WSN Setup by means of Software Agents

Master’s Thesis in Embedded and Intelligent Systems

Sha Mao Xuan & Wang Xi Tao & Zuo Shu
WSN Setup by means of Software Agents

School of Information Science, Computer and Electrical Engineering
Halmstad University
Box 823, S-301 18 Halmstad, Sweden

September 2011
Description of cover page picture/figure: A typical wireless sensor networks structure
Preface

This Master Thesis in Embedded and Intelligent Systems has been conducted at the School of Information Science, Computer and Electrical Engineering in Halmstad University, as part of the Master degree programme. In particular, we would like to express our sincere gratitude to our supervisor, Edison Pignaton de Freitas for providing us the opportunity to work in this project and his guidance and help throughout this project.

Furthermore, we are grateful to our families and friends who always support and encourage us.

Sha Mao Xuan & Wang Xi Tao & Zuo Shu
Halmstad University, September 2011
WSN Setup by means of Software Agents
WSN Setup by means of Software Agents

Abstract

A significant challenge in the Wireless Sensor Networks (WSN) research field is to find flexible and energy efficient strategies to perform the network setup and configuration to accomplish specified sensing missions. This thesis presents an approach which uses mobile agents to disseminate and allocate sensing missions to the sensor nodes. The addressed problem refers to the selection of appropriate nodes to perform the sensing mission, by using a decentralized approach supported by mobile software agents. Traditional approaches to deal with WSN setup use the pre-planned strategies, which are deliberately modelled, designed and tuned before the network deployment, and thus are not flexible. This thesis presents an alternative approach based on Belief Desire Intention-model agents using JASON, instead of traditional approaches. Simulation results provides evidences that this approach can achieve the goals of a sensing mission setup by decisions autonomously taken by the sensor node, diminishing then the need for communication among the sensor nodes, hence saving energy resources.
# Contents

PREFACE ................................................................................................................................. 1
ABSTRACT ................................................................................................................................. 3
CONTENTS ................................................................................................................................. 4
LIST OF FIGURES .................................................................................................................. 6
LIST OF TABLES ...................................................................................................................... 8
LISTING LIST ........................................................................................................................... 9
LIST OF ABBREVIATIONS AND ACRONYMS ..................................................................... 10

1 INTRODUCTION .................................................................................................................. 12
  1.1 TECHNOLOGY AREA ....................................................................................................... 14
  1.2 PROJECT OVERVIEW ..................................................................................................... 15
  1.3 PROBLEM STATEMENT ................................................................................................. 15
  1.4 THESIS GOALS AND EXPECTED RESULTS ............................................................... 15
  1.5 THESIS STRUCTURE ................................................................................................. 16

2 WIRELESS SENSOR NETWORK ......................................................................................... 17
  2.1 INTRODUCTION OF WSN ......................................................................................... 17
  2.2 THE KEY TECHNOLOGY OF WSN ........................................................................... 18
  2.3 WSN APPLICATIONS ................................................................................................... 20
  2.4 EVOLUTION OF WSN TECHNOLOGY ....................................................................... 24
  2.5 TRADITIONAL AND EMERGING APPROACHES IN WSN ......................................... 25

3 AGENT-ORIENTED APPROACH ......................................................................................... 26
  3.1 INTRODUCTION ........................................................................................................... 26
    3.1.1 Agent Definition ................................................................................................... 26
    3.1.2 Agent benefits .................................................................................................... 27
    3.1.3 Category of soft Agent ....................................................................................... 27
    3.1.4 Multi-agents System (MAS) ................................................................................ 28
  3.2 COMPARISON WITH OBJECT-ORIENTED APPROACH ................................................ 31
  3.3 BDI .............................................................................................................................. 31
  3.4 JASON – AN AGENT ORIENTED APPROACH ............................................................ 32
    3.4.1 JASON Introduction ............................................................................................ 32
    3.4.2 Introduction of the JASON Agent Programming Language ............................... 33
    3.4.3 Reasoning cycle .................................................................................................. 35
  3.5 SAMSON ...................................................................................................................... 37

4 METHOD FOR SETUP WSN ................................................................................................. 39
  4.1 THE CONCEPT OF MISSION SETUP ............................................................................ 39
WSN Setup by means of Software Agents

List of Figures

Figure 2.1. The WSN overview ........................................................................................................... 18
Figure 2.2. WSN in military [13] ..................................................................................................... 21
Figure 2.3. SNs in Great Duck Island [15] ....................................................................................... 22
Figure 2.4. ITS overview [18] ........................................................................................................ 23
Figure 3.1. MAS overview ............................................................................................................. 29
Figure 3.2. Home intelligent networks ............................................................................................ 30
Figure 3.3. The example of BDI model ........................................................................................... 32
Figure 3.4. The JASON/Agentspeak interpreter cycle for BDI, from (Bordini et al. (2007)) .......... 35
Figure 4.1. Mission data structure .................................................................................................. 40
Figure 4.2. Agent movement .......................................................................................................... 41
Figure 4.3. Communication between Mission-agent and Local-agent .......................................... 43
Figure 4.4. Mission Allocation Decision Mechanism ....................................................................... 45
Figure 5.1. WSN agent system overview ....................................................................................... 46
Figure 5.2. WSN agent system main class ....................................................................................... 47
Figure 5.3. Agent system overview diagram .................................................................................. 48
Figure 5.4. The process of OOP part ............................................................................................... 51
Figure 5.5. The flow of the initial mission-agent ............................................................................. 54
Figure 5.6. Agent capability diagram-check local-agent type and mission ID ................................ 55
Figure 5.7. The sequence diagram of the capability function ............................................................ 56
Figure 5.8. Mission dissemination capability 1 ................................................................................ 58
Figure 5.9. Sequence request Node status ...................................................................................... 59
Figure 5.10. The capability diagram of findNeighbourNode ............................................................. 60
Figure 5.11. Sequence diagram for clone ......................................................................................... 61
Figure 5.12. The capability diagram of cloneAgent() ...................................................................... 62
Figure 5.13. Mission dissemination capability 2 .............................................................................. 62
Figure 5.14. Mission dissemination capability 3 ............................................................................ 63
Figure 5.15. Sequence diagram for merge node status .................................................................. 64
Figure 5.16. Mission dissemination capability 4 ............................................................................. 65
Figure 5.17. Capability diagram of Start mission allocation ............................................................... 66
Figure 5.18. Sequence diagram for mission allocation ...................................................................... 67
Figure 6.1. The beginning of the simulation .................................................................................. 69
Figure 6.2. Mission dissemination .................................................................................................. 70
Figure 6.3. Mission allocation ...................................................................................................... 71
Figure 6.4. The simulation results of the 20% target percentage ...................................................... 72
Figure 6.5. The result of the 20th simulation .................................................................................. 73
Figure 6.6. The simulation results of the 50% target percentage ...................................................... 74
Figure 6.7. The result of the 12th simulation .................................................................................. 75
WSN Setup by means of Software Agents

Figure 6.8. The simulation results of the 80% target percentage .............................................................. 76
Figure 6.9. The result of the 27th simulation .................................................................................................. 77
Figure 6.10. The comparison between target percentages to average simulation percentage .................. 78
Figure 6.11. The simulation results of random value .................................................................................. 79
Figure 6.12. Average Goodness value and the optimized average Goodness value ................................. 81
Figure 7.1. Architecture of MAWSN (figure adapted from [34]) ............................................................... 84
List of Tables

Table 3.1. Comparison between OOP and AOP............................................................................................................. 31
Table 6.1. Statistical results of engaged SNs................................................................................................................. 78
Table 6.2. Average for the goodness values and the corresponding standard deviation............................................. 82
Listing list

Listing 1. An example code of a JASON .................................................................................. 34
Listing 2. Code of initial status of local-agent........................................................................ 49
Listing 3. Example code of internal action.............................................................................. 50
Listing 4. Code of +startToRun............................................................................................. 50
Listing 5. Code of !readTemp .................................................................................................. 50
Listing 6. Code of !SenseEnvironemnt ................................................................................... 50
Listing 7. Code of updatePercepts(agName) .......................................................................... 51
Listing 8. Code of !sendTemp .................................................................................................. 52
Listing 9. Code of initial part of mission-agent ......................................................................... 52
Listing 10. Code of the message exchange ............................................................................. 53
Listing 11. Code of the checkCurrentLocation ....................................................................... 53
Listing 12. Code of updateMissionAgentPercept ................................................................... 54
Listing 13. Code of askLATypeAndMAID ............................................................................. 55
Listing 14. Code of Literal missionStatus ............................................................................... 56
Listing 15. Code of addPercept ............................................................................................... 57
Listing 16. The example code of four cases .......................................................................... 57
Listing 17. Code of the calculation of goodness value ........................................................... 59
Listing 18. Code of +findNeighbourNode in JAVA part ....................................................... 60
Listing 19. Code of +findNeighbourNode in Agent part ....................................................... 60
Listing 20. Code of @cloneAgent [atomic] ........................................................................... 62
Listing 21. Code of the mergeComplete ................................................................................ 64
Listing 22. Code of the .kill_agent ....................................................................................... 65
Listing 23. Code of the startAllocation ................................................................................. 65
Listing 24. Code of the start missionAllocation ..................................................................... 66
Listing 25. Code of the test .................................................................................................... 68
Listing 26. Code of the missionSuccess ................................................................................. 68
List of Abbreviations and Acronyms

- WSN(s): Wireless Sensor Network(s)
- SN(s): Sensor Node(s)
- AOP: Agent-Oriented Programming
- BDI-model: Belief, Desire, and Intention model
- JASON: Java-based Agentspeak interpreter used with SACI for multi-agent distribution over the net
- OOP: Object-Oriented Programming
- Sun SPOT: Sun Small Programmable Object Technology
- CISRT: Command, Control, Communication, Computing, Intelligence, Surveillance, Reconnaissance and Targeting
- CAMP: Control, Alarm, Monitoring and Protection
- ITS: Intelligent Transport Systems
- DSN: Distributed Sensor Networks
- DARPA: Defence Advanced Research Projects Agency USA
- CEC: Cooperative Engagement Capability
- FDS: Fixed Distributed System
- ADS: Advanced Deployment System
- REMBASS: Remote Battlefield Sensor System
- TRSS: Tactical Remote Sensor System
- p2p: Peer to Peer
- ACA: Ant Colony Algorithm
- ATC: Air Traffic Control
- MAS: Multi-agents System
- VI: Video Image
- JADE: Java Agent Development Framework
- GNU: GNU's Not Unix!
- LGPL: Lesser General Public License
- BRF: Belief revision functions
- RX: Receive
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX</td>
<td>Transmit</td>
</tr>
<tr>
<td>MA</td>
<td>Mission Area</td>
</tr>
<tr>
<td>APEP</td>
<td>Adaptive Probabilistic Epidemic Protocol</td>
</tr>
<tr>
<td>MAWSN</td>
<td>Mobile Agent Based WSN</td>
</tr>
<tr>
<td>MAPS</td>
<td>Mobile Agent Platform for Sun SPOTs</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-in First-out</td>
</tr>
<tr>
<td>EA</td>
<td>External Action</td>
</tr>
</tbody>
</table>
1 Introduction

Wireless Sensor Networks (WSN) have been widely used in the remote monitoring, industrial measurement and health care area in recent years. Indoor monitoring for elderly care, environment monitoring and industrial process monitoring are practical examples about WSN usage. [1]

The major challenges faced in WSN are how to find energy efficient and flexible strategies to setup WSN and support their operation, so that the following problems can be addressed [2]: (a) Overloaded data communication among Sensor Nodes (SNs), coordinators and base station, which may increase overheads in energy consumption in the network. As a consequence of this cost due to the overheads, the lifespan of the WSN can be drastically reduced. (b) The inflexible code deployment in the SNs. This problem is due to, firstly, WSN is usually running in a complex and unpredictable operating environment, secondly, user requirements of WSN usually change based on variations in operating environment. Thirdly, concerning the network hardware cost, the current SNs only have a very limited hardware capability to perform different kinds of networks’ missions. In order for a WSN to work properly, on the one hand, coding for SNs needs to be robust enough to face the environment’s complexity and unpredictability. On the other hand, the software has to take into account the SNs’ hardware limitations. However, it is difficult to use the current programming methodology used for conventional computer systems and keep the balance between robust code and low hardware requirements. (c) As the software is usually deployed before the system runtime, if the users want to change the SNs’ directives in runtime, a challenging condition is imposed, which demonstrates the sensor nodes’ adaptability. Due to these three major problems described above, there is a wide research area for flexible and resource efficient solutions for WSN setup and operation.

A WSN is a practical example of the distributed system. In WSN, each SN has its own CPU processor, which controls the sense units (temperature sense unit, light sense unit, etc.) to collect environment data [3]. These data are stored in SN’s memory. A SN needs to cooperate with the other SNs to perform WSN’s mission by aggregating sensed data, forwarding data, etc. SNs collect raw environment data, and send it through its neighbour SNs toward the base station. Base station obtains and analyses the data at the user end of the
WSN Setup by means of Software Agents

system. System user sends directives from the base station to SNs, so the users can control the network’s operation.

The base station (or sink node) has an important role in the whole network, since all the data is sent or receives from it. If a mission has to be submitted to a WSN, this mission has to pass through the base station to go through the network to reach the target portion of the network from where the information is required. As the communication range of the transceivers is usually small if compared with the area in which the network is deployed, the links between the target SNs and base station in the general case is formed by multi-hop. Each component in WSN- base station, SNs, and coordinators – which communicate with other devices within its own broadcast range, require the cooperation of each other to ensure that its messages reach their destinations. Unfortunately, during this message forwarding process, An unnecessary waste of energy occurs, as many redundant messages are generated. Even if coordinator nodes are used, which are specialized responsible for forward data from SNs to base station, the problem still exists, just less redundant data and thus less energy is wasted. Therefore, in wirelessly connected systems, transmission of redundant data is actually a natural characteristic, which is hard to avoid, but can be diminished.

It is possible to reduce the communication between the sensor nodes and base station, by means of exploring autonomous behaviours of the sensor nodes. This alternative could reduce the SNs’ energy cost due to communication, as the sensor nodes would not be too much dependent upon the commands from the base station nor from interaction with neighbour nodes, but could take decisions autonomously. The basic concept is to enable the SNs to make decision on their own, rather than to ask the base station every time what and when to do. This type of approach can be used since the deployment of a sensing mission, in which the sensor nodes can decide which node will participate in a mission accomplishment, instead of depending on the specific assignment from the base station. Intelligent systems, particularly software agents can implement such an approach.

An agent is a reasoning system, which simulates human behaviour. An agent has its own mind to control what it is going to do reasonably, and responds to the unpredictable environment properly. An agent system is similar to a robot community: robots can work together to perform more complex tasks than conventional systems. In fact robots can be seen as agents, i.e. physical agents [4].

Agent-based solutions are innovative approaches to solve the current WSN problems, which is how to provide autonomy to the sensor nodes. An agent oriented programming (AOP) language is used in this thesis, which is called JASON [4] (Java-based AgentSpeak interpreter
used with SACI for multi-agent distribution over the net) to build two types of BDI-model (belief, desire, and intention model) agents: local-agent and mission-agent, which are theoretically based on [2]. The local-agent represents a static software agent that governs a given sensor node, while the mission-agent is a mobile software agent that migrates around the network nodes disseminating and allocating sensing missions. SAMSON [5] is used as a simulation environment to test simulated agents. In [2], a detailed theoretical model is presented to deal with the three problems in WSN: mission dissemination, mission allocation, and how to re-divide the mission after some unpredictable events happened under operating conditions. AOP language-JASON provides the necessary tools that are needed to implement the autonomous behaviours for SNs to instantiate the mentioned theoretical model. The mission-agent is used to disseminate mission in the networks and after intelligent interaction, local-agents and mission-agents will make a decision by themselves, in order to decide whether they will engage a given sensor node in the mission or not.

The contribution provided in this thesis is to use agent technology to deal with current WSN problems for the sensing mission setup. The experimental results show that, by using agent technology, an autonomous solution for WSN setup of sensing mission can be performed, which may actually be used to address the current WSN problems.

1.1 Technology Area

With regards to this project, two technical areas are of remarkable importance:

- **WSN:**
  WSN can ensure that operatives are able to obtain abundant and reliable information at any time, in any place and under any circumstances. So these networks are being used for a number of applications in several domains, such as in the military field, environmental monitoring, and traffic control and so on. This is an important and increasing research area, which motivates efforts in its study.

- **Agent-oriented programming:**
  Agent-oriented programming is a tool to develop agent-based systems, allowing to the intelligent behaviours and interactions among computer systems. Particularly in this work, the focus is on multi-agent systems, a distributed model of agent-based systems in which agents in a distributed system interact to achieve their goals. This particular interest is due to the fact
that WSN can be seen as highly distributed systems, thus providing a perfect match with the idea of multi-agent systems.

1.2 Project Overview

This thesis aims at the instantiation of a theoretical agent model for WSN setup by using a full featured agent oriented approach using BDI-model agents.

In this thesis, two types of programming languages are used, Agent-oriented programming (AOP) to program the agents, and Object-oriented programming (OOP) to develop a WSN simulator.

JAVA is the OOP language used to provide the simulation environment for WSN. Secondly, SNs and base station are programmed as simulation objects in the environment. This provides all the simulation components for the next step, in which JASON - AOP language - is used to code the agents: local-agents and mission-agents. The local-agent controls the simulation component -SNs - to simulate common actions (broadcast, sense the environment) at WSN. Mission-agents cooperate with local-agents to spread the mission directives at the whole system. The agents are tested in the simulation environment to get results for the mission dissemination and allocation, therefore providing a practical example of the theory discussed in [2].

1.3 Problem statement

The problem to be addressed in this project is how to map a decentralized mission dissemination and allocation strategy proposed in [2] to the BDI (Belief-Desire-Intention) agent model, and then, how to implement this mapped solution in a full featured agent platform.

1.4 Thesis Goals and Expected Results

The goal of this thesis is to apply an agent-oriented approach using the full featured agent-oriented environment to perform WSN setup.

The main result expected from this thesis is the mapping from a theoretical model of WSN mission setup to a BDI (Belief-Desire-Intention) agent model. Quantitative results are expressed in terms of measurements of metrics related to the theoretical model used, which
WSN Setup by means of Software Agents

focuses on the number of sensor nodes engaged in the mission and the quality of the engaged nodes, compared to the mission specification.

1.5 Thesis Structure

Chapter 2 provides technical background information about WSN, highlighting the main aspects of this research area. Chapter 3 provides information about agent-oriented systems, focusing then on details about the agent-oriented programming technology used in this work, which is based on JASON [4]. In the chapter 4, the details about the theoretical agent approach solution [2], in which this thesis is based on, are provided. Chapter 5 presents the details about the implementation of the reference model using Jason and SAMSON. In Chapter 6, the simulation results are presented and compared with optimum solution. In Chapter 7, a survey of the recently agent-oriented approaches for WSN is presented, highlighting the difference between those solutions and the one studied in this thesis, pointing out their pros and cons. Chapter 8 concludes the thesis and details future work suggestions.
2 Wireless Sensor Network

2.1 Introduction of WSN

Due to advances in low-power wireless communications, low-power analog and digital electronics, the development of low-cost and low-power SNs that are small in size has received increasing attention. SNs have the ability to sense the environment nearby, perform simple computations and communicate in a small region. Although their capacities are limited, combining these small sensors in large numbers provides a new technological platform, as WSNs. WSNs provide reliable operations in various application areas including environmental monitoring, health monitoring, and military surveillance.

A WSN is a collection of millimeter-scale, self-contained, micro-electro-mechanical devices, which consist of micro-controller, communication unit, power and sensors. The general processor is not suitable as a control unit, because the number of nodes of a WSN system maybe thousands, the environment where the sensors deploy probably is tiny and the utility of sensors should maintain as long as possible. For WSN, low cost, small and low power consumption micro-controllers should be considered, just as the ARM920T which is used in a kind of sensor node called Sun SPOT [6]. For communication, the environment of the work place, the distance between the nodes, the bit rate that the communication demands and the capability of the low power consumption must differ according to the application requirements. Afterwards, the communication media and the communication protocol can be decided under these conditions.

A WSN consists of a number of sensors spread across a geographical area. Each sensor has wireless communication capability and sufficient intelligence for signal processing and networking of the data [7]. A WSN can be deployed in remote geographical locations and requires minimal setup and administration costs. Moreover, the integration of a WSN with a bigger network such as the Internet or a wireless infrastructure network increases the coverage area and potential application domain of the ad hoc network. Sensed information is relayed to a sink node by using multi-hop communication. The sink node is a sensor node with gateway functions to link to external networks such as the Internet and sensed information is normally distributed via the sink node. The whole system framework of WSN usually consists of sensor nodes, sink nodes and manager node as shown in Figure 2.1. The manager node can be any
WSN Setup by means of Software Agents

computer in a local area network that has access to the sink or even a computer linked to the internet that is located thousands of kilometers from the sink node.

![WSN Diagram](image)

**Figure 2.1. The WSN overview**

- **Network topology**
  Because of the characteristics of an ability to cope with node failures, and also dynamic network topology must be fulfilled, many WSN choose the ZigBee [8] which is a mature, low-cost, wireless mesh network protocol. The mesh network has self-healing capability, as the node can always find another path if its next hop node has failed. Nevertheless, for the feature, the routing and the multi-hop technique must be applied, which will increase the delay time.

- **Construction cycle and adaptability**
  If the user change requirements, for instance, the user wants to add another kind of sensor to monitor other elements of the environment, WSN system have to provide a mechanism to allow the network to be smoothly upgraded without requiring big modifications in the original system.

### 2.2 The Key Technology of WSN

- **Networks security**
  Security is the prerequisite of system availability, it needs to ensure that the premise of communication security to reduce overhead to the whole system and analysis algorithm of energy conservation. Since security threats of WSN are different from conventional networks, the existing network security mechanisms cannot be applied to them, requiring the development of specialized protocols.
WSN Setup by means of Software Agents

Currently there are two ideas: on the one hand, from the perspective of maintaining the routing security, it needs to look for the safest possible route to ensure network security. On the other hand the focus should be on the security agreement, which may lead to a universal model [9].

- **Energy management**
  Sensor energy is extremely limited. In the network, sensors often fail due to the limited sensor energy. The constraint of power energy is a big problem which hinders the application of WSN. Information transmission of a sensor is more consuming than a calculation. Sensors transmit information of one bit which is sufficient to perform 3,000 calculation commands [10]. So a clear way to diminish energy consumption in WSN is by diminishing communications among the sensor nodes.

- **Data fusion**
  Data fusion makes multiple copies of data or information comprehensive, and obtains summarized data from the raw redundant data [11]. Data fusion can reduce the data transfer during the data aggregation process, improve the accuracy and reliability for information, and enhance the overall efficiency of the network. In the application layer it can use distributed database technology to filter the collected data gradually. In the network layer, a lot of routing protocols combine with the data fusion mechanism to reduce the amount of data transmission.

- **Mobile management**
  Flooding mechanisms are generally used in mobile ad hoc networks to update routing tables when nodes change their positions. However, for the resource-limited WSN SNs, flooding represents very energy consuming mechanisms, so more efficient solutions are required.

- **Scalability**
  This refers to the characteristic of a network to grow in terms of number of nodes, or coverage capability while keeping affordable costs in terms of resource consumption. This is of great concern in WSN, as many flood-based solutions present an exponential increase in the use of network resources with the increase of the network size

- **Robustness**
  WSN particularly are usually deployed in harsh environments or in inaccessible areas, such as jungles or inside of walls or bridge structures. SNs need to be resilient, strong and adapt easily
WSN Setup by means of Software Agents

to harsh environmental conditions. Therefore, WSN hardware and software also must have the high robustness and fault tolerance [10].

2.3 WSN Applications

In the past few years, many WSNs had been deployed. These applications serve to explore the requirements, constraints and guidelines for general sensor network architecture design. In this part, a snapshot of the recent deployed sensor network applications is presented, trying to identify the related research challenges associated with such applications.

➢ Military applications

The research of WSNs originated in the military domain. In this domain, WSNs are an indispensable part of CISRT (Command, Control, Communication, Computing, Intelligence, Surveillance, Reconnaissance and Targeting). Figure 2.2 presents an example of such CISRT systems. The target of CISRT is to use high-tech knowledge to design a chain of command for a battlefield scenario which includes command, control, communication, computing, intelligence, surveillance, reconnaissance and targeting. This system has gained military interest and attention in developed countries.

Due to WSN being composed of some intensive, lower cost, distributed nodes, it has self-organizational and fault tolerance capability to deal with system breakdowns. Because of this above point, WSNs are applicable to the characteristics of a battlefield environment. In war, surveillance is very important in conflict areas and strategically military terrace by means of paving WSN, in order to observe enemy’s movements.

There are many examples of military use of the WSN. For instance, in 2003, The United Nations Peacekeeping Force entered Iraq, used the commercial spy satellite and the ultramicroscopic WSN to monitor the atmosphere, water and soil in Iraq in order to establish if Iraq violated the international covenant for nuclear and BC Weapons [12].
WSN Setup by means of Software Agents

Figure 2.2. WSN in military [13]

- **Agricultural applications**
  
  High yields and good quality of crops is a significant requirement for all countries. In this aspect, WSN has applicability [12], as it can be used in monitoring irrigation, any change to soil and atmosphere, conditions for wildlife and vegetation on the ground.

  Information acquisition, transmission, disposal and application are the biggest four elements in digital agriculture. The advanced sensor technology and smart information disposal is important measure to guarantee to accurate agricultural information. WSN provides a new situation for information collection and disposal, and it can offset some limitation, compared with the traditional supervisory data.

  WSN can send some data to agro-organizations which include growth information about crops and CAMP (Control+Alarm+Monitoring+Protection) information about crop pests. The application can assist farmers to identify problems and achieve successful digital etc.

  Thus, WSN will be wide foreground in any aspects of agriculture.

- **Environment Observation and Forecasting System**

  WSN has many advantages, such as easy to dispose, densely to collocate and lower cost etc. So it can provide some well-suited in Environmental Science. In other words, WSN can be used in Meteorological Research, the monitoring of natural disasters and tracking rare animals.
WSN Setup by means of Software Agents

For example, the Petrel, already a rare bird, now in danger of extinction. Thus in 2002, the Inter Co. installed many sensors besides their nests in the natural reserves of Florida Great Duck Island [14]. The researcher used sensors which included the ray sensor, the humidity sensor, and the air-pressure sensor etc. These sensors send the data about living Petrels to the server in California (Figure 2.3) where the data have collected. Finally, it was achieved self-monitoring.

![Figure 2.3. SNs in Great Duck Island [15]](image)

- **Industrial applications**
  Self-organization, miniaturization and sensing capability to exterior determine essentiality of WSN in industrial areas. In some dangerous working environment, such as coal, oil drilling and nuclear power plants, WSN can detect important information on the work site about workers.

  According to a fault diagnosis of mechanism, the Inter Co. have already installed 200 nodes of WSN on the chip equipment. They were used to monitor the equipment about jarring and provided the testing result, after the result exceeded the specified value. The effect is very significant. At the same time, the biggest construction firm in USA also used WSN to monitor the running status of London Subway [12].

  In addition, the use of WSN can monitor the pressure and temperature of electric poles, which can quickly locate and identify the fault and deal with the problem fast.

- **Home and Health applications**
  With the aging population and increased need to care for the elderly, there are fewer of the younger generation to administer the necessary care and supervision. This is the reason that
many researchers devote their time to evolving smart homes. These homes offer the occupant a level of convenience not seen in traditional homes by using technology to create an environment that is aware of the activities taking place within it.

WSN system of Home care is the part of “Coping with the aging of population in USA” by Inter Co.. This system embeds the sensors into facilities such as shoes, furnishings and electrical household appliances. Thus it can help some people including the older, patients and some handicapped people living alone [16].

**Intelligent Transportation Systems**

The fast-growing domain of Intelligent Transport Systems (ITS) spans from flight transport and traffic management to in-vehicle services, such as driver alert or traffic monitoring. Figure 2.4 provides an example of such type of system. As a consequence, transportation information collection and communication play a key role in all intelligent transport applications. Unfortunately, most conventional ITSs can only detect the vehicle in a fixed position, and the costs for deploying communication networks and power cables represent a barrier for their widespread adoption. In fact, nowadays, collecting traffic data for traffic planning and management is achieved mostly through wired sensors. The equipment and maintenance cost and time-consuming installations of these existing sensing systems prevent large-scale deployment of real-time traffic monitoring and control [17].

![Figure 2.4. ITS overview [18]](image-url)
Finally, there are many applications in WSN, and presently the shipment rate of wireless sensors is up to a hundred million. Some corporations such as Cisco, Inter etc. have already achieved various applications and services of WSN. [12]

### 2.4 Evolution of WSN Technology

The evolution of WSN can be divided into three phases.

- **The 1st phase: Traditional sensor system**
  
  The first use of traditional sensor systems can be traced back to the Vietnam War during the 1970s. The American troops put more than 20000 sensors which were called ‘Tropical Tree’ into the battlefield. Once the enemy passed these sensors, which detected noise, then they sent these messages to the command center automatically. In this way, the troops unfolded an offensive to bomb out almost 46,000 trucks [12].

  Such the early sensor, the characters are: the SNs only can get data by themselves. There are no computing capability nor communication skills between any two sensors.

  The traditional sensor system can only normally capture a single signal and two SNs can do easily communicate (Point to Point). The network often uses a structure which is hierarchical.

- **The 2nd phase: the nodes integration of WSN**
  
  The second phase was between the 1980s and 1990s. In 1980, the project of DSN (Distributed Sensor Networks) by DARPA (Defense Advanced Research Projects Agency USA) was a pioneer of modern sensor networks. [12]

  These nodes of WSN have mutual co-ordination and also can run independently. This was an enormous plan for the technology of that time. In this phase, WSN also was applied in military affairs such as CEC (Cooperative Engagement Capability), FDS (Fixed Distributed System), ADS (Advanced Deployment System), REMBASS (Remote Battlefield Sensor System) and TRSS (Tactical Remote Sensor System).

  Such sensor systems use a modern minimize node which has perceptual and communication skill and computing capability.

- **The 3rd phase: Multi-hop Mesh Networks**
  
  The third phase was until the 21st Century to now. This kind of sensor’s characteristic is that there is self-organization, lower cost and low power consumption.
2.5 Traditional and Emerging Approaches in WSN

In the WSN research area, a significant challenge is how to provide flexible strategies to perform the network setup and configuration with low overheads. Traditional approaches to deal with this problem use the pre-planned strategies, which are deliberately modelled, designed and tuned before the network deployment. [19]

For these traditional approaches, one of these is discussed in [20]. Self-organization is the key to implement self-calibration, autonomous coordination and p2p (peer to peer) communication in sensor networks. Current research is mainly dedicated to solve this problem in an inappropriate pre-planned manner. The swarm intelligence of the Ant Colony Algorithm (ACA) provides a novel and efficient method for self-organization. For the similarities between a sensor network and an ant colony, the sensor networks will benefit from the bio-inspired self-organization by applying the ACA in optimization, structure formation and task/resource allocation.
3 Agent-oriented Approach

3.1 Introduction

Agent-oriented programming (AOP) is a fairly new programming paradigm that supports a societal view of computation. In AOP [21], objects known as agents interact to achieve individual goals. Agents can exist in a structure as complex as a global internet or as simple as a module of a common program in a single computer. Agents can be autonomous entities, deciding their next step without the interference of a user, or they can be controllable, serving as an intermediary between the user and another agent.

3.1.1 Agent Definition

Agent is a wide spread used term in Computer sciences which is refers to Distributed Artificial Intelligence, Machine Interaction, Software engineering, System Emulation, etc. It is also applied for Business Process Management, Heterogeneous Information System, ATC (Air Traffic Control), Electronic Commerce, On-line Games, etc.[22]

A class of systems in the agent domain is one referring to reactive systems. An agent is a reactive system that exhibits some degree of autonomy in which it reacts to stimulus from its environment. Another more sophisticated class of agents are those that have higher degree of autonomy, in the sense that tasks are delegated to them, and the system itself determines how to achieve or perform this task. Such systems are called ‘autonomous agents’ because they are considered as being active, purposeful producers of actions: they are sent out into their environment to achieve goals for their users, and they actively pursue these goals, figuring out for themselves how best to accomplish these goals, rather than having to be told in low-level detail how to do it. [4]

Wooldridge and Jennings [23] defined that agents should have the following properties:

- **Autonomous:**
  Agents operate without direct intervention by humans or other agents and have some control over their internal state.
WSN Setup by means of Software Agents

- **Reactive:**
  Agents should have one or more “senses”, meaning that they need to perceive one or more dynamic characteristic of its environment.

- **Proactive:**
  Agents are directed and goal oriented, taking steps towards creating a world state where their goals succeed.

- **Social ability:**
  Agents are able to communicate through some medium to other agents or actors.

### 3.1.2 Agent benefits

As Wooldridge shows in [24], agents are useful for providing four key benefits:

- A natural abstraction method of designing solutions to complex problems involving interactions.
- They are inherently distributed systems.
- They can be used to evolve legacy systems.
- They are open systems that are flexible and adaptable to changes.

Jennings [25], calls such systems a “logical evolution to traditional software engineering”, and has also proposed two reasons in support of the paradigm. One is that of adequacy: being able to improve the method of building complex distributed software systems.

The other is the establishment, being able to be accepted by mainstream software system designers as a standard approach. These benefits provide the essentials for modeling, with the added ability of being able to make use of high level interactions and changing organizational relationships to tackle important problems [5].

### 3.1.3 Category of soft Agent

In [26] [27], Hwanna et al. present a classification that refers to the study of types of entities which presents several dimensions to classify existing software agents according to different characteristics, which are:

- **Collaborative agent**
  In comparison with other types of agent, collaborative agents focus on their autonomy and collaboration with other agents. Collaborative agents compose a system in which the agents
work to complete a common goal. The motivation for using collaborative agents is to provide solutions to inherently distributed problems.

- **Interface agent**
  Interactivity with users is the most important attribute of an interface agent. A typical example is Personal Assistant, which can use a new application to provide guidance information for users, it also can analyze the operation users used, in order to provide better service for the future users. The key characteristics of an interface agent are autonomy, and cooperation with users and/or other agents.

- **Mobile agent**
  A software agent is called mobile agent if it is able to migrate from node to node to work in a heterogeneous network environment. Its key characteristics are mobility, autonomy, and cooperation. Mobility is not the necessary attribute of a software agent, but for mobile agents this characteristic is outstanding besides other characteristics shared by other types of agents, such as autonomy and coordination.

- **Information agent**
  The information agent's role is to help people deal with the explosive growth of information and to collect or handle distribution of information. This means that information agents are most useful on the Web service where they can help people with laborious tasks.

- **Reactive agent**
  Reactive agent acts and responds to the current state of their environment based on a stimulus-response scheme. Autonomy and reactivity are key characteristics of a reactive agent.

- **Hybrid agent**
  Hybrid agents refer to those agents whose constitution is a combination of two or more agents’ properties within a singular agent. These properties may be mobile, interface, information, collaborative … etc. This is a common way in the based-agent system, and an example of this is collaborative interface agents.

**3.1.4 Multi-agents System (MAS)**

A large number of systems may have not only one agent, especially the complex distributed systems, which have two or more agents. In these systems, each agent is an independent
WSN Setup by means of Software Agents

behaviour entity. Because of encapsulation for status and behaviour, they are not only independent but also have a complex relationship:

- Between each different agent may exist a structural dependence, which provides a Reciprocal relationship, Client/server relationship, Member Relation and Inheritance relationship.

- Between each different agent may exist associative behaviour, there is some communication and regular interaction between them. These agents may collaborate, compete and consult in order to achieve the design objective for the entire system.

Figure 3.1 gives an overview of MAS [4]. From the figure it is possible to observe the shared environment that the agents occupy at the bottom of the figure. Each agent has a ‘sphere of influence’ in this environment, but more generally, and more problematically, is the possibility that the spheres of influence overlap: the environment is jointly controlled which makes life for the agents more complicated. Due to this kind of shared and overlapping environment, it is more difficult for the agents to achieve an outcome that they desire, i.e. they will have to take into account how the other agents with some control are likely to act.

![Figure 3.1. MAS overview](image-url)
WSN Setup by means of Software Agents

These agents will have some knowledge of each other, though it may be the case that an agent does not have complete knowledge of the other agents in the system. According to the definition of MAS, a concrete example is provided in the following to illustrate it [22].

➢ **Home Intelligent Network**

Considering a home intelligent network, this is composed of a set of computers. Many software agents reside in these computers and internet, which include a user agent, home information management agent, message agent, VI (video image) agent and alarm agent etc.. Every agent should play one or more roles and take the responsibility for home networks. They have certain functions and provide services as well. Through these agents to interact with a variety of family facilities, sequentially the whole network can provide a security service for the entire family. The whole system can be known as a MAS (Figure 3.2).

![Figure 3.2. Home intelligent networks](image-url)
WSN Setup by means of Software Agents

3.2 Comparison with Object-oriented approach

A classical methodology for computer systems development is object-orientation [28]. Table 3.1 provides a brief comparison of the main elements used in agent-oriented programming and object-oriented programming.

<table>
<thead>
<tr>
<th>Basic Unit</th>
<th>OOP</th>
<th>AOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters defining states of</td>
<td>unconstrained</td>
<td>beliefs, commitments,</td>
</tr>
<tr>
<td>basic unit</td>
<td></td>
<td>choices</td>
</tr>
<tr>
<td>Process of computation</td>
<td>message passing and response method</td>
<td>message passing and response method</td>
</tr>
<tr>
<td>Type of message</td>
<td>unconstrained</td>
<td>inform, request, offer,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>promise, decline, ....</td>
</tr>
<tr>
<td>Constraints on methods</td>
<td>none</td>
<td>honesty, consistency, ....</td>
</tr>
</tbody>
</table>

3.3 BDI

The Belief-Desire-Intention (BDI) agent model is a fundamental concept for this thesis. The BDI model which originated by Rao [29] is the most popular architecture for intelligent agents. The agents reside in a dynamic environment, perceive from the environment, and maybe change the environment. There are three internal abstractions involved in this process which are beliefs, desires and intentions. These abstract concepts represent respectively the information of the environment, motivation, which means that the state the agent wishes to achieve, and decision components, which can be interpreted as actions the agent chose to perform.

Figure 3.3 illustrates a simple BDI agent decision making procedure. This is a temperature sensor node which gathers the information of temperature, remaining statements of battery, sampling rate and node’s state from the environment. It is possible to see the information in the belief set showing that the battery is low and the sampling rate is high. These information triggers the desire to achieve a goal of saving energy and both maintaining the node activated. To implementation the goal, an action ‘low sample rate’ is chosen to carry out. The grey highlight is use to illustrate the procedure.
3.4 JASON – An agent oriented approach

3.4.1 JASON Introduction

JASON is an interpreter of an extension of the AgentSpeak logic-based agent-oriented programming language which has benefited from Belief-Desire-Intention (BDI) model. Some of the features available in JASON are:

- Speech-act based inter-agent communication (and annotation of beliefs with information sources);
- Annotations on plan labels, which can be used by elaborate (e.g., decision theoretic) selection functions;
- The possibility to run a multi-agent system distributed over a network (using JADE, but other middleware can be used);
- Fully customizable (in Java) selection functions, trust functions, and overall agent architecture (perception, belief-revision, inter-agent communication, and acting);
- Straightforward extensibility (and use of) by means of user-defined “internal actions”;
- Clear notion of multi-agent environments, which can be implemented in Java (this can be a simulation of a real environment, e.g., for testing purposes before the
WSN Setup by means of Software Agents

system is actually deployed).

There are various kinds languages which can constrict a MAS, e.g. Jack, JASON, Impact, etc.

Jack, for instance, is unlike most of other agent-oriented programming languages which belong to the declarative language or the hybrid language of declarative and imperative features. It is designed by a kind of a purely imperative approach. Jack has two advantages. The first one is that Jack uses the BDI model, so it inherits the merit of BDI as mentioned above. The other one is Jack is a Java-based language, therefore it can be easy to handle by any programmer who is skilled in Java. Also, Jack presents drawbacks. The primary issue is that Jack is a commercial project. So it is not open source, as is Jason. In addition, as Jack is not an agent-oriented language, in practice the project which was written by Jack will be very difficult to maintain or modify [30].

3.4.2 Introduction of the JASON Agent Programming Language

The main language constructs of JASON are: Beliefs, Goals and Plans. An agent constantly perceives the environment for updating the belief base or triggering an event, choosing a plan to achieve the goal, so as to change the environment. [4]

The syntax of JASON can be briefly introduced as:

- **Beliefs**
  Beliefs represent the information available to an agent (Beliefs can be derived from initialization of the user, perceiving the environment and the beliefs of other agents through communication.)

  \[ \text{Likes (john, music)} \]

- **Goals**
  There are two types of goals: achievement goals and test goals.

  Achievement goals represent states of affairs the agent wants to bring about (come to believe, when goals are used declaratively)

  \[ \text{!clean (house)} \]

  Test goals attempt to retrieve information from the belief base

  \[ \text{?publisher (P)} \]
Events and Plans

The way reactive planning systems react to such events is by executing plans; plans are courses of action that agents commit to execute so as to handle such events.

When the agent’s beliefs or goals change, trigger events happen. Then plans should be executed. Plans are sets of events, which represent the agent’s know-how.

An Agentspeak plan has the following general structure:

\[
\text{Triggering event: context } \leftarrow \text{ body}
\]

The triggering event denotes the events that the plan is meant to handle. There are six kinds of triggering events, which are:

- +b (belief addition)
- -b (belief deletion)
- +!g (achievement-goal addition)
- -!g (achievement-goal deletion)
- +?g (test-goal addition)
- -?g (test-goal deletion)

The context represents the circumstances in which the plan can be used. The context is a logical expression, typically a conjunction of literals to be checked whether they follow from the current state of the belief base.

The body is the course of action to be used to handle the event if the context is believed true at the time a plan is being chosen to handle the event. The body is a sequence of actions and (sub) goals to achieve. An example of a JASON code is provided as follows:

**Listing 1. An example code of a JASON**

```
+green_patch(Rock): not battery_charge(low)
  \leftarrow
  ?location(Rock, Coordinates);
  !at(Coordinates);
  !examine(Rock).

+!at(Coords): not at(Coords) & safe_path(Coords)
  \leftarrow
  move_towards(Coords);
  !at(Coords).
  +!at(Coords) ...
```
Environment

To simulate multi-agent systems, the environment where the agent can be situated has to be implemented in Java. The environment is shared by multiple agents. Also an agent can perceive and affect the environment.

3.4.3 Reasoning cycle

The reasoning cycle represents the process of JASON interpreter, which can be divided into 10 steps. Initially, it is important to explain about what each pattern represents in the following Figure 3.4. Rectangles represent the agent states (i.e. the belief base, the set of events etc.); rounded boxes, diamonds and circles represent the functions used in the reasoning cycle. Some elements in the figure are labeled with numbers, which refer to steps of the reasoning cycle in turn.

![Figure 3.4. The JASON/Agentspeak interpreter cycle for BDI, from (Bordini et al. 2007).](image-url)
WSN Setup by means of Software Agents

- **Step 1 – Perceiving the Environment**
  In this step, agents perceive the environment, then get some information about the environment, which can be considered as beliefs updating.

- **Step 2 – Updating the Belief Base**
  After Step 1, the beliefs need to be stored in the Belief Base. This process can be done by a belief update function and the method implementing, which is called BRF (Belief Revision Function).
  The BRF updating follows 2 rules:
  - Each belief which was perceived by step1 also do not presents in Belief Base should be added to Belief Base.
  - Each belief which was not perceived by step1 should be deleted from Belief Base.

- **Step 3 – Receiving Communication from Other Agents**
  The reasoning cycle works on receiving messages from other agents of the MAS. The reasoning cycle stores messages by using checkMail method. After the messages have been received, the reasoning cycle only processes one message which has been selected by the message selection function in each reasoning cycle turn.

- **Step 4 – Selecting `Socially Acceptable` Message**
  The interpreter then needs to implement a method called “social acceptance function”. This method checks if the message passed by the message selection function can be accepted by the agent.

- **Step 5 – Selecting an Event**
  In BDI architecture, events are produced by the changes of beliefs or goals. Also, in JASON interpreter there is only one pending event which can be dealt with in one reasoning cycle. So, as there is more than one event needed to process, a customized method called “event selection function” is needed to decide which event would be executed first. Otherwise, events will be selected by chronological order. If there are no more events to process, then the reasoning cycle will directly skip to step 9.

- **Step 6 – Retrieving all Relevant Plans**
  Once the event has been selected, what is going to do of the interpreter is retrieving all the plans in the Plan Library. The compare will process between the event got from step 5 and the
WSN Setup by means of Software Agents

trigger event in the Plan Library. If there is no matched plan, then that event will be discarded. Also, maybe there is more than one plan that is available.

- **Step 7 – Determining the Applicable Plans**
  In this step, the interpreter goes to pick up the more suitable plan by checking the context is true or not. In JASON the context’s forms are expressions, which can be texted by the belief base. As in step 6, the set of applicable plans may have more than one plan, this can be solved by the next step.

- **Step 8 – Selecting One Applicable Plan**
  The application plan selection function can implement the selection of one suitable plan from the applicable plans. The function can be customized by the programmer. As pre-defined by JASON, the plan is selected by the order in which they appear in the Plan Library. After that plan was selected, under the external events circumstance, the interpreter creates a new intention which can be chosen for further execution; on the other hand, under the internal events circumstance, if the plan body does have a goal, then the applicable plan of this goal should be pushed to the set of intention and be executed first.

- **Step 9 – Selecting an Intention for Further Execution**
  As expected from the previous similar case, there is more than one intention in the set of intention. However in one reasoning cycle, only one intention can be executed. So they have to compete with each other. To solve this problem, JASON pre-defines an intention selection function using the ‘round-robin’ scheduling. Also, programmers can customize which function that can set the priority for each intention.

- **Step 10 – Executing One Step of an Intention**
  A set of action which is the body of suitable plan would be processed in this step. A plan’s body has a set of formulas for implementation, and formulas carry them out one by one .Until a formula is finished, this formula would then be deleted.

### 3.5 SAMSON

SAMSON is a simulator of strong multi-agent system for WSNs [5], relied on the work of Rao’s BDI model, also using JASON framework. This simulator is based on rational agents, a kind of physical node named TMote Sky Sensing Unit, as well as its radio operations, and a
WSN Setup by means of Software Agents

widespread model of sensor network environment. Therefore, SAMSON provides an extensible simulation WSN system for general purposes.

In particular, the hardware model is implemented by simulating the working of the TMote Sky sensing unit with Controller, Sensors and actuators, Communication device, and Power Supply. With regards to the environment part, it is grid-based. It contains the environment elements, i.e. temperature or humidity etc., which represent the factors in the real world, as well as the transmission features which represent as radio ranges, attenuation factors and obstacles. SAMSON implements the TileWorld approach which provides an interface to show the node states, the environment parameters and the dynamic node map to users.
WSN Setup by means of Software Agents

4 Method for Setup WSN

4.1 The Concept of Mission Setup

One of the main problems in WSN is how to construct the network in an efficient and flexible way. A traditional approach is to implement a centralized strategy that provides an 'oracle view' of the network, which needs periodically to retrieve information from the sensor nodes to a coordinator node and results in a lot of consumption of energy and network traffic in an inefficient way. In this thesis, an approach in which each node in the WSN has a certain processing ability to decide what they should do autonomously is explored. Freitas et al. [2] present an approach to implement this autonomous WSN network, in which the WSN mission setup is divided into two parts, namely Mission dissemination and Mission allocation. The first part is about how missions transfer from node to node, and the latter is about which nodes will be engaged to the mission.

It is important to introduce the concept of Mission that is bonded with a Mission Agent (the mobile agent that carries and performs the mission in the network) first. A Mission can test whether a node in the environment is suitable or not to perform a given measurement required for that mission. The criterion of choosing nodes can consist of the type and the capacity of nodes and the amount of sensors that are needed to be involved in the mission. Moreover, it informs selected sensors what actions they are going to perform.

Figure 4.1 illustrates the structure of missions which include a mission ID, demanded characteristics, the evaluation criterion and the desired amount of sensors. The first parameter is mission ID that can be used by sensors to identify each mission. The second parameter is characteristics of demand which include some factors that should be considered when the mission is being disseminated. As a mission arrives to a node, the mission needs to inspect if the node corresponds with the mission demands, e.g. if the type of sensor or the accuracy of a node complies with the mission requirements. The third parameter carries the evaluation criterion which decides if a node will engage to a particular mission. This process is involved in the mission allocation procedure. The fourth parameter is the desired amount of sensors which define the number of nodes could implement the mission. In [2], there is an extra parameter in the structure of the mission which includes the location information that indicates the Mission Area (MA). MA is an area in which a mission has to be performed. Any node in the MA can be selected to perform the mission. When a mission agent is injected in
any node in the environment, it firstly calculates the path to move to the MA by using the location parameters. However, in the work developed in this thesis, it is considered that the whole environment is the Mission Area, in other words, all nodes in the simulations are consisted to be in the MA. Thus, the action that performs the movement to the MA is not implemented.

<table>
<thead>
<tr>
<th>Mission ID</th>
<th>Required Sensors</th>
<th>Evaluation Criterion</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Id_t$</td>
<td>$\triangle \Box$</td>
<td>$g(p_1, p_2, \ldots p_n)$</td>
<td>$% \cdots n$</td>
</tr>
</tbody>
</table>

Figure 4.1. Mission data structure

4.1.1 Agent Description

As already mentioned, agents are usefully classified into six types according to their different characteristics. In this thesis the agents are used for different purposes and are according to their mobility, so agents are classified as either mobile or static, which means the agents can be static in hosting nodes or have the ability to move from one node to another.

- **Static Agents:**
  
  In this thesis, it also can be named local-agent which responsible for providing local information about the sensor node. They also provide this information to the mobile agents that may visit the node, so that these mobile agents can perform their missions.

- **Mobile Agents:**
  
  The specified sensing missions are sent to network via mobile software agents which are called mission-agents. Their role is to move or migrate from node to node until the whole network is informed about the mission. An intelligent mechanism of these mobile agents is responsible for the decision about which nodes should receive the agent and take part in the mission and then perform the mission. The movement of the mission-agent in the network solves the first part of the problem, which is the mission dissemination.
4.2 Mission Dissemination

In the scenario considered in this thesis, mission-agents are spread to the whole network, but only the suitable nodes remain with a duplicate or clone of these agents. This process works as follows:

A mission-agent can be injected into the WSN by any node, if this node matches the condition of mission, for example the type of node that the mission demands, then an act called “clone” will be executed. Clone represents the action in which a mission-agent leaves a copy to current node, and after that, this node broadcasts the mission-agent to its neighbour nodes. Alternatively, if this current node is not eligible to the mission, the mission-agent performs another action called “move-clone”. The move-clone consists of the following: first the node broadcasts the mission-agent to its neighbour nodes, and then the mission-agent in this node is discarded.

Figure 4.2 presents the agent movement through the nodes. In part (a), the initial node is shown inside the entire area in which the mission-agent currently is, (b) and (c) shows the actions of “move-clone” and “clone” respectively.

Figure 4.2. Agent movement
After a mission-agent arrives at a node, first the mission-agent requests the characteristics of the current node from the local-agent that governs that node, and then the local-agent replies its status. Afterwards, the mission-agent transmits its mission ID to the local-agent and this local-agent answers if it is aware of it or not.

Assuming that the mission-agent estimate current node fulfils the demands of the mission-agent and if the local-agent knows about the mission-agent, it implies that there is a mission-agent with the same mission occupying the node. Under this condition, the latest arrived mission-agent informs the local-agent about the information of nodes which it previously visited. After that, the local-agent tells this information to the preoccupied mission-agent. The preoccupied mission-agent should then fuse this information with its own. At the end of the procedure, the latest arrived mission-agent will be discarded. Otherwise, if the local-agent does not have knowledge about that mission-agent, there will be a data exchange between the mission-agent and the local-agent, which the node’s status should be transferred to the mission-agent by the local-agent. After that, the mission-agent will perform the clone action.

Assuming that the node is not eligible to perform the mission, if the local-agent knows about the mission-agent, it implies that at least one such mission-agent has already visited the node. In this case, to avoid the redundant transmission, it directly decides to perform the discard action. On the contrary, if the local-agent does not have knowledge about that mission-agent, the move-clone action will be executed. This interaction between the mission and local agents is illustrated in the sequence diagram presented in Figure 4.3.
An important feature to mention is that during the mission-agent clone and move-clone actions, it adds to itself the particular information in relation to the last visited node about its capability to perform the mission. This is called the *goodness* of the node to perform a given mission. This information will be used by the other nodes that receive this mission-agent in the mission allocation process that is presented in the following.
WSN Setup by means of Software Agents

4.3 Mission Allocation

Mission allocation is the second main part of the proposed solution to setup a sensor network to accomplish a sensing mission. An ineffective solution would be to take centralized decisions and send the specific part of a given mission to the specific node that will take care of it. However, these decisions must have information about the network as a whole. In this way, it may give rise to unnecessary resource consumption. In order to overcome extra overheads, a fully decentralized decision mechanism with this goal is proposed.

4.3.1 The decision mechanism

When the mission-agent decides to clone, the initial allocation must be set a timer. This timer is a duration time which the mission-agent should visit all nodes to disseminate a specific mission. The mission is disseminated after the timer expiration. So two factors are considered to choose this duration time, one is the number of hops that a mission-agent should perform to cross all nodes from the initial arriving node to the most distant one, and another is the broadcast range of the SNs. To avoid a premature start of the decision process, the timer is set to a value, which is two times as the actual calculated value. By the expiration of the timer, it means all nodes should have been visited by the mission-agent using the clone or move-clone actions at least one time, and there are no more movement of information.

During the dissemination of the mission by the mission-agents and in each node, the mission-agent will perform a local decision making process that will provide as a result if the node may join or not the mission. When the neighbours transmitted the mission-agent during a clone or move-clone movement, it can obtain goodness information of the node and the information of node’s neighbours.

The decision process uses a weighted probability calculation. By receiving the information about its neighbours, via mission-agents, the mission-agent makes a generalization of that configuration in its neighbourhood on what exists in the entire area. By making this, it assumes that the amount of requested nodes in its neighbourhood is proportional to what is required in the mission directions for the entire node area.

Each mission-agent adds the goodness values based on goodness function. However, the goodness values can be the random values or anything else depending on the simulation for this thesis. The goodness received from its neighbours to the one of its hosting node; calculates its contribution to this result; and based on that, evaluates the adequacy of the node's engagement in the mission, by means of a probability calculation, according to (1).
Each node will then randomly decide whether to engage or not in the mission, based on this calculated probability [2]:

$$\text{prob}_i = \begin{cases} 
    \frac{g_i}{\sum_{j=1}^{n} g_j} \cdot p \cdot n, & \text{if } \text{prob}_i < 1 \\
    1, & \text{otherwise}
\end{cases} \quad (1)$$

And $g_i$ is the goodness calculated value of node $i$, which means how good the node is to engage in a given mission.

$p$ is how much percentage of the capable node will be selected to accomplish the mission.

$n$ is the number of neighbours with the required sensor.

In the other hand, the random value can be obtained from function (2):

$$\text{rand} = \text{random}(0,1) \quad (2)$$

This function which is a linear distribution random function, the random value can be acquired from 0 to 1.

And then there is a comparison between $\text{prob}_i(1)$ value and random value (2). If $\text{prob}_i$ is bigger than random value, the node will engage in the mission, or it will not engage.

The process describing the engagement of a node in a mission is shown in Figure 4.4.

![Mission Allocation Decision Mechanism](image)

**Figure 4.4. Mission Allocation Decision Mechanism**


## 5 Modelling and Architecture

### 5.1 System design

A WSN agent approach simulator is organized by two parts: simulation environment and agents. Simulation environment is programmed by OOP language. Agents are programmed by AOP language.

This chapter provides the overview of the system first, and then discuss how the SAMSON simulation environment was used and what has been modified in SAMSON in order to make it suitable for the specific cooperation needed to be implemented by local-agents and mission-agents. Next, agents design pattern are discussed. Lastly, each main function’s flow chart is discussed.

#### 5.1.1 System architecture

The simulation architecture is based on model, view and controller design pattern (MVC pattern). This work used SAMSON system architecture and simulated environment to test the agents.

![Diagram of WSN agent system overview](image)

**Figure 5.1. WSN agent system overview**

In Figure 5.1, model component is a simulated environment. The controller components are the agents. The viewer component is used to show the simulation result. The architecture is almost kept the same as SAMSON. This simulation has added new functions at each part of this architecture, which will be discussed in detail in the following sub-chapter.
5.1.2 Simulation environment

SAMSON simulation environments are used to simulate SNs’ hardware operation, (energy consumption, sensing operation, sending operation, etc.). SNs communicate with each other within the communication range, based on the remaining energy. The simulation environment also simulates wave propagation, temperature, light and so on. For more details, interested readers are directed to [5].

![Diagram of WSN agent system main class](image)

Figure 5.2. WSN agent system main class

Figure 5.2 presents the main class diagram of a simulation environment, but differently from SAMSON, this thesis has two other additional classes: “missionAgent” and goodness. Once one mission-agent decides to rest on the one local-agent, the program will generate one corresponding mission-agent class and one corresponding goodness class in JAVA. Goodness is a value which could be used to weigh the capability of SNs according to mission requirement. Goodness could be reserved in the mission-agent class in order to request and use goodness value. There are also some corresponding changes in other classes, which are presented in the following test or diagram.
5.2 Agent designs overview

Figure 5.3. Agent system overview diagram

Figure 5.3 presents the agent system overview diagram according to the Prometheus design methodology [31], in which the functions of each agent are shown. The mission protocol shown in this figure is implemented in JAVA, together with the environment.

5.3 Mapping from the theory to the practice using BDI

5.3.1 Local-agent code and simulated environment

This thesis adopts agents which implement the BDI agent model. Beliefs represent the information which is available to an agent. Desires are all the possible states of affairs that the agent might like to accomplish. Intentions are the states of affairs that the agent has decided to work towards [4]. Each BDI-model agent has four main components: the belief base, the plan library, the set of events and the set of intentions. The belief base is used for storing the gathered information (belief). In order to understand how beliefs work in the programme, the JASON language structure has to be discussed first.

A brief JASON language structure is shown below:

\[
\text{Triggering event: context} \\
\text{<- body}
\]

Belief is interpreted as the input data of agents. And increase of belief is called “Belief Addition”. In the programme, it marks as: +belief. A new Belief addition generates a new
event. This event is stored into the set of events component, and the set of events is using first-in first-out (FIFO) strategy to store events. And then an agent can choose the plan from its own plan library based on the set of event and the belief base.

To map the concepts from Chapter 4 to BDI-model, the exchanged data (for example: agent’s reply) in [2] it is implemented as “belief” in BDI-model agent; after the communication between agents (for example: in the Figure 4.3- step 3), it is implemented as one “trigger event” in BDI-model agent; at last, the agents’ actions (for example in the Figure 4.3- step 4.3a (discard), 4.3b (clone), 4.1d(move-clone), etc), are implemented as “body” in BDI-model agent. Below, the agent’s code is given in order to show the mapping from theory to the practical implementation using the BDI-model.

Listing 2 provides an example of initial status of a local-agent.

```
/* Initial beliefs and rules */
/* Initial goals */
!run.
/* Plans */
+!run : true
  <- .at("now + 1 s", "startToRun").
```

In the code presented in Listing 2, the initial belief base is an empty belief base and initial setting of initial goals is “!run”. When the code is running, the programme is executed by using the top-down method, so the programme code will check whether it has an initial belief or not. But there is an initial goal which is marked as +!run, and then the programme can find out the only corresponding plan: +!run:true from the plan library based on the current belief (!run) and the set of events (+!run).

The JASON structure is shown as follows:

```
+event : context
  <- intentions.
+!run:true.
```

In the initial status of agents, the interpreter do not check the context part (for example: true is the context part of the event: +!run) of a plan and the executable section: body part.

After agents start to run, the JASON interpreter begins to check the context part, as the Boolean value of context part is always true, then this plan could be selected from the plan library and the programme starts to run the executable section (body/intention).
WSN Setup by means of Software Agents

Listing 3. Example code of internal action

```
.at("now + 1 s", "+startToRun").
```

In the example of Listing 3, the executable section is called “internal action”. It means that after starting to run one second, the program will add the belief: startToRun to the belief base of local-agent and generate the event: +startToRun. And then the program selects the corresponding plan from the plan library, due to the context part is set as true, the program will run into the body part, which is presented in Listing 4.

```
Listing 4. Code of +startToRun

+startToRun : true
<- !readTemp;
-temp(_);
.abolish(temp(_));
!run.
```

Firstly, the programme executes: !readTemp, and generates the event: +!readTemp, and then checks the plan library, the program will run the code presented in Listing 5.

```
Listing 5. Code of !readTemp

+!readTemp : true
<- !senseEnvironment;
!sendTemp.
```

At this time, the plan: +startToRun has been suspended, and then executes the plan: !SenseEnvronement. The plan: +!readTemp should be suspended, the program will execute the plan: !SenseEnvronement and find the corresponding plan (Listing 6).

```
Listing 6: Code of !SenseEnvronement

+!senseEnvironment : true
<- sense(0). //0 = Temp sensor
```

Sense(0) is an external action (EA) which is inherited from SAMSON, also the following EAs are inherited from SAMSON: setMCUStatus, setRadioState, setTXPower, sendTX().

After executing the EA, the programme will switch to the simulated environment which is implemented by JAVA as follows:
Figure 5.4. The process of OOP part

Figure 5.4 represents the process of the simulated environment. The controller: NetEnvironment class is the connecter of the agent and the simulated environment. By using the executeAction method, the sense action is detected and performed. To execute the sense action, the JAVA classes in Figure 5.4 are invoked.

In the tile class, the simulated environment’s temperature of each tile is found, and then the temperature is sent back to the NetEnvironment class. Finally, this information is returned to the agent as a new percept through the updatePercepts(agName) method (Listing 7).

Listing 7. Code of updatePercepts(agName)

```java
downloadedCode
void updatePercepts(String agName) {
    …
    Literal physMsg = Literal.parseLiteral("temp(temp)");
    addPercept(agName, physMsg);
    …
}
```

Figure 5.4 illustrates that the NetEnvironment class delivers a belief of temperature named temp(temp) to a specific local-agent. In this case this specific local-agent has a belief: +temp(temp) in its belief base. Then this local-agent returns to the its suspended plan: +!readTemp(Listing 5).

In this moment, the remaining plan of Listing 5: !sendTemp is executed (Listing 8).
Because there is a belief addition: $+\text{temp}(\text{Temp})$ in the belief base, the context of $+!\text{sendTemp}$ plan is met. In this condition, the plan: $+!\text{sendTemp}$ is executed. The suspended plans of local-agent are performed sequentially in this way. At last, the first plan: $+\text{startToRun}$ is backtracked (Listing 4).

Due to the last sentence: $!\text{run}$, the local-agent repeats the mentioned actions.

For the code in detail, check the appendix: local-agent.

### 5.3.2 Mission-agent code and Sequence diagrams

The beginning of mission-agent execution is addressed by the code presented in Listing 9.

**Listing 9. Code of initial part of mission-agent**

```pascal
/* Initial beliefs and rules */
missionID1.

/* Initial goals */
!run.

/* Plans */
+!run : true
  <-
  ..at("now + 40 s", "+\text{checkCurrentLocation}").
```

The initial belief has already stored the mission-agent’s mission identifier: Mission ID, so there is an initial goal: $!\text{run}$ will generate the event: $+!/\text{run}$.

When the plan: $+!/\text{run}$ is performed, the mission-agent needs to wait 40 seconds, then the mission-agent adds a belief: $\text{checkCurrentLocation}$. The duration is set with 40 seconds which is the maximum duration to generate a complete neighbour node list in this simulation. After 40 seconds, the belief $\text{checkCurrentLocation}$ is added, and a corresponding event would be performed.

The following part presents the code about message exchange based on SAMSON (Listing 10).
Listing 10. Code of the message exchange

```java
node.receive(m);
String sourceNodeName = m.getOriginator();
SensorNode sourceNode = getSensor(sourceNodeName);
node.setNeighbours(sourceNode);
node.setHasNeighbours(true);
```

The following part is presenting the code of mission-agent and the flow chart of it.

- **Set 1: Mission dissemination**

  Once a mission-agent X is sent to the network and reaches the first sensor node, it interacts with the local-agent of this node to check its location.

  In the mission-agent, `askMAlocation` is used to check location. *(MA means mission-agent).*

Listing 11. Code of the checkCurrentLocation

```java
+checkCurrentLocation : true
<- .my_name(MA);
   askMAlocation(MA);
```
The figure 5.5 shows the flow of the initial execution of the mission-agent in which it requests its location. When this flow finishes, updateMissionAgentPercept method will be performed. In Listing 12, if the external action (askMALocation) is executed, the updateMissionAgentPercept method will add a belief – location(X, Y, La) – to the mission-agent.

Listing 12. Code of updateMissionAgentPercept

```java
if (externalAction.equalsIgnoreCase("askMALocation")) {
    int row = ma.getRow();
    int col = ma.getCol();
    String nodeID = ma.getMACurrentLA().getNodeID();
    Literal maLocation = Literal.parseLiteral("location(" + row + ", " + col + "," + nodeID + ")");
    addPercept(agName, maLocation);
}
```

In Listing 13, a new belief + (location(X, Y, LA)) is sent to the mission-agent. The parameters X, Y represent the current coordinates and the LA represents the ID of the visiting local-agent. Then the next function askLATypeAndMAID will be performed.
Listing 13. Code of `askLATypeAndMAID`

```erl
+location(X, Y, LA) : true
<- askLATypeAndMAID(LA, temperature, missionID1).
```

Figure 5.6 shows a capability diagram for checking the local-agent type, which represents the type of the sensor node which the local-agent governs, and mission ID. A capability diagram is used to provide a detailed explanation of an action as presented in [31].

![Agent capability diagram](image)

**Figure 5.6. Agent capability diagram-check local-agent type and mission ID**

The sequence diagram of the capability function (Figure 5.7) as follows:
The programme adds the belief: \texttt{+missionStatus(LA)} to mission-agent, LA represents the corresponding local-agent ID. Afterwards it is used to find out the corresponding data from JAVA class, as presented in Listing 14.

\begin{verbatim}
Listing 14. Code of Literal missionStatus

Literal missionStatus = Literal.parseLiteral("missionStatus( " + localAgent + ")");
addPercept(agName, missionStatus);

\end{verbatim}

When a mission-agent visits an SN, the mission-agent gets the information from the local-agent about this SN, as described in [2]. This information includes the SN’s type and if the local agent knows about the mission ID which the mission-agent carries.

This information will be added into mission-agent by means of belief, as presented in Listing 15.
According to the type of SNs, it will generate belief of four different combinations as follows:

1) \(+typeAccept\) and \(+newMission\)
   It means the type of SN is correct and it has no mission ID.

2) \(+typeNotAccept\) and \(+newMission\)
   It means the type of SN is incorrect and it has no mission ID.

3) \(+typeAccept\) and \(+hasMission\)
   It means the type of SN is correct and it has a mission ID.

4) \(+typeNotAccept\) and \(+hasMission\)
   It means the type of SN is incorrect and it has a mission ID.

Then, the program will find out the corresponding data according to the stored data in belief base at this time.

For instance, \(+\text{missionStatus } (\text{Local agent ID, +typeAccept and +newMission})\) find out the corresponding data.

---

**Listing 15. Code of addPercept**

```plaintext
addPercept(agName, typeAccept);
addPercept(agName, newMission);
```

---

**Listing 16. The example code of four cases**

```
// Case 1: Has Right Sensor And Not has Mission
+missionStatus(CLA) : typeAccept & newMission
   <-
       ....

// Case 2: Not Has Right Sensor And Not Has Mission
+missionStatus(CLA) : typeNotAccept & newMission
   <-
       ....

// Case 3: Has Right Sensor And Has Mission
+missionStatus(CLA): typeAccept & hasMission
   <-
       ....

// Case 4: Not Has Right Sensor And Has Mission
+missionStatus(CLA): typeNotAccept & hasMission
   <-
       ....
```
WSN Setup by means of Software Agents

According to four different addition manners of belief, the program will emerge four different EAs:

1) requestNodeStatus(CLA).
2) moveClone(CLA).
3) mergeNodeStatus(CLA).
4) kill_agent(CLA).

Next, each EA’s procedure will be presented:

1) In the first case, if the information of SN’s type is the same as the mission requested type, a percept: typeAccept is created. In the meantime, another percept: A newMission is generated, when the information of mission ID is not identical with the mission ID of the mission-agent.

An action: requestNodeStatus will be processed (Figure 5.8). According to the function that defines the goodness (3), this action acquires SN’s battery and accuracy of current SN status, and calculates the goodness value.

![Figure 5.8. Mission dissemination capability 1](image_url)
In Figure 5.9, after `requestNodeStatus()` method was executed, the goodness value can be obtained by executing `calculateGoodness()` method.

**Listing 17. Code of the calculation of goodness value**

Calculation of goodness value:

```java
public double calculateGoodness() {
    double goodness;
    goodness = (batteryHandler.getAvailableCapacity() / batteryMaxCapacity) * 0.25 + 
               (batteryHandler.getCurrentVoltage() / batteryMaxVoltage) * 0.25 + 
               (accuracy * 0.5);
    return goodness;
}
```

Due to SAMSON is playing a fundamental role in this simulation, the battery measurement method of SAMSON is inherited. In the goodness value calculation process, the coefficients of `AvailableCapacity` and `CurrentVoltage` are both set with 0.25.

After `saveGoodness()` method was executed, the current mission-agent obtains the goodness value of the SN which is being visited. Finally, `updateMissionAgentPerceptes()`
WSN Setup by means of Software Agents

method produces (Figure 5.10) next belief: \( +\text{findNeighbourNode} \).

➢ **JAVA part:**

```java
Listing 18. Code of \( +\text{findNeighbourNode} \) in JAVA part

Literal findNeighbourNode =
Literal.parseLiteral("findNeighbourNode");
addPercept(agName, findNeighbourNode);
```

➢ **Agent part:**

```java
Listing 19. Code of \( +\text{findNeighbourNode} \) in Agent part

/* Case 1: Has right sensor but not has mission: clone*/
+\text{findNeighbourNode} : true
<- clone(MA).
```

![Figure 5.10. The capability diagram of \( \text{findNeighbourNode} \)](image.png)
In Figure 5.11, by using the method `setStayLocation()`, a mission-agent should be treated as a tied mission-agent, which means it will leave a clone in the SN. By using the methods of
WSN Setup by means of Software Agents

getNeighbours() and newMissionAgent(), this SN’s neighbours will acquire this mission-agent. Finally, the updateMissionAgentPercepts() method produces the belief: +cloneAgent().

![Figure 5.12. The capability diagram of cloneAgent()](image1)

**Listing 20. Code of @cloneAgent [atomic]**

```plaintext
@cloneAgent [atomic]
+cloneAgent(MANAME) : true
<- 
 .create_agent(MANAME, "src/asl/missionAgentApproach/mission_agent.asl").
```

@cloneAgent [atomic] is the name of a plan. The reason of using [atomic] is that when the agent executes this plan, it is not suspended by any other programmes.

To create an agent, an internal action .create_agent is used(Figure 5.12)

2) In the second case (Figure 5.13), if the information of SN’s type is not exactly identical to the mission requested type, a percept: typeNotAccept is created. In the meantime, another percept: newMission is generated, when the information of mission ID is not same with the mission ID of the mission-agent.

Then an action: moveClone will be processed.

![Figure 5.13. Mission dissemination capability 2](image2)
WSN Setup by means of Software Agents

In this case, the action which will be chosen is move-clone. As there is only one difference between clone and move-clone, which is that with the move-clone the mission-agent will not occupy the SN that it is visiting, i.e. the agent will be discarded from the current node after the action is completed. So in this part, the process of the move-clone is not shown.

3) In the third case (Figure 5.14), if the information of SN’s type is identical to the mission requested type, a belief: +typeAccept is created. In the meantime, another belief: +hasMission is generated, when the information of mission ID is identical to the mission ID of the mission-agent.

Then an action: mergeNodeStatus will be process.

![Diagram](image)

**Figure 5.14. Mission dissemination capability 3**
In Figure 5.15, mission-agent A is the most recently arrived mission-agent, and mission-agent B is the mission-agent that was already in the SN. Mission-agent B merges and retains the received information. After the merging process, by using the MissionAgentPercepts() method, a belief: \texttt{+mergeComplete} is created. Then the mission-agent A will be discarded by means of executing internal action: \texttt{.kill_agent(ELIMINATEAGENT)}, as presented in Listing 21.

\begin{verbatim}
Listing 21. Code of the mergeComplete

+mergeComplete(ELIMINATEAGENT) : true
    <-
        .kill_agent(ELIMINATEAGENT).
\end{verbatim}

4) In the fourth case (Figure 5.16), if the information of SN’s type is not same with the mission requested type, a percept: \texttt{typeNotAccept} is created. In the meantime, another
percept: \textit{hasMission} is generated, when the information of mission ID is same with the mission ID of the mission-agent.

Then an internal action: \textit{.kill\_agent} will be process.

![Diagram](image)

\textbf{Figure 5.16.} Mission dissemination capability 4

Because \textit{.kill\_agent} is an internal action, it should not be shown on mission dissemination capability diagram (Figure 5.15). A piece of code with respect to this process is shown in Listing 22.

\begin{verbatim}
Listing 22. Code of the .kill\_agent
+missionStatus(CLA): typeNotAccept & hasMission
<- kill_agent(CLA).
\end{verbatim}

When the first mission-agent is injected into a SN, at last it will be as a resident mission-agent, the mission-agent will add a belief: \textit{+nodeAllocation} and also start a timer to activate the mission allocation (Figure 5.17) when it expires. This timer is set with absolute time, and not the simulated time due to the way the simulator handles this timer. The simulations incur in heavy processing, thus taking long time to be performed. This requires that this timer to be set to 7 minutes (absolute time).

A piece of code involved in this process is demonstrated in Listing 23.

\begin{verbatim}
Listing 23. Code of the startAllocation
+nodeAllocation : true
<- .at("now + 7m", "+startAllocation").
\end{verbatim}
Listing 24 presents the code for mission-agent when it starts the mission allocation.

Listing 24. Code of the start missionAllocation

```plaintext
+startAllocation : true
  <-
    .my_name(NAME);
    missionAllocation(NAME).
```
Figure 5.18. Sequence diagram for mission allocation

In Figure 5.18, the process of a weight probability test, the calculated \( \text{prob}_i \) (1) value needs to be compared to a random value (2). If an SN’s \( \text{prob}_i \) value is better than a random value, this SN will engage in the mission, otherwise, the mission-agent on this SN will discard. By using the \( \text{updateMissionAgentPercepts()} \) method, a belief: \textit{missionFailed} is created to discard the non-engaged mission-agent.
Parts of the code involved in this process show in Listing 26.

**Listing 25. Code of the test**

In the condition of probability value > random value:

```java
if (prob > randomNodeCheckValue) {
    externalAction = externalAction + "Join";
    this.mainEnv.updateMissionAgentPercepts(agName, externalAction);
    node.setJoinMission(true);
}
```

In the condition of probability value < random value:

```java
else if (prob <= randomNodeCheckValue) {
    externalAction = externalAction + "Discard";
    this.mainEnv.updateMissionAgentPercepts(agName, externalAction);
    node.setHasLocalMA(false);
}
```

Discard a non-engaged mission-agent:

```java
+missionFailed(MANAME) : true
<- .kill_agent(MANAME).
```

When the mission is executed successfully, the agent will execute the code presented in Listing 26.

**Listing 26. Code of the missionSuccess**

```java
/* Mission Plan*/
+missionSuccess(LA) : true
<- .send(LA, unachieve, sendTemp).
```
6 Experiments

In this chapter, experiments are presented, which include WSN’s mission dissemination and the mission allocation processes that were presented above. These experiments were implemented as simulations using JASON to perform the agents and Java to perform the environment part. JASON was chosen because of its support that has all the main elements for expressing reactive planning systems with BDI notions. Moreover, it is based on JASON, which is available Open Source under GNU LGPL [4]. The model of SAMSON which was mentioned above plays a fundamental role in the simulations. It provides a structure about how to construct an environment. It also provides a simulator of a hardware model named “TMoto Sky unit” and structure of ad hoc network.

The whole environment has the dimensions of 50 x 50 tiles, in which each tile represents 1 x 1 meter and each SN occupies one tile. The maximum communication range of each node is set to 12 meters, which means 12 tiles. Based on SAMSON, this range can be reduced with the nodes’ battery level decrease. There are 100 SNs with their local-agents in this environment. Every SN is positioned randomly, as well as their properties considered for the mission allocation, i.e. the sensor device accuracy and the remaining battery level. These properties are randomly distributed from 0 to 100% of their full capability.

A snapshot of the beginning phase of the simulation is seen in the Figure 6.1.

Figure 6.1. The beginning of the simulation
WSN Setup by means of Software Agents

In Figure 6.1, two kinds of SN are presented: temperature SN, represented by green color dots and light SN, represented by yellow color dots, aimed at sensing different elements from environment, i.e. temperature and light. Each kind of SN takes a random value of the entire number of SNs, which means that the number of temperature sensation SNs and number of light sensation SNs of each simulation are different. The number of SNs of each kind is set as almost 50% of the whole number of SNs.

In the performed simulations, the temperature SNs are requested by mission-agents to perform a sensing mission. The first mission-agent injects into local-agent1 which is represented by the color of cyan, on the upper right part of the Figure 6.1.

In the process of mission dissemination, all SNs should be visited by a mission-agent. Figure 6-2 illustrates this process. The colors of SNs turn to blue when the SNs have been visited at least once by a mission-agent. Figure 6.2 shows that all nodes in the environment have already been visited by the mission agent.

![Figure 6.2. Mission dissemination](image)

After the process of mission dissemination, every SN takes an autonomous decision to judge if it should engage in a specific mission or not, which is called mission allocation. This decision process obeys the flow path of Figure 4.4. Figure 6.3 shows the result of simulation, the SNs that have been selected to be engaged in the mission turn to red. If any SN runs out of battery (lower than 20%), the colour of this SN turns to black.
6.1 Simulated Mission

Each SN can be engaged in a specific mission or not is decided by the result of the comparison between \(prob_i\) and \(rand\) calculated by (1) and (2) respectively. \textit{Goodness} is an evaluation criterion which indicates how good an SN is to perform a given mission. In this simulation, the goodness function is defined as (3) based on the theoretical reference model, where \(ac\) is the sensor accuracy and \(e\) is the remaining energy level. The coefficients in front of \(ac\) and \(e\) represent the weight factors for importance of both parameters.

\[
g = 0.5ac + 0.5e \quad (3)
\]

In the case of the WSN modeled in this thesis, a percentage of all the eligible SNs is specified by each mission, in the fourth field of Figure 4-1. The expected result is that the number of SNs effectively engaged in the mission should be as close as possible to the desired number of SNs specified in the mission.

With regard to this simulation, different setups were performed in order to test if the decision method mentioned above is useful to get a number of nodes close to the optimal number of SNs required by a mission (the number specified in the sensing mission).

With the environment setup parameters presented above, the simulations base on three different percentages of all the eligible SNs, which are 20%, 50% and 80%.
6.2 Results and Discussion

Figure 6.4 represents the simulation results of number of the engaged nodes when the variation setting is 20% (20% of the number of eligible SNs required by the mission).

Figure 6.4: The simulation results of the 20% target percentage
Figure 6.5 is the result of the twentieth simulation which is the closest result to the mission requirement. In this simulation 11 SNs were chosen to participate in the mission and the result is 21.5% of the eligible nodes.

![Figure 6.5. The result of the 20th simulation](image-url)
Figure 6.6 represents the simulation results of number of the engaged nodes when the variation setting is 50%.

Figure 6.6. The simulation results of the 50% target percentage
WSN Setup by means of Software Agents

Figure 6.7 is the result of the twelfth simulation which is the closest result to the mission requirement. In this simulation 26 SNs were chosen to participate in the mission and the result is 52% of the eligible nodes.

Figure 6.7. The result of the 12th simulation
Figure 6.8 represents the simulation results of number of the engaged nodes when the variation setting is 80%.

Figure 6.8. The simulation results of the 80% target percentage
WSN Setup by means of Software Agents

Figure 6.9 is the result of the twenty-seventh which is the closest result to the mission requirement. In this simulation 40 SNs were chosen to participate in the mission and the result is 81.6% of the eligible nodes.

![Figure 6.9. The result of the 27th simulation](image)

(a)

(b)
WSN Setup by means of Software Agents

Figure 6.10 presents the comparison between each target percentage to the average simulation percentage of engaged SNs related to it.

![Figure 6.10. The comparison between target percentages to average simulation percentage](image)

Figure 6.10 indicates that by using the \textit{prob}_t function, the SNs can automatically decide whether to participate in a mission or according to their goodness values and the target percentage of this mission, having results very close to this target value. Although these simulation results are not precisely the same as the target value (the distances between the average simulation percentages and target percentages are not negligible), they could be improved by enlarging the number of SNs in the simulation environment (1000 SNs).

Table 6.1 represents the statistical results of each variation setup which completes the result from Figure 6.10. In order to facilitate reader understanding, in Table 6.1, the target number is presented together with the average number of nodes actually engaged in the mission, as well as the corresponding standard deviation values. The standard deviations are not too large, which shows the stability of the proposed method.

<table>
<thead>
<tr>
<th>Setup Variation</th>
<th>Target Number</th>
<th>Average</th>
<th>Standard Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>20%</td>
<td>10</td>
<td>12.70</td>
<td>4.91</td>
</tr>
<tr>
<td>50%</td>
<td>25</td>
<td>27.66</td>
<td>6.73</td>
</tr>
<tr>
<td>80%</td>
<td>40</td>
<td>41.60</td>
<td>4.83</td>
</tr>
</tbody>
</table>

Table 6.1. Statistical results of engaged SNs
As source of comparison to proposed intelligent decision approach, Figure 6.11 presents the simulation results of number of engaged nodes for the variation of setting $prob_i$ to a random value, instead of with the goodness calculation as presented above.

![Figure 6.11. The simulation results of random value](image)

Considering the random comparison, the percentage of engaged nodes would be both approximate as 50%. Thus the result is just about mathematic probability. It does not address the goal number of desired number of nodes that should be engaged, as it will always provide results close to 50% in average.

The presented assessed metric provides information about the number of nodes engaged to perform the mission, but the quality of engaged nodes is also important. Figure 6.12 presents the averages of the goodness value for selected sensor nodes and the optimized average goodness value for each setup variation and each run. The dotted lines represent the average of obtained results.
WSN Setup by means of Software Agents

(a): 30 run times for 20% variation setup

(b) 30 run times for 50% variation setup
Observing the differences in Figures 6.12 (a), (b) and (c), the average of obtained results decreases with the required percentage rising. These downward trends were coherent with mathematics. As the goodness value of each node is decided by the battery level and accuracy which are assigned with random values, if more SNs were requested to engage in a mission, more SNs with lower goodness values would be selected. That caused lower average of obtained results is relation to higher percentage demand. And about each run’s optimized average goodness value, the optimum has to be always higher, because the optimum is composed of the best nodes to engage in the mission, so it is better that the value achieved with the heuristic in any case.
Table 6.2. Average for the goodness values and the corresponding standard deviation

<table>
<thead>
<tr>
<th>Setup Variation</th>
<th>Average optimized Goodness value</th>
<th>Average Goodness Values</th>
<th>Standard Deviation Of Average Goodness Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>20%</td>
<td>0.872</td>
<td>0.788</td>
<td>0.062</td>
</tr>
<tr>
<td>50%</td>
<td>0.777</td>
<td>0.691</td>
<td>0.062</td>
</tr>
<tr>
<td>80%</td>
<td>0.697</td>
<td>0.607</td>
<td>0.053</td>
</tr>
</tbody>
</table>

Table 6.2 complements Figure 6.12. The second column presents the average optimized goodness value while the third and the forth columns respectively presents the goodness values achieved by the applied proposed method and the corresponding standard deviation for each setup variation.
WSN Setup by means of Software Agents

7 Related Work

7.1 Agilla

Agilla [32] is a middleware for WSN that provides a mobile-agent style of programming. Their applications consist of mobile agents that can proactively migrate their code and state across the network. Mobile software agents offer more flexibility by allowing applications to control the way they spread. Also they present an approach to setup the SNs.

This work is very similar to our approach, but there is an important difference between these two works. Our approach uses AOP language to build two BDI-model agents which interact and have intelligent decision capabilities, while in their work the agents are rather simple, being only reactive and not as deliberative as ours.

7.2 Adaptive Probabilistic Epidemic Protocol (AEPAP)

In [33], Tan and Munro present the approach to solve the problem of message dissemination using multi-level WSN, and this approach is based on the epidemic routing protocol. It proposed an adaptive probabilistic epidemic protocol (APEP) which allows a node to decide whether to respond to a broadcast based on simple information, for example the number of neighbours of the broadcasting node and hence suppresses redundant messages and lowers the contention/collision level.

This case is similar to our case, but there is a difference between these two works. In their case, the decision must take into account the neighbour’s information. On the other hand, in our case it must monitor the neighbour’s activities to decide to forward or not, i.e. a sensor node will just forward a mission-agent if none of its neighbours did not forward it before.

7.3 Mobile Agent Based WSN (MAWSN)

Chen. et al [34] proposed the architecture is called mobile agent based WSN (MAWSN) which uses the mobile agent for reducing and aggregating data. A proposal is to aim to solve the problem of the overwhelming data traffic. The reason to develop this architecture is that source nodes are close to each other, and in the sensing data, the considerable amount of redundancy must be considered. Hence, the source nodes generate a large number of traffic on
WSN Setup by means of Software Agents

the wireless channel, which not only wastes the bandwidth of the scarce wireless, but also consumes a lot of battery energy.

This architecture is presented as following Figure 7.1.

![Figure 7.1. Architecture of MAWSN (figure adapted from [34])](image)

In our work energy consumption is not directly assessed, neither are the number of sent messages. It assumes that by making the sensor nodes perform autonomous decisions, they do not have to communicate with each other after the mission dissemination and like this save energy by avoiding communication. Their work avoids communication by means of agents that help in sensor data aggregation, as in the data retrieval process. It is possible to state that both works are complementary, as each of them try to address the problem of diminishing the exchange of messages in two different processes, ours in the mission dissemination and allocation, while theirs is in the sensing mission performance and reporting results.
7.4 Mobile Agent Platform for Sun SPOTs (MAPS)

In [35], Francesco et al present the design, implementation and experimentation of MAPS (Mobile Agent Platform for Sun SPOTs), an innovative Java-based framework for WSNs based Sun SPOT technology which enables agent-oriented programming of WSN applications. They proposed mobile agent as an effective paradigm to program WSN application. By using MAPS, a WSN application could be structured as a set of static and mobile agent distributed on sensor nodes supported by a component-based agent execution engine which provides some basic services such as message exchange, agent creation, agent cloning, etc.

Comparing with our work, ours also have these actions above mentioned using static and mobile agent, their simulation shows some performance penalty mainly due to time-consuming operations. Our simulation shows some agents performance due to energy-consuming operations, but no result in relation to timing properties.

7.5 Simulating BDI-based WSNs

Morris et al. [36] presents a BDI-agent model for WSNs. The agents in the sensor nodes can decide whether they have to perform and accomplish the sensing mission. Their work does not invest much in the reasoning mechanism itself, while it can be found in our work. However, their work provides the theoretical basis for ours, as it can be used to model the agents that we use.
8 Conclusions and Suggestions to Future Work

This thesis focuses on the problems of WSN setup for sensing missions including the mission dissemination and the mission allocation. In order to address these problems, two BDI-model agents (mission-agent and local-agent) are defined to implement a theoretical decentralized mechanism for WSN setup. This approach provided a flexible way to disseminate a sensing mission to the sensor nodes, and provided autonomous capabilities to the sensor nodes, so that they are able to decide autonomously, whether to by themselves about to engage in a given mission or not. Using the move-clone and clone actions, the mission agent obtains information about the goodness values from the nodes visited during the mission dissemination. After completing the mission dissemination, the effective decision mechanism use this acquired data to perform the mission allocation. Simulation results of the mission dissemination and allocation were presented and discussed.

In regards of the simulation work, SAMSON [5] was used as a simulation environment to test simulated agents. Results were expressed in terms of measurements of metrics related to the theoretical model used, which focus on the number of sensor nodes engaged in the mission and the quality of the engaged nodes compared to the mission specification. Experiment results demonstrated an effective mission allocation after a comparison between the number of engaged nodes and the number specified in the sensing mission.

In terms of secondary contributions of this thesis, it can be mentioned the extension of SAMSON by defining the mobile agents using JASON, and the adoption of an intelligent mechanism to perform the mission allocation decision making by the introduced mobile agent.

However, there are many aspects that could be considered in the future as follows: 1) Simulation scale: the limited number of SNs used in the simulations was due to the fact that using more than 100 SNs, the simulation became too slow and the computer can even freeze. The reasonable explanation for this is that this programme is not optimized, so enhancements in the SAMSON and in the simulations themselves could help to increase the simulations’ scale. 2) Different node distributions could also be considered besides the random one used in the presented experiments. It would be useful to test the approach in relation to its robustness in spite of unfavourable conditions, for instance, the presence of regions in which the sensor nodes have very similar conditions, which in principle may influence the decision of the sensor nodes.
9 References


http://lama.disi.unitn.it/page.php?34


[8] ZigBee: http://www.zigbee.org/


WSN Setup by means of Software Agents


[27] Chrysanthi, E., Georgakarakou and Anastasios A.: *Software Agent Technology: an Overview Application to Virtual Enterprises*. Economides Information Systems Department University of Macedonia.


WSN Setup by means of Software Agents


Appendix A: Source code for agents’ behaviour

- Local-agent

/* Initial beliefs and rules */
/* Initial goals */

!run.

/* Plans */

+!run : true
   <- .at("now + 1 s", "+startToRun").

+!readTemp : true
   <- !senseEnvironment;
      !sendTemp.

+!senseEnvironment : true
   <- sense(0). //0 = Temp sensor

+!sendTemp : temp(Temp)
   <-
      setMCUState(0);
      setRadioState(3);
      setTXPower(-24);
      sendTX("dc", Temp, 1,1);
      setRadioState(2);
      setMCUState(2).

+startToRun : true
   <-
      !readTemp;
      -temp(_);
      .abolish(temp(_));
      !run.

WSN Setup by means of Software Agents
WSN Setup by means of Software Agents

**Mission-agent**

/* Initial beliefs and rules */
missionID1.

/* Initial goals */
!run.

/* Plans */
+!run : true
   <-
       .at("now + 40 s", "+checkCurrentLocation").

+checkCurrentLocation : true
   <-
       .my_name(MA);
       askMALocation(MA).

//LA = local_agent
+location(X, Y, LA) : true
   <-
       askLATypeAndMAID(LA, temperature, missionID1).

// Case 1: Has Right Sensor And Not has Mission
+missionStatus(CLA) : typeAccept & newMission
   <-
       requestNodeStatus(CLA).

// Case 2: Not Has Right Sensor And Not Has Mission
+missionStatus(CLA) : typeNotAccept & newMission
   <-
       moveClone(CLA).

// Case 3: Has Right Sensor And Has Mission
+missionStatus(CLA): typeAccept & hasMission
   <-
       mergeNodeStatus(CLA).

// Case 4: Not Has Right Sensor And Has Mission
+missionStatus(CLA): typeNotAccept & hasMission
WSN Setup by means of Software Agents

```prolog
<- .kill_agent(CLA).
/* Case 1: Has right sensor but not has mission: clone*/
+findNeighbourNode : true
  <- clone(MA).

@cloneAgent [atomic]
+cloneAgent(MANAME) : true
  <- .create_agent(MANAME, "src/asl/missionAgentApproach/mission_agent.asl").

/* Case 2: Not has right sensor and not has missionID: moveClone */
@moveCloneAgent [atomic]
+moveCloneAgent(MANAME) : true
  <- .create_agent(MANAME, "src/asl/missionAgentApproach/mission_agent.asl").

/* Case 3: Has Right Sensor and Has Mission: mergeInfo and then discard */
+mergeComplete(ELIMINATEAGENT) : true
  <- .kill_agent(ELIMINATEAGENT).

/* Node Allocation*/
+nodeAllocation : true
  <- .at("now + 7 m", "+startAllocation").
+startAllocation : true
  <- .my_name(NAME); missionAllocation(NAME).

+missionFailed(MANAME) : true
  <- .kill_agent(MANAME).
/* Mission Plan*/
+missionSuccess(LA) : true
```

92
WSN Setup by means of Software Agents

<- .send(LA, unachieve, sendTemp).
## Appendix B: Raw data results from simulations

- The simulation results of the 20% target percentage

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.1</th>
<th>Simu.2</th>
<th>Simu.3</th>
<th>Simu.4</th>
<th>Simu.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>48</td>
<td>54</td>
<td>51</td>
<td>53</td>
<td>47</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>14</td>
<td>17</td>
<td>7</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>Result</td>
<td>29.2%</td>
<td>31.5%</td>
<td>13.7%</td>
<td>24.5%</td>
<td>40.4%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.6</th>
<th>Simu.7</th>
<th>Simu.8</th>
<th>Simu.9</th>
<th>Simu.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>49</td>
<td>53</td>
<td>56</td>
<td>49</td>
<td>51</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>16</td>
<td>5</td>
<td>18</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>Result</td>
<td>32.6%</td>
<td>9.4%</td>
<td>32.1%</td>
<td>28.6%</td>
<td>15.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.11</th>
<th>Simu.12</th>
<th>Simu.13</th>
<th>Simu.14</th>
<th>Simu.15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>52</td>
<td>45</td>
<td>54</td>
<td>53</td>
<td>49</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>13</td>
<td>13</td>
<td>6</td>
<td>14</td>
<td>9</td>
</tr>
<tr>
<td>Result</td>
<td>25.0%</td>
<td>28.8%</td>
<td>11.1%</td>
<td>26.4%</td>
<td>18.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.16</th>
<th>Simu.17</th>
<th>Simu.18</th>
<th>Simu.19</th>
<th>Simu.20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>50</td>
<td>52</td>
<td>51</td>
<td>49</td>
<td>51</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>15</td>
<td>8</td>
<td>13</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Result</td>
<td>30.0%</td>
<td>15.4%</td>
<td>25.5%</td>
<td>16.3%</td>
<td>21.5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.21</th>
<th>Simu.22</th>
<th>Simu.23</th>
<th>Simu.24</th>
<th>Simu.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>48</td>
<td>57</td>
<td>54</td>
<td>47</td>
<td>55</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>13</td>
<td>24</td>
<td>9</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>Result</td>
<td>27.0%</td>
<td>42.1%</td>
<td>16.6%</td>
<td>34.0%</td>
<td>9.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.26</th>
<th>Simu.27</th>
<th>Simu.28</th>
<th>Simu.29</th>
<th>Simu.30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>53</td>
<td>56</td>
<td>47</td>
<td>55</td>
<td>47</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>21</td>
<td>19</td>
<td>15</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>Result</td>
<td>39.6%</td>
<td>33.4%</td>
<td>31.9%</td>
<td>10.9%</td>
<td>25.5%</td>
</tr>
</tbody>
</table>

From Table above, the average percentage of the number of engaged SNs is 24.91%.
The simulation results of the 50% target percentage

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.1</th>
<th>Simu.2</th>
<th>Simu.3</th>
<th>Simu.4</th>
<th>Simu.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>53</td>
<td>55</td>
<td>49</td>
<td>52</td>
<td>47</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>31</td>
<td>37</td>
<td>30</td>
<td>34</td>
<td>28</td>
</tr>
<tr>
<td>Result</td>
<td>58.5%</td>
<td>67.3%</td>
<td>61.2%</td>
<td>65.4%</td>
<td>59.5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.6</th>
<th>Simu.7</th>
<th>Simu.8</th>
<th>Simu.9</th>
<th>Simu.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>46</td>
<td>50</td>
<td>54</td>
<td>48</td>
<td>46</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>20</td>
<td>29</td>
<td>14</td>
<td>29</td>
<td>27</td>
</tr>
<tr>
<td>Result</td>
<td>43.4%</td>
<td>58.0%</td>
<td>25.9%</td>
<td>60.4%</td>
<td>58.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.11</th>
<th>Simu.12</th>
<th>Simu.13</th>
<th>Simu.14</th>
<th>Simu.15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>54</td>
<td>50</td>
<td>47</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>33</td>
<td>26</td>
<td>17</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>Result</td>
<td>61.1%</td>
<td>52.0%</td>
<td>36.2%</td>
<td>56.2%</td>
<td>58.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.16</th>
<th>Simu.17</th>
<th>Simu.18</th>
<th>Simu.19</th>
<th>Simu.20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>54</td>
<td>51</td>
<td>49</td>
<td>53</td>
<td>47</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>38</td>
<td>32</td>
<td>13</td>
<td>32</td>
<td>21</td>
</tr>
<tr>
<td>Result</td>
<td>70.3%</td>
<td>62.7%</td>
<td>26.5%</td>
<td>60.4%</td>
<td>44.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.21</th>
<th>Simu.22</th>
<th>Simu.23</th>
<th>Simu.24</th>
<th>Simu.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>49</td>
<td>51</td>
<td>54</td>
<td>46</td>
<td>53</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>32</td>
<td>35</td>
<td>19</td>
<td>26</td>
<td>39</td>
</tr>
<tr>
<td>Result</td>
<td>65.3%</td>
<td>68.6%</td>
<td>35.1%</td>
<td>56.5%</td>
<td>73.6%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.26</th>
<th>Simu.27</th>
<th>Simu.28</th>
<th>Simu.29</th>
<th>Simu.30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>47</td>
<td>49</td>
<td>52</td>
<td>50</td>
<td>48</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>27</td>
<td>28</td>
<td>32</td>
<td>19</td>
<td>27</td>
</tr>
<tr>
<td>Result</td>
<td>57.4%</td>
<td>57.1%</td>
<td>61.5%</td>
<td>38.0%</td>
<td>56.3%</td>
</tr>
</tbody>
</table>

From Table above, the average percentage of the number of engaged SNs is 55.22%.
The simulation results of the 80% target percentage

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.1</th>
<th>Simu.2</th>
<th>Simu.3</th>
<th>Simu.4</th>
<th>Simu.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>53</td>
<td>51</td>
<td>47</td>
<td>50</td>
<td>48</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>45</td>
<td>48</td>
<td>40</td>
<td>37</td>
<td>40</td>
</tr>
<tr>
<td>Result</td>
<td>84.9%</td>
<td>94.1%</td>
<td>85.1%</td>
<td>74.0%</td>
<td>83.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.6</th>
<th>Simu.7</th>
<th>Simu.8</th>
<th>Simu.9</th>
<th>Simu.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>54</td>
<td>52</td>
<td>49</td>
<td>46</td>
<td>48</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>46</td>
<td>30</td>
<td>47</td>
<td>33</td>
<td>41</td>
</tr>
<tr>
<td>Result</td>
<td>85.2%</td>
<td>57.7%</td>
<td>95.9%</td>
<td>71.7%</td>
<td>85.4%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.11</th>
<th>Simu.12</th>
<th>Simu.13</th>
<th>Simu.14</th>
<th>Simu.15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>54</td>
<td>48</td>
<td>46</td>
<td>55</td>
<td>49</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>42</td>
<td>44</td>
<td>33</td>
<td>50</td>
<td>42</td>
</tr>
<tr>
<td>Result</td>
<td>77.8%</td>
<td>91.7%</td>
<td>71.7%</td>
<td>90.9%</td>
<td>85.7%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.16</th>
<th>Simu.17</th>
<th>Simu.18</th>
<th>Simu.19</th>
<th>Simu.20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>50</td>
<td>47</td>
<td>49</td>
<td>53</td>
<td>46</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>45</td>
<td>34</td>
<td>45</td>
<td>46</td>
<td>42</td>
</tr>
<tr>
<td>Result</td>
<td>90%</td>
<td>72.3%</td>
<td>91.8%</td>
<td>86.8%</td>
<td>91.3%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.21</th>
<th>Simu.22</th>
<th>Simu.23</th>
<th>Simu.24</th>
<th>Simu.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>48</td>
<td>53</td>
<td>51</td>
<td>47</td>
<td>50</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>43</td>
<td>37</td>
<td>43</td>
<td>42</td>
<td>44</td>
</tr>
<tr>
<td>Result</td>
<td>89.6%</td>
<td>69.8%</td>
<td>84.3%</td>
<td>89.4%</td>
<td>88.0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.26</th>
<th>Simu.27</th>
<th>Simu.28</th>
<th>Simu.29</th>
<th>Simu.30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>52</td>
<td>49</td>
<td>54</td>
<td>50</td>
<td>49</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>39</td>
<td>40</td>
<td>46</td>
<td>38</td>
<td>46</td>
</tr>
<tr>
<td>Result</td>
<td>75.0%</td>
<td>81.6%</td>
<td>85.1%</td>
<td>76.0%</td>
<td>93.9%</td>
</tr>
</tbody>
</table>

From Table above, the average percentage of the number of engaged SNs is 83.27%.
The simulation results of the random value

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.1</th>
<th>Simu.2</th>
<th>Simu.3</th>
<th>Simu.4</th>
<th>Simu.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>53</td>
<td>46</td>
<td>53</td>
<td>51</td>
<td>57</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>38</td>
<td>29</td>
<td>31</td>
<td>26</td>
<td>22</td>
</tr>
<tr>
<td>Result</td>
<td>71.6%</td>
<td>63.0%</td>
<td>58.4%</td>
<td>50.9%</td>
<td>38.5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.6</th>
<th>Simu.7</th>
<th>Simu.8</th>
<th>Simu.9</th>
<th>Simu.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>55</td>
<td>48</td>
<td>51</td>
<td>53</td>
<td>44</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>27</td>
<td>15</td>
<td>36</td>
<td>31</td>
<td>28</td>
</tr>
<tr>
<td>Result</td>
<td>49.0%</td>
<td>31.2%</td>
<td>70.5%</td>
<td>83.6%</td>
<td>56.8%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.11</th>
<th>Simu.12</th>
<th>Simu.13</th>
<th>Simu.14</th>
<th>Simu.15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>56</td>
<td>48</td>
<td>45</td>
<td>47</td>
<td>51</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>16</td>
<td>29</td>
<td>19</td>
<td>25</td>
<td>29</td>
</tr>
<tr>
<td>Result</td>
<td>32.1%</td>
<td>60.4%</td>
<td>42.2%</td>
<td>53.2%</td>
<td>56.8%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.16</th>
<th>Simu.17</th>
<th>Simu.18</th>
<th>Simu.19</th>
<th>Simu.20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>54</td>
<td>49</td>
<td>53</td>
<td>43</td>
<td>55</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>18</td>
<td>22</td>
<td>26</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>Result</td>
<td>33.3%</td>
<td>44.9%</td>
<td>49.1%</td>
<td>72.1%</td>
<td>58.1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.21</th>
<th>Simu.22</th>
<th>Simu.23</th>
<th>Simu.24</th>
<th>Simu.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>52</td>
<td>50</td>
<td>54</td>
<td>44</td>
<td>55</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>33</td>
<td>19</td>
<td>39</td>
<td>28</td>
<td>23</td>
</tr>
<tr>
<td>Result</td>
<td>63.4%</td>
<td>38.0%</td>
<td>68.5%</td>
<td>63.6%</td>
<td>41.8%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Simu.26</th>
<th>Simu.27</th>
<th>Simu.28</th>
<th>Simu.29</th>
<th>Simu.30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount of Temp. SNs</td>
<td>50</td>
<td>46</td>
<td>52</td>
<td>49</td>
<td>47</td>
</tr>
<tr>
<td>Engaged SNs</td>
<td>34</td>
<td>20</td>
<td>24</td>
<td>33</td>
<td>18</td>
</tr>
<tr>
<td>Result</td>
<td>68.0%</td>
<td>43.4%</td>
<td>46.1%</td>
<td>67.3%</td>
<td>38.3%</td>
</tr>
</tbody>
</table>

From Table above, the average percentage of the number of engaged SNs is 53.8%.