorMonitor :" + function.getDateTime()+"\n");
    System.out.println("Suspend Queue \n");

    timeInterval = timeInterval + delayTime;

      // For the all services in the aMonitor Vector //
      /*if service is not in the ReadyQ it means there is failure
        *This case need to find alternative service from ASL
        *If A Service Replaced by another servcie then that service
        *not failure in this case Setting by altServiceStatus = 1 ;*/

    for ( int m = 0; m < aMonitorV.size(); m++ ) {
              /*1- Get a vector from aMonitor Vector */
                              aMonitor getMonitorValue = aMonitorV.get(m);
                              serviceStatus = 0; //Read Q size Setting
                              for ( int q = 0; q <readyQueueList.size()  ;q++) {
                             readyQueue getReadyQvalue = readyQueueList.get(q);

                                          if ( getReadyQvalue.getServiceName() ==
                                                      getMonitorValue.getMonitorName())
                                          {
                                                      serviceStatus = 1;  //Service is match
                                          }
                              }//End For

                                altServiceStatus = 0;
                              //To confirm a Service is running or not //
                                altServiceStatus = function.confirmAltService
                                            (getMonitorValue.getServiceName());


                Vector<aAltMonitor> getAltMonitor = getMonitorValue.getAltMonitorVector();
                aAltMonitor aAltMonitor2 = new aAltMonitor();
                aAltMonitor2.setAltMonitor( timeInterval ,serviceStatus,altServiceStatus);
                getAltMonitor.add(aAltMonitor2);

                functionx.verify_Monitor(); //Verify Monitor

              } //Checking aMonitor
              }
} catch (InterruptedException e) {
  return; // end this thread;
                                    }
                                  }
  } //Sensor Monitor

} //End of class
```

7. sensorUDDI.java

```
import java.util.Vector;

public class consistencyCheck extends mini_SOA_ESB {
   public void consistencyCheck1( )   {

                new ConsistencyContorl ( 200, 5).start();  //
   }
   // compare between UDDI structure //
   public class ConsistencyContorl extends Thread {
   private function  function = new function();
   private int delayTime; // delay time second
   private int executionTime;
   private int numberOfLoops;
   private int MismatchCount =0;

   public ConsistencyContorl(int executionTime, int delayTime) {
                this.delayTime = delayTime;
                this.executionTime = executionTime;
                numberOfLoops = this.executionTime / this.delayTime;
   }

   public void run() {
                try {
                   for( int i = 0 ; i < numberOfLoops; i ++ ) {
                      sleep(delayTime*1000); // wait until next time
                      function.getDateTime();
                      /*  Compare A_sensorUDDI <> W_sensorUDDI  */
                       compareSensorUDDI_A_W();

                      System.out.println("UDDI consistency Check " +
                                              function.getDateTime()+"\n");
                      }
                } catch (InterruptedException e) {

                  return; // end this thread;
                }
   } //run

   public void compareSensorUDDI_A_W() {
                        // First, compare whole things  //

               System.out.println ("=== First time compare VALUE OF W");
               function.verify_ServiceA();
               function.verify_ServiceW();
                 //1-get a value from the Vector A_sensorUDDI(0);
               sensorUDDI getMainService_A = mini_SOA_ESB.A_sensorUDDI.get(0);
               sensorUDDI getMainService_W = mini_SOA_ESB.W_sensorUDDI.get(0);

                 //2-get A Service from the a Service Vector //
                         Vector<aService>getAservice_A=getMainService_A.getaServiceVector();
                         Vector<aService>getAservice_W=getMainService_W.getaServiceVector();
                         int sizeOfUDDI_A =  getAservice_A.size(); //get size of UDDI_A.
                         int sizeOfUDDI_W =  getAservice_W.size(); //get size of UDDI_W.

                         //3-get A service //
                         for ( int i = 0; i < sizeOfUDDI_A ;i++ ) {
                                  aService gotAservice_A = getAservice_A.get(i);
                                  aService gotAservice_W = getAservice_W.get(i);
```

```
                                        // gotAservice_A = gotAservice_W; // copy all of the UDDI value.
                                        // found Mismatch case found increase the Mishmatch Mismatch checking
                                        //   to the A service
                      // 4- If mismatch values then add 1 to mismatch count and rewrite //

                                        // 5- Chceking ASL List and rewrite the ASL //
                                        Vector<aAltService> getAltServie_A =  gotAservice_A.getaAltServiceVector();
                                        Vector<aAltService> getAltServie_W = gotAservice_W.getaAltServiceVector();
                                        Vector<aAltService> getAltTemp =
                                                    gotAservice_A.getaAltServiceVector();

                                        if ( gotAservice_A.getAserviceName() !=
                                                     gotAservice_W.getAserviceName()) {

                                                     aService aServiceAdd = new aService(); // New object

                               aServiceAdd.setAservice
                                 ( gotAservice_A.getAserviceID(),
                                  gotAservice_A.getAserviceName(),
                                     gotAservice_A.getAserviceMishmatch()+1);

                               getAservice_A.set(i,aServiceAdd);
                               aService gotAserviceF = getAservice_A.get(i); //Get A Service//
                               Vector<aAltService> getAltServiceF =
                                     gotAserviceF.getaAltServiceVector();

                          //6 - Recovery of ASL //
                             for ( int t=0;t < getAltTemp.size(); t++){
                               aAltService gotAltTemp= getAltServie_A.get(t);
                               aAltService gotAltW    = getAltServie_W.get(t);

                               if (gotAltTemp.getAltserviceName()
                                        != gotAltW.getAltserviceName())
                                          MismatchCount=(gotAltTemp.getAltserviceMismatch()+1);
                               else  MismatchCount=(gotAltTemp.getAltserviceMismatch());


                               aAltService aAltService2 = new aAltService();
                               aAltService2.setaAltService2( gotAltTemp.getAltserviceID(),
                                                             gotAltTemp.getAltserviceName(),
                                                             MismatchCount);

                                         getAltServiceF.add(aAltService2);

                                        function.verify_ServiceA();
                                        function.verify_ServiceW();
                               }
                                        }
                            }//for 3-get A Service
                 }//Compare UDDI compareSensorUDDI_A_W();
   //----Compare UDDI CompareSernsorUDDI_B_W(); ----//
   } //Thread
}//Mini-SOA/ESB
```

8. Service.java

```
import java.util.Vector;
public class aService {

              public Vector<aAltService> aAltServiceVector = new Vector<aAltService>();

              public Vector<aAltService> getaAltServiceVector() {
                  return aAltServiceVector;
              }

              private int aServiceID = 0;
              private String aServiceName = " ";
              private int aServiceMismatch = 0;

  public int getAserviceID( ) {
                      return aServiceID;
              }
              public String getAserviceName() {

                      return aServiceName;
              }

              public int getAserviceMishmatch() {
                      return aServiceMismatch;
              }

              public void setAservice(int aServiceID, String aServiceName,
                                              int aServiceMismatch ) {
                              this.aServiceID = aServiceID;
                              this.aServiceName = aServiceName;
                              this.aServiceMismatch = aServiceMismatch;
                      }
  } //a Sservice
```

9. aMonitor.java

```
import java.util.Vector;
// This calss for the logging values that
public class aMonitor {

                public Vector<aAltMonitor> aAltMonitorVector = new Vector<aAltMonitor>();

                public Vector<aAltMonitor> getAltMonitorVector() {
                    return aAltMonitorVector;
                }

                private String serviceName = " ";
                private int     aMonitorID = 0;
                private String aMonitorName = " ";


      public int getMonitorID( ) {
                        return aMonitorID;
                }

                public String getMonitorName() {
                        return aMonitorName;
                }

                public String getServiceName() {
                        return serviceName;
                }

                // altServiceTime is total time of alterNative Service //
                public void setAmonitor(int aMonitorID, String aMonitorName,
                                                String serviceName ) {
                                this.aMonitorID = aMonitorID;
                                this.aMonitorName = aMonitorName;
                                this.serviceName = serviceName;
```

```
                         }
  } //a Sservice
```

10. aMonitor.java

```
import java.util.Vector;
public class aAltService {
                private int aAltServiceID = 0;
                         private String aAltServiceName = " ";
                         private int aServiceMismatch = 0;
                         public void setaAltService2(int aServiceID, String aServiceName,
                                              int aServiceMismatch) {
                           this.aAltServiceID = aServiceID;
                                    this.aAltServiceName = aServiceName;
                                    this.aServiceMismatch =  aServiceMismatch;
                         }

                         public String getAltserviceName( ) {
                                              return  this.aAltServiceName ;
                         }

                         public int getAltserviceID( ) {
                                    return aAltServiceID;
                         }

                         public int getAltserviceMismatch () {

                                    return aServiceMismatch;
                         }
} // aAltService
```

11. aAltMonitor.java

```
 public class aAltMonitor {
    private int timeInterVal = 0;    //Time interval
    private int serviceStatus = 0;   //
    private int altServcieStatus = 0;

  public void setAltMonitor(int timeInterVal,
                              int serviceStatus,
                              int altServiceStatus ){

             this.timeInterVal = timeInterVal;
     this.serviceStatus =  serviceStatus;
     this.altServcieStatus = altServiceStatus;
  }

  public int  getTimeInterval( ) {
     return  this.timeInterVal ;
  }

  public int serviceStatus( ) {
             return serviceStatus;
  }

  public int altServiceStatus( ) {
             return altServcieStatus;
  }
} //End of Class aAltMonitor
```

VITA

JONGYEOP KIM

Candidate for the Degree of

Master of Science

Thesis: MINI-SOA/ ESB DESIGN GUIDELINES AND SIMULATION FOR WIRELESS SENSOR NETWORK

Major Field: Computer Science

Biographical:

Personal Data: Born in Taean, South Korea, On August 3, 1965.
Education:
Received Bachelor of Science degree in Computer Science from Korea National Open University, Seoul, Korea, Feb 1996. Received Master of Science degree in Computer Science from Oklahoma State University, May 2009.

Experience: Operated and maintained the information systems on a regular basis to be available to users of the information systems. Administered database on the military register record and the reserve army. Managed server systems and network systems to efficiently operate the information systems of conscription administration, Developed and deployed programs and application tools used for the information systems of conscription administration.

Professional : 1988 – Present , Officer, Information Management Division, Military Manpower Administration, Korea. As an public officer, I managed various projects and participated in enhancing the information systems.

Major projects: A project for rebuilding the home page in 2006, A project for rebuilding the information systems of conscription administration in 2002, A project for migration of the military register record. (2000-2001, 1996-1997), A project for amending a program used for managing the reserve army against Y2K issue in 1999, A project for introducing a new information system of conscription administration in 1999.

Name: JONGYEOP KIM Date of Degree: May, 2009

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: MINI-SOA/ ESB DESIGN GUIDELINE AND SIMULATION FOR WIRELESS SENSOR NETWORKS

Pages in Study: 71 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

One of the major challenges in wireless sensor networks (WSNs) is the difficulty of efficient integration with other WSN application domains. This is because there is no open standard framework to support heterogeneous types of sensors that are produced by many sensor manufacturers.

In order to integrate different domain services, Service Oriented Architecture (SOA) /Enterprise Service Bus (ESB) has been widely used as an open standard for providing location transparency and segregation. In this thesis, we propose mini-SOA/ESB Architecture for the integration of wireless sensor networks. However, previous work on SOA/ESB has focused on large scale Enterprise service level integration, and is not adaptable to the WSN domains because of limited hardware and software capabilities.

Sharing sensor data requires an open standard prototype to support various kinds of sensor applications composed of heterogeneous sensor nodes. This standard prototype can be applied to any application, such as OS-based architecture, VM-based architecture, Middleware architecture, and Stand-alone protocols.

To address the issue, this thesis presents design considerations and a new model, which we call mini-SOA/ESB for WSNs, as an open standard. We believe that the proposed Architecture will be a basic building block for the integration of WSNs.

ADVISER'S APPROVAL: Johnson P. Thomas