Real-time monitoring and validation of waste transportation using intelligent agents and pattern recognition

Russell, Sean E.

2015

University College Dublin. School of Computer Science and Informatics

http://dissertations.umi.com/ucd:10055

http://hdl.handle.net/10197/6848
Real-time monitoring and validation of waste transportation using intelligent agents and pattern recognition

Seán Russell
06373373
The thesis is submitted to University College Dublin for the degree of PhD in the College of Science

May 2015

UCD School of Computer Science and Informatics

CLARITY: Centre for Sensor Web Technologies

Head of School: Prof. Pádraig Cunningham

Research Supervisors: Prof. Gregory M.P. O’Hare
and Dr. Rem Collier
## Contents

Abstract i  

Statement of Original Authorship i  

Acknowledgements ii  

1 Introduction 1  

1.1 Motivation ................................................. 2  

1.2 The Approach Adopted in this Thesis .......................... 2  

1.3 Objectives ................................................. 3  

1.4 Contributions of this Thesis .................................. 4  

1.5 Outline of Thesis .......................................... 4  

I Background Research 6  

2 Intelligent Transportation Systems 7  

2.1 Intelligent Transportation Systems ................................. 7  

2.2 Freight Logistics and Tracking ................................ 7  

2.3 Content Tracking ........................................... 8  

2.4 Vehicle Augmentation ........................................ 9  

2.5 Waste Management ............................................ 9  

2.5.1 Examples of Waste Management Products ..................... 10  

2.5.2 The Waste Transfer Form .................................. 10  

2.6 Discussion .................................................. 12  

3 Agent-Oriented Programming 14  

3.1 Introduction ................................................ 14  

3.2 Defining Agency ............................................. 15  

3.2.1 Adopting a Definition .................................... 16  

3.3 Agent-Oriented Programming .................................. 17  

3.4 Environment Interface ....................................... 18  

3.4.1 Environment Interface Standard ............................. 19  

3.4.2 CArtAgO .................................................. 20
II Requirements and Design

6 Waste Augmentation and Integrated Shipment Tracking

6.1 The Waste Management Problem
6.1.1 Exploitation
6.1.2 Solution
6.2 Design
6.3 Database
6.3.1 Raw Data
6.3.2 Generated Data
6.3.3 Organisational Data
6.4 In-situ Application
6.4.1 Hardware
6.4.2 Software
6.4.3 Determining Location
6.5 Central Application
6.5.1 Data Receiver
6.5.2 Server Agents
6.5.3 Live Stream
6.5.4 Classifiers
6.5.5 Database Access
6.6 Visualisation Application
6.6.1 WAIST Visualisation Core
6.6.2 Location Bundle
6.6.3 Acceleration Bundle
6.6.4 Light Bundle
6.6.5 Database Bundle
6.6.6 Live Connector Bundle
6.6.7 Event Bundle
6.7 Discussion

III Implementation and Evaluation

7 Facilitating Intelligent Behaviours using Agents
7.1 Intelligent Agents
7.1.1 Agent Programming Language
7.2 Environment Abstraction
7.3 Integration with SIXTH
7.3.1 AdaptorArtifact
9.2.5 In-situ Agents ........................................ 112
9.2.6 Determining Location ................................ 113

9.3 Central Application ...................................... 114
9.3.1 Data Receiver ........................................ 115
9.3.2 Window Builder ...................................... 115
9.3.3 Classifiers ........................................... 116
9.3.4 Server Agents ........................................ 117
9.3.5 Database ............................................. 119
9.3.6 Live Stream ........................................... 119

9.4 Visualisation ............................................. 120
9.4.1 Visualisation Core .................................... 120
9.4.2 Location Bundle ..................................... 122
9.4.3 Acceleration .......................................... 123
9.4.4 Light .................................................. 124
9.4.5 Database ............................................. 125
9.4.6 Live Data ............................................. 125
9.4.7 Events ................................................ 126

9.5 Discussion ................................................ 127

10 Evaluation .................................................. 128
10.1 Introduction ............................................. 128
10.2 Evaluation ............................................... 128
10.2.1 Timeliness .......................................... 129
10.2.2 Stop Identification ................................... 129
10.2.3 Illegal Disposal Identification ......................... 130
10.2.4 Energy Management ................................ 130
10.3 Results .................................................. 130
10.3.1 Timeliness .......................................... 131
10.3.2 Stop Identification ................................... 133
10.3.3 Illegal Disposal Identification ......................... 134
10.3.4 Energy Management ................................ 135
10.4 Discussion ................................................ 136

IV Closing ..................................................... 138

11 Critique and Further Work ............................... 139
11.1 Critique .................................................. 139
11.2 Further Work ........................................... 140
11.2.1 Further Study of Object State Classification ............... 141
11.2.2 Investigation of Generalisation of Application ............... 141
11.2.3 Improvements in the Functionality of Agents and the Events they Generate ........................................... 141

11.3 Summary ................................................................. 142

12 Conclusions ............................................................... 143

Glossary ........................................................................... 147

Bibliography .................................................................... 149

A AOSE Worksheet: Programming for wireless sensor networks using agents 167

A.1 Description .................................................................... 167
A.2 Problem 1 ...................................................................... 167
A.3 Problem 2 ...................................................................... 168
A.4 Working with artifacts .................................................. 168
  A.4.1 Getting access to an artifact ....................................... 168
  A.4.2 Using artifact operations .......................................... 169
A.5 GPSStreamArtifact ....................................................... 170
  A.5.1 Signals ..................................................................... 170
A.6 DeploymentArtifact ....................................................... 171
  A.6.1 Operations ................................................................ 171
  A.6.2 Signals ..................................................................... 172
A.7 RetaskerArtifact ............................................................. 172
  A.7.1 Operations ................................................................ 172
  A.7.2 Signals ..................................................................... 173

B AOSE Worksheet: Programming for wireless sensor networks using Java 175

B.1 Description .................................................................... 175
B.2 Problem 1 ...................................................................... 175
B.3 Problem 2 ...................................................................... 176
B.4 SIXTH, WAIST and OSGi functionality ......................... 176
  B.4.1 Getting data ............................................................ 176
  B.4.2 Converting Data ...................................................... 177
  B.4.3 Getting Deployment Information ............................... 178
B.5 Sending WAIST System Events ....................................... 178

C AOSE Class Survey .......................................................... 181

C.1 Questions ....................................................................... 181
  C.1.1 Personal Information ............................................... 181
  C.1.2 Agents in General ................................................... 181
  C.1.3 ASTRA ................................................................. 182
  C.1.4 WAIST ................................................................. 183
C.2 Results ........................................................................... 183
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Schematic of container integrity monitoring system (Taken from [24])</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Consignee Selection on waste transfer form [48]</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Detail of waste materials [48]</td>
<td>11</td>
</tr>
<tr>
<td>2.4</td>
<td>Confirmation component of waste transfer form [48]</td>
<td>12</td>
</tr>
<tr>
<td>3.1</td>
<td>Architecture of EIS (taken from [11])</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Architecture of CArtAgO (taken from [161])</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Sample AgentSpeak code in Jason (from [19])</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>Sample 2APL agent (Excerpts from [44])</td>
<td>25</td>
</tr>
<tr>
<td>3.5</td>
<td>Sample GOAL agent (Excerpt from [85])</td>
<td>27</td>
</tr>
<tr>
<td>3.6</td>
<td>Sample Jadex agent definition (Excerpt from [153])</td>
<td>28</td>
</tr>
<tr>
<td>3.7</td>
<td>Sample Jadex agent plan definition (Excerpt from [153])</td>
<td>29</td>
</tr>
<tr>
<td>3.8</td>
<td>Sample ASTRA agent</td>
<td>31</td>
</tr>
<tr>
<td>3.9</td>
<td>Excerpts of sample simpaA agent (taken from [164])</td>
<td>33</td>
</tr>
<tr>
<td>4.1</td>
<td>Conversion to combined acceleration</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Temperature Decision Tree</td>
<td>40</td>
</tr>
<tr>
<td>5.1</td>
<td>Representation of wireless sensor networks</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>GSN Model (Taken from [173])</td>
<td>47</td>
</tr>
<tr>
<td>5.3</td>
<td>Sample TinyDB Query (taken from[120])</td>
<td>48</td>
</tr>
<tr>
<td>5.4</td>
<td>The RUNES software architecture. (Taken from [41])</td>
<td>49</td>
</tr>
<tr>
<td>5.5</td>
<td>Registering for filtered data in SIXTH</td>
<td>53</td>
</tr>
<tr>
<td>6.1</td>
<td>Overall WAIST Architecture</td>
<td>58</td>
</tr>
<tr>
<td>6.2</td>
<td>Representation of the tables within the WAIST database.</td>
<td>61</td>
</tr>
<tr>
<td>6.3</td>
<td>Linux based SheevaPlug computer</td>
<td>62</td>
</tr>
<tr>
<td>6.4</td>
<td>SunSPOT WNS devices</td>
<td>62</td>
</tr>
<tr>
<td>6.5</td>
<td>Structure and data flow within in-situ application</td>
<td>63</td>
</tr>
<tr>
<td>6.6</td>
<td>Structure and data flow within central application</td>
<td>66</td>
</tr>
<tr>
<td>6.7</td>
<td>Main window of the visualisation application</td>
<td>68</td>
</tr>
<tr>
<td>6.8</td>
<td>Structure and data flow within visualisation application</td>
<td>69</td>
</tr>
<tr>
<td>6.9</td>
<td>Empty Application</td>
<td>69</td>
</tr>
<tr>
<td>6.10</td>
<td>Map view</td>
<td>70</td>
</tr>
</tbody>
</table>
# List of Tables

3.1 OOP versus AOP (taken from [178]) ........................................ 17  
3.2 Environment Interface support ............................................. 34  
4.1 Example Data Set ............................................................. 41  
5.1 Comparison of Wireless Sensor Network Middlewares ................. 50  
7.1 Number of tasks completed ................................................ 92  
7.2 Question responses .......................................................... 93  
8.1 Converted Frequencies ...................................................... 97  
8.2 Summary of Wavelet Features ............................................. 101  
8.3 Summary of Frequency Domain Features ................................ 101  
8.4 Summary of Time Domain Features ..................................... 103  
8.5 Individual axis results ...................................................... 104  
8.6 Sensitivity and Specificity for best performing feature per domain 105  
8.7 Combined axis results ...................................................... 105  
8.8 Accuracies of Converted Frequencies .................................. 106  
8.9 Confusion matrix for best performing feature set ....................... 107  
10.1 Timeliness results ......................................................... 132  
10.2 Events breakdown .......................................................... 134
Abstract

Within Ireland and other Organisation for Economic Co-operation and Development countries there has been a growing problem of unauthorised waste activity. A report on this activity highlighted a number of problems. Of these, unauthorised collection and fly-tipping of waste is of particular concern due to the potential to cause pollution and health problems.

This thesis presents the Waste Augmentation and Integrated Shipment Tracking (WAIST) system. WAIST utilises technologies from the area of pattern recognition, agent-oriented programming and wireless sensor networks to enable the monitoring and validation of waste transportation in near real-time.

As components of the WAIST system, this thesis also introduces and evaluates two technologies. The first is the classification of object state based on accelerometer data, and the second is the use of agent-oriented programming languages as a high level abstraction for reducing “programmer effort” when implementing intelligent behaviours within WAIST.

Both evaluations show positive results. In the classification component, an accuracy of 95.8% has been achieved in an eight class problem. In the agent component, students completed more tasks when using agents than when using Java. Additionally, subjective feedback highlighted a perception that problems were easier to solve using agents.

Finally the WAIST system itself was evaluated over a number of simulated waste shipments based on a number of criteria. The results are very positive for the timeliness of the system, the ability to track stopping locations of the shipment, the accuracy when identifying illegal dumping and the efficient management of energy resources.
Statement of Original Authorship

I hereby certify that the submitted work is my own work, was completed while registered as a candidate for the degree stated on the title page and I have not obtained a degree elsewhere on the basis of the research presented in this submitted work.

The research was conducted with the several collaborators. Their collaboration was as follows:

I collaborated in the design and development of components of the SIXTH middleware for the sensor web which Dominic Carr is the primary contributor.

Some contributions were made to the ASTRA programming language which is primarily the work of Rem Collier.
Acknowledgements

This thesis was possible in no small part because of the support and encouragement of my family. Their support through the long years of my education gave me the belief that this goal was attainable. I am forever grateful for this and the environment in which I was raised.

I am also thankful of the foresight and advice of my supervisors Dr. Rem Collier and Professor Gregory O’Hare. In particular I would like to thank Rem, as he first identified and nurtured my academic interests. He became a friend and the first person I would come to with my ideas even before I undertook this work.

I would also like to thank the various bodies that have supported me financially through this journey. Principally the Environmental Protection Agency of Ireland which was the initial source of funding for my research. The money was never very regular in coming but usually arrived eventually. This may sound like a slight, but without regular requirement to secure needed funds for living I would not have gained the required experience for my current career. Additionally I would like to thank the CLARITY: Centre for Sensor Web Technologies.

This would could not have been completed without the help of John Dunnion and Henry McLoughlin who were instrumental in giving me the opportunity to greatly broaden my horizons by spending a semester lecturing in Beijing-Dublin International college. A mutually beneficial arrangement which also funded my final year of research. This brief period solidified my beliefs about what I wanted to do with my life and offered my the necessary experience to make it a reality.

Additionally, I would like to thank Waseem Akhtar in Griffith College Dublin for giving me my first taste of lecturing. My time working there not only supported me financially when it was required but gave me the first real idea as to what I wanted to do after my PhD.

Finally, I would like to thank my colleagues within the research group, in particular Dominic Carr. All of my work was based on the successful use of his work and he was always available to help me understand how to achieve something or to make a change to help me achieve it. His nature to make himself available to help, the comradeship of another person facing similar challenges and his keen sense of humour make him an individual I will always be happy to work with in the future. Finally I would like to thank David Lillis and Abey Campbell who helped support the lowly postgrads under their direction.
List of my Publications


Chapter 1

Introduction

“to see if anyone got in or out, and where it went. ... we just construct a virtual of the entire city for Sunday evening. Every smartdust mesh, every spectrum, every road macromesh; sling it all together in an AI and watch our own history play out in hi-rez detail.” - Peter F. Hamilton [81]

This quote by the science fiction writer Peter F. Hamilton is from the 2013 novel *The Great North Road*, which is set in the year 2143. By this time, current technologies have evolved to the point where imperceptible sensing devices are simply sprayed onto buildings and roads. These devices automatically form networks capable of monitoring and, in some cases, controlling life within a city. The evolution of these technologies is quite predictable with respect to the current trends of diminishing size and increasing complexity. However the author has identified one possible problem arising from this development: the volume of data produced would overwhelm a human's ability to process it. In this world, programs sophisticated enough to be referred to as artificial intelligence are required to integrate this data and produce something acceptable for human perception.

A great deal of research still remains between the future world envisaged and the current level of technology today. Developments in the area of microelectromechanical systems (MEMS) [182] and into the future, nanoelectromechanical systems (NEMS) [84] may one day provide the means to achieve devices the size of pin-heads. Research in the area of wireless sensor networks may enable the most energy-efficient manner for these devices to communicate and operate. Thereby allowing the devices to operate for long periods on battery reserves or energy harvested from the environment.

Wireless sensor networks, when combined with autonomic computing aims to ensure that recovery and self optimisation occur without human intervention when changes in the environment impede on the ability of these systems to achieve their goals.

Pattern recognition [180] techniques and intelligent agents [178] are providing the first steps towards the autonomous management of the deluge of data potentially available from these evolving devices. However it is likely that dramatic increases in difficulty will be faced to
enable the integration of these heterogeneous sources of data.

This thesis embraces the potential of utilising two disparate branches of artificial intelligence [189]; pattern recognition and agent-oriented programming, to produce an intelligent transportation system for the problem of monitoring and validating the transportation of waste. Pattern recognition is used to determine the state of objects, and agent-oriented programming facilitates intelligent reasoning. Collectively both technologies enable inferences be made about not only what occurred but why.

1.1 Motivation

According to the Environmental Protection Agency (EPA) of Ireland there has been a growing problem of unauthorised waste activity [56]. This unauthorised activity takes a number of forms, however this work within this thesis only concerns illegal dumping that may be carried out by a licensed haulier. A possible solution put forward by the EPA is the “development and implementation of a national waste tracking system to allow for the pre-notification and tracking of all waste movements within and outside the state”.

The requirements of such a system should have:

- The ability to monitor and record the location of waste shipments in transit.
- The ability to highlight possible instances of illegal dumping during a shipment.
- The ability to present all captured data in an easily understandable manner.

The primary motivation behind this work is the development of a system for the monitoring of certain types of waste while in transit. Principally, this refers to units of waste that are transported by a licensed carrier and destined for an authorised processing facility. This waste can be commercial or domestic and may be deemed hazardous. The system known as Waste Augmentation and Integrated Shipment Tracking (WAIST), is designed to use data gathered from various sensors to reach this end.

Principally, GPS sensors are used to determine the location of a shipment of waste and other sensors are used to ascertain the condition of the waste within the shipment. The determination of the condition of the waste based on reading from acceleration sensors attached to the waste forms one of the major components of this thesis. Additionally investigated is the use of agent-oriented programming languages as a high-level abstraction for easily implementing behaviours.

1.2 The Approach Adopted in this Thesis

This Thesis demonstrates a number of technologies that can be used in the development of intelligent transportation systems for waste management which are built using wireless sensor
networks. Having reviewed candidate technologies, a wireless sensor network middleware, environment interface and agent-oriented programming language were chosen. The foundation of the system is the SIXTH middleware for the sensor web [28, 31, 147], which will facilitate the gathering and routing of data. The CArtAgO platform [165] will be utilised to allow the introduction of intelligent agents into the system. This is achieved through the provision of standard interfaces with SIXTH and the system itself. Finally intelligent agents are realised through the use of the ASTRA agent-oriented programming language [39, 114].

These technologies and their development will be described within this thesis. The main target of these technologies was for use in an intelligent transportation application for waste management, and individually they have been evaluated in this light. The extent to which they can be generalised for use in other intelligent transportation systems, in the case of the pattern recognition, sensor network middleware or intelligent agents, is outside the scope of this thesis.

All of the technologies will be evaluated in isolation and then used as components in an intelligent transportation system for waste management based on wireless sensor networks. This system will illustrate the advantages of using an agent-oriented programming approach in achieving the goals of intelligent and accurate decision making, as well as the re-usability of components across multiple applications. The combination of the concept of agent-oriented programming and wireless sensor networks creates a natural synergy within this framework. The agents view the world through software sensors (not to be confused with physical sensors which may be the ultimate source), and act out their beliefs, desires and intentions through actions. The source of the data sensed by these agents is typically, but not always, from the physical world.

1.3 Objectives

The objectives of this thesis are based on a single primary objective. This objective is the creation of a near real-time [183] application for the monitoring and validation of waste transportation. It is believed that the achievement of this overall objective would be facilitate by achieving the following secondary objectives:

Ob. 1 An evaluation of the applicability of pattern recognition techniques used in human activity classification studies to a previously unstudied problem of determining object state within a transportation context.

Ob. 2 To demonstrate the use of agent-oriented programming languages as a high level abstraction in order to minimise the effort required when adding and changing behaviours within an intelligent transportation system.

Ob. 3 To develop and evaluate a state of the art waste tracking application using pattern recognition and agent-oriented programming, with the capability to:
• Track the location of waste shipments in “Near real-time”.
• Identify all locations where each shipment was stationary.
• Based on pattern recognition techniques, highlight when tampering may have occurred.
• Using intelligent agents, manage the energy resources of the sensors within the deployment.

1.4 Contributions of this Thesis

The main contributions of this thesis are:

1. Detailed reviews into the areas of agent-oriented programming and pattern recognition and introductions to the areas of intelligent transportation systems and wireless sensor networks. These focused on the application of technologies to the area of waste management and represent the state of the art in this context.

2. The demonstration of pattern recognition techniques applied to the domain of object state classification during transportation. An implementation of this near real-time classification system capable of accuracies of up to 95.8% in an eight class problem.

3. The use of intelligent agents as a means to ease the development of sensor network applications. Demonstrated by a small scale evaluation of the efficacy of the system.

4. The development of an end to end wireless sensor network application named WAIST, capable of near real-time monitoring and validation of waste transportation.

5. The development of a number of domain specific visualisation methods. These are designed to simplify the consumption of the large amounts of data generated within the application.

1.5 Outline of Thesis

The structure of the thesis is as follows:

Chapter 2: discusses areas of intelligent transportation systems relevant to this work. Particular attention is focused on the area of waste management and potential solutions to the problem of verifying correct disposal.

Chapter 3: introduces the area of agent-oriented programming. A number of agent-oriented programming languages are reviewed and the manner of integration with their environment is discussed.
Chapter 4: describes the area of pattern recognition with a focus on classification of human activity. This information is supplied with a view to being applied later in the problem of classifying object state within a transportation application.

Chapter 5: briefly introduces the area of wireless sensor networks and reviews select middleware solutions that can enable their use. The selected technology is then described in detail.

Chapter 6: describes the problem of waste management more specifically. Cases where the currently used systems can be exploited are discussed and the requirements of a potential solution are detailed. The design of the Waste Augmentation and Integrated Shipment Tracking (WAIST) system is introduced as a potential solution to the problem detailed.

Chapter 7: describes the integration of an environment interface with WAIST, thereby facilitating the use of intelligent agents within the application. An evaluation is performed based on the ability of the system to reduce “programmer effort” when implementing intelligent behaviours within the application.

Chapter 8: details the evaluation of pattern recognition techniques from human activity classification when applied to the classification of object state in a transportation context. A large number of feature extraction techniques are evaluated as well as a method of data combination. The best performing feature is then utilised in a study of the reduction in accuracy with decreased sampling rate.

Chapter 9: describes the implementation of the WAIST system initially described in Chapter 6. The system is described in terms of the three applications it is composed of and the interoperation between them is highlighted.

Chapter 10: details the evaluation of the WAIST system. This assessment is carried out with respect to the objectives discussed in Section 1.3.

Chapter 11: presents the limitations of this thesis. Additionally, a number of possible avenues for further work are discussed.

Chapter 12: concludes the thesis by detailing how the objectives discussed were achieved.

Appendices A and B: present the full problem statements and reference materials used in the evaluation of Chapter 7.

Appendix C: contains the unabridged survey used in the evaluation of Chapter 7. The full results are also presented.
Part I

Background Research
Chapter 2

Intelligent Transportation Systems

The work within this thesis encompasses a number of different technologies combining the use of wireless sensor networks, intelligent agents and pattern recognition. Within this thesis these independent areas are only discussed in the context of their use within a small domain. The domain is that of Intelligent Transportation Systems (ITS), although more specifically this work is aimed at intelligent transportation systems that are based on the use of wireless sensor networks.

This chapter briefly introduces intelligent transportation systems and describes a number of active research areas that relate in some way to the work of this thesis. Finally the area of waste management is discussed, as the work of this thesis focuses on the monitoring and validation of waste disposal, several products and technologies designed in this area are discussed.

2.1 Intelligent Transportation Systems

Intelligent transportation systems refers to applications created with the aim of providing innovative services relating to transport and traffic management. Often these systems will aim to make transport networks safer or more efficient for users, others aim to reduce congestion and carbon dioxide levels.

2.2 Freight Logistics and Tracking

The area of freight and logistics is closely related to the work of this thesis in terms of the technologies utilised, commonly employing GPS sensors or wireless sensor networks to achieve similar goals. Research in the area of freight and logistics tracking intelligent transportation systems is often dominated by security concerns, this could be attributed to the Container Security Initiative, which was launched in 2002 by the U.S. Bureau of Customs and Border
Protection, an agency of the Department of Homeland Security in the United States of America. Its purpose was to increase security for container cargo shipped to the United States and resulted in an increase in research in the area [10].

A number of systems propose different candidate technologies as potential solutions to the problem. Within the context of port-based container logistics, radio-frequency identification (RFID) is a technology with the potential to enable tracking the movements of containers through the transport network [177]. This type of system does not provide a fine grain location of a container, but with scanning all containers on entry and exit of multiple ports can display a history of the journey of a container.

Enhancements to this system also utilise GPS to improve the process of determining the location of the container within the port. GPS receivers are not directly installed on containers, but on top of the transport and stacking equipment. The position is measured, translated into yard coordinates, and transmitted to a central system whenever a container is lifted or dropped.

An alternative method of locating containers within ports based on the use of wireless sensor networks (which will be discussed in detail in Chapter 5) has been investigated by Abbate et al. [1]. The principle idea in this work is that a wireless sensor network exists within each container and the proximity of other containers is calculated based on the measurement of the received signal strength (RSS) of the wireless communication channel. The positioning of the container is then inferred based on the calculated distances between different sensors within the wireless sensor networks.

### 2.3 Content Tracking

Finer grain tracking of a container can be achieved than that of merely determining location. Tracking the contents of a container has also been the subject of research. However this tracking tends not the focus on the location, but on the condition of the contents [24]. The area of content tracking relates quite closely to the work carried out in Chapter 8, which forms a major component of the overall system.

In an effort to detect damages during transit, acceleration sensors are used to determine the state of cargo within a container. Figure 2.1 shows the schematic of the system where sensors are attached to individual packages within a container and transmit data to a base station located within the same container. Using pattern recognition techniques, this system was capable of distinguishing between the different mechanical conditions of when a package was sliding, colliding with another package or container, tilting and wobbling, with accuracies of up to 100% [176].
2.4 Vehicle Augmentation

Much research focuses on the enhancement of on-board systems within vehicles such as information gathering and communication. The aim of this research is typically to improve overall road safety. Systems that can warn a driver of an impending collision or upcoming red light [83] are an example of such research. The use of wireless sensor network devices in vehicles appears regularly within intelligent transportation system research, often referred to as vehicular sensor networks (VSNs) when working in cooperation with neighbouring vehicles and infrastructure [139].

2.5 Waste Management

The area of waste management has received considerable attention in academia [34, 73, 185, 187, 188], however the problem of tracking waste in a verifiable manner has until now received little scrutiny. In contrast, the problem of route optimisation in a waste-handling scenario has been the focus of a number of papers [33, 171]. A recurring theme throughout a number of academic papers concerns the potential of a number of technologies for waste tracking and the prevention of illegal dumping, for example using spatial imagery [174]. However, descriptions of designs and implementations of even prototypical examples are almost non existent.

Interestingly, the use of RFID is proposed on a number of occasions as a technology that could be utilised in this area. However, a note of caution has been raised in the case of RFID [104]. They note that RFID tags are essentially electronic components, and are such, are subject to regulations for the safe disposal of such material (Directive 2002/96/EC [62] and Directive 2002/95/EC [61]). Thus their use for tracking waste material must be considered
The issue of waste management has also attracted the attention of industry. The services offered generally take the form of either a paper or digital method for monitoring the status of waste materials. The primary focus of these service providers has been on the provision of services that remove liability from the waste producer or certify compliance with relevant regional disposal legislations.

2.5.1 Examples of Waste Management Products

RILTA Environmental offers a waste tracking service based upon the use of barcodes [166]. A barcode is generated initially and linked with descriptive information and then scanned every time the container is loaded/unloaded at a destination. Enva use a centralised database system to manage all hazardous and non-hazardous waste they process [53]. Waste Trace is aimed at manufacturers producing toxic waste, and enables tracking of the waste from generation to transportation and disposal [37]. In addition it has a reporting facility that ensures compliance with U.S. Federal, State, and EPA regulations. LinkoHW Hauled Waste tracking software is a vertical application that allows companies manage the tracking and disposal of their waste but is not concerned with what happens after the waste has left their premises [116]. EnviroServ have developed a paper-based system that allows the tracking of hazardous waste, it enables legal compliance to the relevant regulations in New Zealand [57]. Though these are just a sample of the available products, it can be seen that while they do indeed fulfil niche needs for business, none meets the needs of a national organisation or municipal authority that is responsible for the monitoring and auditing of waste disposal in their jurisdictions. Within Ireland shipments of hazardous waste are required to be tracked through the use of an on-line waste transfer form (WTF), which replaced the paper based C1 form. A waste transfer form must be completed and submitted prior to a collection being made and upon delivery the receiving facility logs in to confirm acceptance of the load [91].

2.5.2 The Waste Transfer Form

A waste transfer form can only be created by parties authorised and verified by the National Transfrontier Shipment (TFS) office, and during creation the form is linked to a consignee (facility receiving the waste). Figure 2.2 shows the consignee specification, it can be seen that the destination of a hazardous waste shipment is limited to facilities authorised by the national TFS office.

Comprehensive detail is required in order to complete the waste transfer form. Figure 2.3 shows the details required to describe waste that is to be transferred, it includes weight, volume and a description of the physical characteristics of the waste. The waste is also characterised using categories described in the European Waste Catalogue and Hazardous Waste List [54],
Figure 2.2: Consignee Selection on waste transfer form [48]

this process enables verification that waste is being transferred to a facility that is licensed to receive the described materials.

Figure 2.3: Detail of waste materials [48]

The key verification of transportation lies in the confirmation of the waste transfer form by the receiving facility. Figure 2.4 shows the component of the form filled out by the consignee.
The information relating to the weight and volume of the materials is replicated as well as the European waste catalogue codes with space for any disparities to be explained. Information is also retained about the vehicle in which the waste was received.

![Waste Transfer Form](image)

Figure 2.4: Confirmation component of waste transfer form [48]

### 2.6 Discussion

This chapter introduced the concept of intelligent transportation systems and continued to highlight a number of research areas that are related to the work described in this thesis. The area of freight logistics and tracking is related primarily with respect to the technologies utilised. The area described as content tracking closely resembles work carried out which will be described in this thesis. The employment of sensing devices within vehicles is a common theme in the area, however the problems generally addressed such as vehicle augmentation or inter-vehicle communications are not of concern to this work.

Finally the area of waste management was discussed. As this area is where the work of this thesis is situated, particular attention was focused on the validation of waste materials transportation. A number of commercial systems for waste monitoring were described. However,
none of these are aimed at performing validation of waste for a municipal authority.

The functioning of an on-line system for waste validation was described in detail. This waste transfer form system utilises information from multiple parties to verify that the was is disposed of correctly. Chapter 6 will highlight areas where this system is vulnerable to exploitation, and such further motivate the work in this thesis.

Following from this chapter, the next chapters will continue reviewing areas of importance to this work, describing the areas of agent-oriented programming, pattern recognition and wireless sensor networks.
Chapter 3

Agent-Oriented Programming

3.1 Introduction

Agent-oriented programming (AOP) is a software development paradigm aimed at creating entities known as “intelligent agents” [95] or “rational agents” [198]. Agent-oriented programming languages are typically utilised in the development of large-scale distributed Multi Agent Systems (MASs) and are intended to provide higher level abstractions than traditional distributed programming [179].

One of the aims of this thesis is to explore how the higher level abstractions provided by agent-oriented programming languages can be utilised to ease the development of applications for intelligent transportation systems. To facilitate this aim this Chapter begins this review by examining agent-oriented programming in general.

It is first necessary to arrive at an appropriate definition of what exactly is meant by the term “agent”. Consensus on what exactly constitutes agency has never been reached. Several definitions have been proposed and these definitions vary in the depth to which they go. Despite this, there is sufficient agreement on the core attributes of an agent to allow a relatively uncontroversial definition to be arrived at. This is described in Section 3.2.

The agent-oriented programming paradigm is then introduced in Section 3.3. Section 3.4 will introduce the available technologies that can be used to integrate the agents with the environments they will operate in. This is followed by a short survey of some modern agent-oriented programming languages compatible with the above environment standards in Section 3.5. The work in this thesis aims to provide a functionality that can be used in as broad a set of agent technologies as possible. As such, it is important to identify the capabilities of languages that are already in use, so as to identify languages that can be easily integrated with this work. Finally Section 3.6 summarises the above and discusses the choices made on the basis of this information.
3.2 Defining Agency

In the context of software engineering, the term “agent” is associated with a myriad of often competing definitions. This ranges from simple programs that perform tasks on behalf of users to sophisticated entities that leverage principles of Artificial Intelligence to perform complex planning, reasoning and actions. Within the agent-oriented programming community researchers regularly make use of terms such as “intelligent agents” [95] or “rational agents” [198] when describing their work. The indication in this context is that the “agent” described is distinct from simpler programs.

This distinction is usually centred around the idea of intelligence, which is generally understood that the agent is capable of autonomous action. Without the requirement for intelligence, Jennings and Wooldridge note that processes such as the UNIX xbiff utility, which perceives the environment in which it is situated (monitoring a user’s incoming email) and takes actions when that environment changes (alerting a user to the presence of new mail), could be described as an agent [94]. Similarly Munindar comments that entities are often misrepresented as agents, stating that many “agents” referred to are “typically no more than glorified search engines or user interfaces” [179].

According to Fisher while the central aspect of an agent is autonomy, rational agents are agents that have reasonable and explainable courses of action that they undertake. It is argued further that the key concept that rational agents have brought to the forefront of software design is that, as well as describing what an agent does, it is vital to describe why it does it [63]. In order to exhibit this behaviour a number of aspects of an agent must be represented, namely informational (beliefs and knowledge), motivational (goals or intentions) and deliberative aspects (why it makes the choices it does).

The agents community has long acknowledged that agreement has not been reached on what exactly an “agent” actually is [94]. The weak notion of agency was proposed by Wooldridge and Jennings [199], they believe that this definition is a relatively uncontroversial one. This weak notion of agency includes autonomy, social ability and a combination of reactive and proactive behaviour. The meaning of each term is discussed in the following Sections. These four attributes are sometimes considered the only key features [196] and other times the definition is expanded to include other features such as mobility, benevolence and situatedness.

Expanding the weak notion of agency, Wooldridge and Jennings also propose a stronger notion of agency. This notion states that an agent “is either conceptualised or implemented using concepts that are more usually applied to humans” [199]. This conforms with the definition of an agent given by Shoham, which describes an agent as a computing entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices, and commitments [178].

**Autonomy:** The concept of autonomy is a key component of almost all discussions of intelligent agents [32, 38, 100, 118, 178, 179, 196, 199]. Agents make their own decisions
about what actions they take and can be thought of as requesting actions of other agents rather than invoking methods of other agents.

**Social Ability:** The narrow definition of an agent given by Singh identifies interoperability, or the ability of agents to communicate with one another, as another key concept [179]. This identification is repeated in Wooldridge and Jennings weak notion of agency as the social ability [199].

**Reactivity and Proactivity:** The weak notion of agency contains the requirements for both reactivity and proactivity [199]. In this context proactivity refers to the ability of the agent to take initiative towards achieving any goals it may have and reactivity refers to the ability of an agent to respond to changes in its environment in a timely manner.

**Mobility:** Mobile agents are those that are capable of moving from one environment to another, typically this mobility manifests as moving to another host on a network [199]. Mobility is considered important for many researchers (e.g. [52, 134]), it is however not core to the definition of an agent. The categorisation of agents as static or mobile agents [142] emphasises that mobility is not a requirement of agency.

**Environment:** Most agent definitions make reference to the environment or consider situatedness in an environment as being a central aspect to agency. Typically, this is in the context of the agent having the capability of perceiving its environment [40, 75, 95, 121, 179].

### 3.2.1 Adopting a Definition

In the context of this thesis, the stronger notion of agency as set out by Wooldridge and Jennings [199] has been adopted, this considers agents to be autonomous, social, reactive and proactive and implemented in terms of mentalistic components.

As discussed above, fundamental to any agent definition is the feature autonomy. As such it is important that elements of this thesis should not diminish the autonomy of an agent that makes use of it. As the work in this thesis focusses specifically on the situation of the agents within particular environments, the reactive ability of agents is also an acceptable key requirement. It is assumed that all agents are capable of interacting with the environment through the use of an environment interface which will be described in detail in Section 3.4. The social nature or interoperability of the agents is important in so far as agents within any Multi Agent System (MAS) must be able to communicate. Finally it is expected that an agent utilised within this thesis would display proactivity in the manner to which it attempts to achieve it’s goals.

For all other potential aspects of an agent the creators of MASs and MAS toolkits can decide whether they are relevant to their model of agency. The classification of these aspects as fundamental to the definition of an agent or not is not required in the context of this work. For
instance the requirement that an agent be capable of migrating between different environments is not a requirement of this work. The ability of an agent to migrate to another location can be considered an asset, it is not however at the core of the work presented here.

3.3 Agent-Oriented Programming

The phrase “agent-oriented programming” was coined by Shoham when proposing a computational framework in which the state of agents consist of components such as beliefs, decisions, capabilities and obligations [178]. This initial proposal viewed agent-oriented programming as a specialisation of Object Oriented Programming (OOP). This specialisation is characterised by the constraint of concepts such as state or message types. Table 3.1 (directly taken from [178]) shows the relationship between the two paradigms.

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

Other researchers have focused on the relationship between agents and objects as well (e.g. [23, 25, 102, 146]). Odell et al. define an agent, from a software development viewpoint, is “an object that can say ‘go’ (dynamic autonomy) and ‘no’ (deterministic autonomy)”. In line with the definition of an agent adopted in Section 3.2.1, autonomy is key to the definition. The definition separates “dynamic” and “deterministic” autonomy, the former refers to an agents ability to exhibit proactive behaviours while the latter is the ability to refuse or modify an external request. In contrast objects typically cannot act without external invocation, and must perform the associated action whenever such an invocation occurs.

Within Shoham’s proposal is the notion of agents interacting within a societal construct. The principles of speech act theory [8] are proposed as the basis for how agents communicate, i.e. agents inform, request, offer, accept etc.

Logic programming has oft been utilised in an attempt to reason about mental attitudes by the Artificial Intelligence community. Wooldridge and Jennings see agent-oriented programming as developing practical programming languages that embody the various principles proposed by theorists [199]. Furthermore Shoham remarks that it would be tempting and possible in principle “to view agent-oriented programming as a form of logic programming”, however this is not his intent [178].
Probably the most influential mentalistic model for intelligent agents is the *Belief, Desire, Intention* (BDI) model [21]. One of the earliest models, it represents an agent in terms of three mental attitudes. Beliefs represent information, desires represent motivation and intentions model decisions [159].

The abstract interpreter proposed by Rao and Georgeff [160] inspires most practical implementations of BDI systems. The interpreter operates in a cyclical fashion, each iteration requires a number of stages;

- generating options (actions that the agent can undertake)
- creating intentions to carry out some of the options
- execution of the adopted intentions
- generating new events
- dropping impossible and successfully completed intentions

This iterative approach to an agent interpreter is one that has been followed in many implementations since. Both purely imperative and purely declarative approaches to tackling the issue of programming agents have been tried. However many practical agent-oriented programming languages adopt a hybrid approach. Section 3.5 discusses a number of agent-oriented programming languages and frameworks.

### 3.4 Environment Interface

Agents are situated in environments in which they perceive and act. An issue that repeatedly has to be dealt with, from a software development point of view, is how to connect agents to environments. The design and implementation of the interaction between the agents and the environment often requires substantial effort, even though most environments and agent systems are implemented in Java. Partly this is due to the fact that each environment is different or that different agent-oriented programming languages and toolkits provide different levels of support for agent environment interaction. Commonly this can hinder the exploitation of the full potential of the environment by multi-agent systems [195].

In order to mitigate the impact of the problem, the work in this thesis will utilise an environment layer to simplify the integration of agents and environment. The two most well-known environment layer technologies in the area of agent programming are the Environment Interface Standard (EIS) [12] and the Common Artifacts for Agents Open framework (CArtAgO) [161]. The following sections will discuss the two technologies.
EIS follows a traditional view of environments as it implements an interface that models the environment layer as a set of entities. These entities have an associated state which is modelled through a set of beliefs or percepts. Additionally there are potentially a number of actions that can be performed by the entities. While entities and agents may seem similar, there is a distinction between the them.

The connection of agent and entities is the mechanism through which the situatedness of the agents is established. In the context of agent-environment interaction, an entity may be controlled by an agent. These entities facilitate the connection between the agents running on the agent platform and the environment. Once an agent is connected to an entity it is provided with effecting capabilities and sensory capabilities. Sensory capabilities refers to the reception of the entities percepts and events by the agent and effecting capabilities refers to the ability of the agent to actuate the environment through the actions of the entity.

Figure 3.1 shows the model architecture proposed for the environment standard, the components Environment Management System and Environment Interface for the EIS layer and act as a buffer between the environment model and the agents and the platform they operate on.

From a development perspective, EIS promotes an opaque view of an environment where the implementation of the actual entities is hidden. The set of percepts and events generated by entities as well as the actions that can be performed by the entity are specified only in sup-
porting documentation. Additionally in its current incarnation, EIS environments are loaded from the local file space using a dedicated (and inaccessible) Java ClassLoader. Typically within the agents community this would not be a problem. It does however, make EIS difficult to integrate into other technologies, such as OSGi, that are also designed to manage ClassLoaders.

While it is possible to modify the EIS interface to be OSGi-sensitive, it would require the creation of a new version of EIS that may not be compatible with current EIS-enabled agent toolkits. Instead, the standard EIS approach in such scenarios is either to make the EIS environment into an ad-hoc remote client (e.g. [2, 13, 14]) or to completely embed the system within the EIS environment implementation.

3.4.2 CArtAgO

The operation of CArtAgO is not too dissimilar to the operation of EIS. CArtAgO models the environment as a set of artifacts (approximately equivalent to an EIS entity) that can be manipulated by performing operations. Artifacts can be generally conceived as passive, function-oriented computational entities, explicitly designed to provide some kind of function, and then to be used by agents to support their individual and collective activities [161]. There are several differences between EIS entities and CArtAgO artifacts, primary among them is the fact that artifacts are shared within the environment and may be used by more than one agent, where as entities can only be associated with a single agent at any time.

Figure 3.2 shows the architecture of CArtAgO, the interaction between agents and the artifacts is achieved through what is described as agent bodies. The agent body contains effectors to perform actions upon the working environment, and a dynamic set of sensors to collect stimuli from the working environment. The agent body is meant to be controlled by the agent, thereby interacting with their working environment, executing actions provided for artifact construction, selection and usage, and perceiving observable events generated from such artifacts.

Artifact state is modelled as a set of named observable properties and signals (analogous to events) that store Java objects. Changes in the observable properties result in the generation of custom events that are passed to the agent layer (e.g. property added, property updated, etc.). An environment within CArtAgO more transparent than an EIS environment, it is composed of a number of artifacts, and can be augmented with more by the agent programmer. These artifacts are implemented as Java classes extending from a provided code base, as such the operations, observable properties and signals of the artifact is completely transparent.

From a development perspective CArtAgO does not suffer from the same issue as EIS and can be easily integrated into OSGi as a dedicated bundle. A further benefit of CArtAgO is its built-in support for a distributed runtime. This means that any agent toolkit that supports CArtAgO can interact with artifacts deployed using OSGi even if the agents themselves cannot
be deployed using OSGi.

### 3.5 Agent-Oriented Programming Languages

This section briefly introduces a number of agent programming languages and some frameworks that support the agent-oriented programming development. The following sections contain an in-depth review of some current frameworks and languages. The languages in this review represent the all candidate technologies compatible with the environment interface technologies discussed in Section 3.4. Those selected are languages and frameworks, which were identified as having either language level or additional support devoted to the integration with one or both of those environment interfaces.

#### 3.5.1 Jason

Jason [19] is an interpreter written in Java for an extended version of AgentSpeak(L) [159]. AgentSpeak(L) is a programming language originally aimed at bridging the gap between the theoretical BDI model and the practicality of existing BDI-style programming languages that lacked a sound theoretical underpinning.
AgentSpeak(L) defines events and actions through the use of a restricted first-order language. Although desires and intentions are not explicitly modelled in the language, these mental attitudes can easily be applied to the execution model of the agent. The key mental components of the AgentSpeak(L) language are beliefs and goals. These model the agents view of its environment, in addition to states the agent wishes to bring about. Events or triggering events are used to select those plans to be executed, these events often take the form of belief additions and/or removals. A plan describes a sequence of actions that the agent should carry out in order to satisfy some goal.

There are two types of goals; test goals and achievement goals. Test goals are proactive checks on the belief base of the agent to test if the agent believes some proposition and achievement goals represent the aim to bring about a particular state of affairs that is desired by the agent. Adopting an achievement goal results in the selection of a plan with the means to result in the desired state. An adopted plan represents the intention of BDI parlance. Plans are selected through the use of rules. A rule will consist of a triggering event, a context and a plan body.

In the process of implementation, Jason was extended with practical capabilities not required in a formal definition of the AgentSpeak(L) language. These extensions, such as environment integration, communication and the ability to distribute over a network, are detailed in the extended operational semantics [19].

The operation of the Jason interpreter is iterative. On each iteration it performs the following steps:

- updates the list of events for each agent;
- selects an event and identifies a set of relevant plans;
- relevant plans are narrowed down using the context to those that are applicable;
- one of the applicable plans is chosen;
- an active intention of the agent is selected for execution.

Figure 3.3 shows AgentSpeak(L) for use in Jason. The code is taken from a disaster-recovery scenario where agents cooperate to disarm a bomb in an airport. The early lines (1 - 5) represent the initial beliefs of the agent. Initially the agent believes that it has the skill required to disarm plastic explosives and biological weapons (lines 1 and 2), but not nuclear bombs (line 3). The agent is also aware that “field1” is a safe area where it can put bombs it does not have the skill to disarm.

Following this there are 4 plans (labelled p1 through p4). The triggering event in the first three plans is the same, the addition of a belief that there is a bomb at a given gate in a given terminal of a particular type. The context of the agent determines which of the plans will be selected for execution. Plan p1 is fired when the agent believes that it has the skill to disarm the bomb, the value of the variable BombType is bound in the event and the context attempts to match to a skill belief for that bomb type. For Example if the event was
skill(plasticBomb).
skill(bioBomb).
¬ skill(nuclearBomb).
safeArea(field1).

@p1
+bomb(Terminal, Gate, BombType) : skill(BombType)
<- !go(Terminal, Gate);
    disarm(BombType).

@p2
+bomb(Terminal, Gate, BombType) : ¬ skill(BombType)
<- !moveSafeArea(Terminal, Gate, BombType).

@p3
+bomb(Terminal, Gate, BombType) : not skill(BombType) & not ¬ skill(BombType)
<- .broadcast(tell, alert).

@p4
+!moveSafeArea(T,G,Bomb) : true
<- ?safeArea(Place);
    !discoverFreeCPH(FreeCPH);
    .send(FreeCPH, achieve, carryToSafePlace(T,B,Place,Bomb)).

...
Native environment interaction in Jason is defined through the use of Java classes and methods. For this reason it is relatively easy to combine Jason with the environment layers mentioned in Section 3.4. By default CArtAgO comes bundled with the necessary code to act as a bridge between the MAS and the environment layer. The integration of EIS and Jason is described as straightforward, requiring primarily the translation between types [12].

3.5.2 2APL

2APL [44] is a formally-specified, BDI-based agent-oriented programming language, with a focus on MASs that share access to common external environments. 2APL is an extension of the language 3APL, which was aimed more at the creation of individual agents [45, 86].

At the individual agent level, 2APL agents are implemented in terms of beliefs, goals, actions, plans, events, and three different types of rules. Beliefs and goals of 2APL agents are implemented in a declarative way, while plans and external environments or their interfaces are implemented in an imperative programming style. Agents are assumed to be situated within an environment in which the may sense either actively or passively.

2APL uses three types of rules which can be used to generate plans, execute abstract actions and replace plans when they fail. Goals are not removed until they are achieved, meaning that if a plan fails to achieve the goal for which it was executed, it can be replaced by another plan. This process can continue until the goal has been achieved.

Figure 3.4 shows a number of elements of a 2APL agent taken from [44]. The examples are in the context of a virtual world where trash and gold are placed in particular locations. The agents role is to clean the trash from the world and collect the gold.

A 2APL agent typically begins with the declaration of initial beliefs and belief rules, lines 2 to 5 show a number of initial beliefs and line 6 shows the example of a belief rule which uses the same syntax as Prolog. The declaration of goals is also typically at the beginning, although new beliefs and goals can be adopted by the agent during execution. Line 8 show the declaration of goals for the agent, where goals are separated by commas. The agents goals are that it has 5 gold and the world is clean and that it simply has 10 gold.

Belief updates rules are given in the from of a Hoare triple, which contains a precondition, action and post condition [89]. The example given on line 11 relates to the action of picking up gold, it states that the precondition of picking up gold is that you are not already carrying gold and that the post condition of action is that the agent will be carrying gold.

Complex tasks are performed by combining multiple basic actions within an atomic plan. Plan operators include a sequence operator, conditional choice operator, conditional iteration operator and a non-interleaving operator. Line 14 shows an example of a plan, the brackets around the first two actions creates an atomic plan that ensures that the ChgPos(5,5) is executed immediately after (the semicolon is the sequential operator) the agent enters the
Figure 3.4: Sample 2APL agent (Excerpts from [44])

Beliefs:
  pos(1,1).
  hasGold(0).
  trash(2,5).
  trash(6,8).
  clean(blockWorld) :- not trash(_,_).

Goals: hasGold(5) and clean(blockWorld), hasGold(10)

BeliefUpdates:
  {not carry(gold)} PickUp() {carry(gold)}

Plans:
  [@blockworld(enter(5,5,red),L); ChgPos(5,5)]

PG-rules:
  clean(R) <= pos(X1,Y1) and trash(X2,Y2) |
    {[goTo(X1,Y1,X2,Y2); RemoveTrash()]}

PC-rules:
  event(gold(X2,Y2),blockworld) <- not carry(gold) |
    {getAndStoreGold(X2,Y2)}

world. The enter action is an external action provided by the blockworld environment.

A planning goal rule can be used to implement an agent that generates a plan when it has
certain goals and beliefs. The specification of a planning goal rule consists of three entries:
the head of the rule, the condition of the rule, and the body of the rule. Lines 17 and 18
show a planning goal rule. The head and the condition of a planning goal rule are goal and
belief query expressions used to check if the agent has a certain goal and belief, respectively.
In this example the head of the rule is a query as to the existence of a goal to clean a space
(denoted R). The condition is that we are at a position (X1, Y1) and there also exists trash at
a location (X2,Y2). This body of the plan is uses the variables bound in the queries to move
the agent to the position of the trash and remove it. Plan repair rules are used to replace a
failed plan with another. Finally procedure call rules are use to react to events or messages.
Lines 21 and 22 show the reaction to an event from the environment indicating that gold has
appeared.

It is not explicitly stated that 2APL is compatible with CArtAgO, however earlier versions of
the code base included the required code for integrating the APL with the environment layer.
While the most recent version does not contain this integration the previous existence would
imply that the two pieces of software are compatible. Integrating with EIS required a greater
number of steps than other languages, a two-way converter for the interface intermediate
language had to be developed and the environment loading mechanism had to be replaced
[12].
3.5.3 GOAL

GOAL (Goal Oriented Agent Language) is an agent programming language where the key underlying concept is that of declarative goals [18, 85, 88]. A declarative goal represents a state that is desired to be brought about, this is in contrast to goals in languages such as AgentSpeak and 3APL where a goal is in effect a plan as it represents the desire of an agent to perform some action.

The mental state of a GOAL agent is represented primarily using beliefs and goals, but also allows for the definition of a basic concept of capabilities.

GOAL agent program consists of conditional action rules in the form if X then Y, X is a mental state condition (i.e. a goal or belief, or a conjunction/disjunction of these) and X is an action. This specifies for actions, the conditions under which it will be performed.

Central to the definition of GOAL is the formal operational semantic it is built upon and a theory to prove the correctness of programs based on temporal logic. A simple and intuitive correctness property, which is natural in this context and is applicable to our example agent, states that a GOAL agent is correct when the agent program realises the initial goals of the agent.

Figure 3.5 shows an example of a GOAL agent, it is an excerpt taken from [85]. The figure shows parts of an agent written to solve the classical “dining philosophers” problem. Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when he has both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it’s not being used by another philosopher. After he finishes eating, he needs to put down both forks so they become available to others. A philosopher can grab the fork on his right or the one on his left as they become available, but can’t start eating before getting both of them.

Knowledge is represented using Prolog, lines 3 and 4 show an example of these inference rules. Beliefs and goals (lines 5 and 6) allow the definition of the beliefs and goals that the agent has at the beginning of its execution. This agent will believe that he holds a fork in his left hand and has the goal to hold a fork in both its left and right hand.

The program begins on line 7 and contains a number of rules. Line 10 is a good example of example of a conditional rule, if the agent has the belief that it is hungry it then adopts the goals to hold both forks. This goal will trigger the rule on line 13 which in response to the goal to hold a fork begins the process of negotiating for the release of the forks by its neighbours.

The final section (line 31 to 35) contains the specifications for each of the actions. This includes both pre and post conditions of the action. The action think on line 32, has the precondition that the agent believes it is not hungry and has the post condition that the agent will believe it is hungry.
There is no record of the completion of an integration between CArtAgO and GOAL, there is however such an integration with EIS. The process of integrating the two is described as “quite easy” as most of the functionality in the EIS interface matched to that provided by GOAL.

### 3.5.4 Jadex

Jadex is a BDI layer which is built on the JADE middleware platform [22, 153]. JADE (Java Agent DEvelopment framework) is a Java-based middleware platform designed to facilitate the creation and deployment of MASs[15, 16]. The focus of JADE is the provision of platform services and an internal communication structure. Jadex focuses on the reasoning to the agents themselves.
Jadex agents are specified in two parts, plans, goals and initial belief are specified using an XML format whereas the implementation of these beliefs and plans take the form of Java classes. Beliefs, capabilities, goals and plans are the principal agent-oriented programming constructs provides.

Goals are supported in a number of types, all are contained within the goal base and accessible to the reasoning components. These component reason based on the available plans as to the achievability of the goals within the goal base. The goal types are perform, achieve, maintain and query.

- **Perform** goals are used for executing (possibly repeatedly) certain actions.
- **Achieve** goals add support for the specification of a target and a failure condition.
- **Query** goals are used for information retrieval purposes.
- **Perform** goals are used to monitor a specific world state and automatically tries to re-establish this state whenever it becomes invalid.

```
<agent name="Blocksworld" package="jadex.examples.blocksworld">
  <imports>
    <import>java.awt.Color</import>
  </imports>
  <beliefs>
    <belief name="table" class="Table">
      <fact>new Table()</fact>
    </belief>
    <beliefset name="blocks" class="Block">
      <fact>new Block(new Color(240, 16, 16), $beliefbase.table)</fact>
      <fact>
        new Block(new Color(16, 16, 240), $beliefbase.table.allBlocks[0])
      </fact>
      ...
    </beliefset>
  </beliefs>
  <goals>
    <achievegoal name="clear">
      <parameter name="block" class="Block" />
      <targetcondition>$goal.block.isClear()</targetcondition>
    </achievegoal>
    <achievegoal name="stack">
      <parameter name="block" class="Block" />
      <parameter name="target" class="Block" />
      <targetcondition>$goal.block.lower==$goal.target</targetcondition>
    </achievegoal>
  </goals>
</agent>
```

Figure 3.6: Sample Jadex agent definition (Excerpt from [153])

Figures 3.6 and 3.7 show an example agent taken from [153]. The agent is designed for the classic blocksworld problem, wherein a a number of blocks are placed on a table and an agent is required to achieve particular configurations.

Figure 3.6 shows the initial agent definition. Lines 6 to 8 show the definition of a belief, in this example the `<fact>` tags contain the code responsible for creating the object representation...
of the belief. The definitions of both individual beliefs and of beliefsets contain a the type of
the object that they will store and are associated with a name by which it can be identified
in the agent definition.

Similar to beliefs goals are also identified by a name, lines 21 to 24 show a goal named clear
with a parameter named block. The goal is the intention that the particular block given as a
parameter is “clear”, meaning that there are no other blocks on top of it. This is specified by
the <targetcondition> tag which refers to the method isClear() from the parameter that
was defined, the goal is considered achieved when the return value of the method invocation
is true.

```xml
<plans>
  <plan name="clear">
    <bindings>
      <binding name="upper">
        select $upper from $beliefbase.blocks where $upper.getLower()==$event
          .goal.block
      </binding>
    </bindings>
    <body>new StackBlocksPlan($upper, $beliefbase.table)</body>
    <trigger><goal ref="clear"/></trigger>
  </plan>
</plans>
</agent>
```

**Figure 3.7: Sample Jadex agent plan definition (Excerpt from [153])**

Figure 3.7 shows a single plan, this plan is intended to achieve the clear goal defined in
Figure 3.6 as its trigger is the adoption of the goal “clear” (line 9). The <binding> binding
defined on lines 4 to 6 show the creation of a new variable called $upper and the binding of
its value using an SQL-like query of the belief base. The query selects the block above the
parameter of the goal, this is done by matching the result of the method getLower(), which
returns the block below, with the parameter of the goal. The plan then instantiates a Java
class the defines the steps required to achieve the goal.

In order to integrate Jadex with the EIS layer, a special class was provided to provide the
necessary conversion between the two environment concepts [12]. This allows the ability to
simply specify the environment within the Jadex application descriptor. The integration and
testing of Jadex and CArtAgO is available has previously been detailed [152].

3.5.5 Agent Factory

The Agent Factory Framework is an open source collection of tools, platforms, and languages
that support the development and deployment of multi-agent systems [38, 97, 136, 168].
Agent Factory is provided in two editions, the standard edition for use on most devices and
the micro edition for use on resource-constrained devices compatible with the CLDC Java
implementation [134, 136].
Originally, Agent Factory only supported the Agent Factory Agent Programming Language (AFAPL) and its successor AFAPL2. However, recent work on the Agent Factory framework has focused on supporting heterogeneous logic-based agent architectures with the goal of providing a common toolset that can be adapted to different agent models, ranging from custom Java agents, through to reactive agent architectures and finally to high-level agent programming languages. This toolset, called the Common Language Framework, was used to develop a number of APLs [168]. The languages developed were based on different models. Examples include the AF-TeleoReactive (AF-TR), which is based on Nilssons teleoreactive model [140, 141], AF-AgentSpeak (AF-AS) an extension of AgentSpeak(L) and a hybrid of the two approaches called AF-ASTR, which combines elements of the two approaches. This hybrid approach attempts to combine the BDI reasoning of AgentSpeak with reactivity of Teleoreactive programs and reflects similar work done by Coffey and Clark [36].

The Agent Factory framework is no longer under active development and will not be described in great detail for this reason. Development has continued however, on the hybrid language which is described in Section 3.5.6.

Environment integration was implemented for each of the languages, with different languages offering different levels of support. AFAPL and AFAPL2 used the concepts of Actuators and Perceptors implemented as Java classes, but there was no integration with either of the environment technologies described in Section 3.4. Languages based on the CLF shared a common model of environment, using Sensors and Actions implemented as Java classes. Differences in the operation of the languages sometimes required that implementations of these Actions and Sensors were not common. Support was built into the CLF for the integration of EIS environments, but there was not the same support for CArtAgO environments.

3.5.6 ASTRA

Successor to the AF-ASTR language, ASTRA, which is based on the same hybrid approach, was redeveloped outside the environment of Agent Factory. This redevelopment was undertaken in order to focus on the performance of the language which proved too inefficient to take part in the Multi-Agent Programming Contest (MAPC). The MAPC which allows only 2 seconds for the deliberation of the agents in the contest introduced a qualification round in 2012 [103]. In order to take part teams had to have less than 5% timeout-rates, this was not achievable within the architecture of the Common Language Framework and as such development began on an independent interpreter [39].

Figure 3.8 shows an example ASTRA agent, it is an agent designed to solve the classic blocksworld problem. The mental state of an ASTRA agent consists of a belief base and a goal base. The actual agent itself is composed of a number of components, modules are used to provide extra functionalities to agents, these are generally centred on some concept and grouped together. The EISAPI module provides an agent with the capability of launching and controlling and situating an agent within an EIS environment. Line 3 shows the decla-
ration of a module and lines 9 to 11 show examples of its use, in this case the blocksworld environment is loaded, the agent is lined to the entity gripper and the environment is started.

There are a number of ways to query the belief base of the agent using either if, query or foreach statements which are shown on lines 23, 12 and 17 respectively. Agents actions are defined through the use of plans and rules. Plans are the means of associating a number of steps with a name and parameters, pickup on lines 36 to 39 shows a plan for picking up a block and waiting for the block to be held.

Rules are similar to their Jason equivalents, they contain a triggering event, often the addition of a belief or goal, an optional context and a body. The triggering event for rule shown on lines 22 to 28 is the addition of the goal to clear a block, there are two different rules that
are selected base on the context, in this case the context is that the block we are to clear is not on the table. The body of the rule is very simple, we first check if there is a block above us using the if statement. If such a block exists its name is bound to the string S and we generate a new goal to clear the block above us, once completed we are free to pick up the current block and place it on the table.

ASTRA was designed and built with the integration of both EIS and CArtAgO environments in mind. Beliefs and events for these environments are incorporated into the syntax of the language but distinguished from the beliefs and events of the agents internal mental state. This is quite apparent in the example shown in Figure 3.8, as the agent is based on an EIS environment, any query of the state of the environment must be preceded by “EIS->”.

3.5.7 SimpA

SimpA was designed to support the development of complex, multi-threaded / concurrent applications, simpA is a library-based extension of Java providing an agent-oriented abstraction layer on top of the basic OO layer to organize and structure applications [164]. simpA is based on the A&A (Agents and Artifacts) meta-model, introduced in the context of agent-oriented programming and software engineering as a novel basic approach for modelling and engineering multi-agent systems [162, 163]. Java annotations are exploited to define simpA and the new programming constructs required: consequently, a simpA program can be compiled and executed using the standard Java compiler and virtual machine.

The long term memory of the agent is implemented as an associative store called memo-space, there an agent can attach, read and retrieve chunks of information called memos which are tuples characterised by a label, ordered arguments and possibly some object.

Figure 3.9 shows an excerpt of a simpA agent taken from [164]. It is designed to operate in a coffee show environment. Activities are methods that have been annotated with @ACTIVITY and represent atomic actions of the agent. The method makeOneCoffee is an example of such an atomic action, it contains all the necessary environmental interaction necessary to complete the action. In order to complete more complex tasks, the idea of an agent is introduced, allowing the specification of the set of sub-activities by annotating them with @TODO. These complex activities are methods annotated @ACTIVITY.WITH_AGENDA and upon execution each TODO is executed as soon as their pre-conditions hold.

The development of simpA is tied closely to the development of CArtAgO and as such no work is required to integrate the two technologies. As far as integration with EIS is concerned, there is no description of an integration or any attempt to do so, as such the work required in integrating the two is unknown.
public class Waiter extends Agent {

    @ACTIVITY_WITH_AGENDA({
        @TODO(activity="makeOneCoffee"),
        @TODO(activity="makeOneTea"),
        @TODO(activity="deliverBoth", 
            pre="completed(makeOneCoffee),
                completed(makeOneTea)"),
        @TODO(activity="deliverJustCoffee",
            pre="completed(makeOneCoffee),
                memo(tea_not_ready)"),
    }) void main() {}

    @ACTIVITY void makeOneCoffee() throws Exception {
        SensorId sid = linkDefaultSensor();
        ArtifactId id = lookupArtifact("cmOne");
        use(id, new Op("selectCoffee"));
        use(id, new Op("make"), sid);
        sense(sid, "making_coffee");
        Perception p = null;
        IPerceptionFilter filter = new GenericFilter("property_updated", "
            sugarLevel");
        focus(id, sid);
        do {
            use(id, new Op("addSugar")); p = sense(sid, filter);
        } while (p.doubleContent(1) < 0.5);
        Perception p1 = sense(sid,"coffee_ready",5000); memo("drink1", p1.
                getContent(0));
    }
}

Figure 3.9: Excerpts of sample simpaA agent (taken from [164])

3.6 Discussion

This chapter introduced the concept of agent-oriented programming and within this context described the notion of agency adopted for use with this work. Following from this, Section 3.4 describes two candidate technologies for use as an interface between intelligent agents and their environment. The primary distinction between the two was down to the compatibility of the two platforms with other technologies, specifically OSGi which is in compatible with the Environment Interface Standard (EIS).

CArtAgO was chosen as the environment interface primarily because of the compatibility issues between using EIS and OSGi but additionally as due to its more distributed nature facilitating the placement of artifacts with relevant code [114].

A number of agent-oriented programming languages were introduced, these were chosen for their demonstrated compatibility with at least one of the environment interfaces at some point. Table 3.2 shows the languages that were included within the study and the level of support that is provided for the most recent versions. As work has progressed some of the
languages no longer provide built in support, however as it is likely that the integration could be achieved they are still included within the review.

Table 3.2: Environment Interface support

<table>
<thead>
<tr>
<th>Language</th>
<th>EIS Support level</th>
<th>CArtAgO Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jason</td>
<td>Not included(^1)</td>
<td>Built in</td>
</tr>
<tr>
<td>2APL</td>
<td>Not included</td>
<td>Not included(^2)</td>
</tr>
<tr>
<td>GOAL</td>
<td>Not included(^1)</td>
<td>Not included</td>
</tr>
<tr>
<td>Jadex</td>
<td>Built in</td>
<td>Built in</td>
</tr>
<tr>
<td>Agent Factory</td>
<td>Built in</td>
<td>Built in</td>
</tr>
<tr>
<td>ASTRA</td>
<td>Language level</td>
<td>Language level</td>
</tr>
<tr>
<td>simpA</td>
<td>Not included</td>
<td>Language level</td>
</tr>
</tbody>
</table>

A distinction is made between the levels of support provided by the languages, which can be describe as not included, built in or language level. Language level describes support where the syntax of the language itself is designed with an environment interface in mind. This is the case in only two of the languages, simpA and ASTRA. ASTRA was chosen for use in this work not only due to a higher level of support, but because it presents a higher level language than simpA.

\(^1\)While an implementation is not provided, it is described as straightforward 
\(^2\)Current version, does not include support but previous versions have
Chapter 4

Pattern Recognition

During the past decade a large amount of research effort has been dedicated to the classification of activities through the use of body mounted sensors accumulating acceleration data [20, 127, 128, 138, 157, 197, 204]. This arguably is driven by the growing concerns over the links between levels of physical activity and health problems such as cardiovascular disease, diabetes and osteoporosis [200]. This process involves the recognition of patterns within the gathered data and based on these patterns making a determination as to the class of the activity. Typical classes in this type of research would be walking, jogging, running, climbing stairs etc.

The classification of object state requires many of the same techniques as human activity classification, but the same level of interest in the research community is not present. The techniques required to successfully classify human activities have previously not been applied to the problem of object classification within a transportation context in any depth.

This Chapter begins by describing the process of Pattern recognition. This is followed by a brief explanation of human activity classification and the techniques that are applicable to the classification of object state within an intelligent transportation application. Finally other works are discussed that are linked to both intelligent transportation and to Pattern recognition.

4.1 Pattern Recognition

Pattern recognition is a branch of machine learning that focuses on the recognition of patterns and regularities in data. These patterns are generally identified using tools from statistics and probability amongst others. Identifying these patterns is often difficult because each can contains a large amount of information. Typical examples are speech recognition or computer vision algorithms. Within the context of this thesis, the focus will be on the classification of human activity (and later object state) based on readings from acceleration sensors attached to the person/object.
The process begins with the acquisition of data, as this is discussed in Chapter 5, it is not detailed here. Once the data has been acquired, it must be converted to a usable format. As the data in this system is based on a time series, it must be grouped together into usable time periods. This time periods are referred to as windows and the selection of the length of the periods will be described later. Based on this windows of data a number of measurable properties are calculated. These properties are referred to as features and are used to determine to which class the data belongs. The process of calculating these is called feature extraction and can take any form such as a simple average or a complex edge detection in an image. Features are typically used in combination, the particular features that we are using to determine the class are called the feature set. The calculated value of each feature within a set for a single collection of data is referred to as an instance.

Before we can utilise the classifier, we must first identify a way of discriminating between the data from the different classes. This is done by supplying the classifier with a number of instances for which the correct classification is already known. This process, referred to as training, allows the classifier to construct an internal model of the relationships between different values for features within these instances and the correct classification. Based on this model the classifier can now make a determination as to the probable class of any new instance.

4.2 Data Preprocessing

The raw data received from the sensors is often processed before features are extracted. This step is performed for a number of reasons. Typical examples of preprocessing would be sampling or interpolating in order to reduce the data rate, combination of multiple signals to reduce the overall amount of data, or the actual process of forming continuous data into discrete windows for further analysis.

4.2.1 Windowing

Windowing is the process of converting continuous data into discrete windows which can be used within the classifier. All windows are of a consistent length, however the period of time can vary depending on the application. This time period is usually chosen either based on an initial pilot study or previously carried out research.

Once the data is grouped together, features are extracted for use within the classifier. Often there is an overlap between windows in order to improve the probability of the event we are looking for being central within one of the windows it occurred in. For human activity classification several previous studies have chosen an overlap of 50% and windows of approximately 2 seconds [108, 143, 157]. This means that each piece of data is examined as a part of two windows, thus increasing the chances of successfully detecting the correct class at a given
4.2.2 Signal Combination

The acceleration values used within this thesis and other studies of activity classification are separated into three components, the acceleration with respect to the X, Y and Z axes. A number of studies have examined the combination of these three signals into a single acceleration signal. This can be viewed as the process of calculating the magnitude of the vector represented by a single x, y, and z value. Figure 4.1b shows the result of the aggregation on an example window of data from the falling state. If we view a triaxial acceleration signal as three sequences of values, namely X, Y and Z all with length n, then the combined acceleration, M is defined as the values \( (m_1, \ldots, m_n) \) as defined by Equation (4.1).

\[
m_i = \sqrt{x_i^2 + y_i^2 + z_i^2}
\]  

(4.1)

This aggregation has been used previously in studies of human movement patterns [128] as well as in studies of fall detection amongst the aged [197].

4.2.3 Data Reduction

In signal processing, the minimum frequency for sampling a phenomenon is given by its Nyquist rate, which is twice the highest frequency component of the phenomenon [72, 144]. As it is not always apparent what the highest frequency component will be, often data is sampled at a much higher rate than required and converted to a lower rate if necessary. This process is called *sample rate conversion* and while it can be used to approximate a higher frequency signal, in digital signal processing it is usually used to reduce the frequency of the signal.
In practice it is possible to reduce the frequency of the data by removing readings from the original signal. For example if the data was sampled at 250 hz, then if were remove every second sample, we produce a 125 hz sample of the same phenomenon. If we wanted to produce a signal that was for example 200 hz from our original signal, the process would not be as easy. This new sampling rate does not overlap with the periods of the original signal and therefore we are required to interpolate the missing values based on the values in the vicinity.

4.3 Feature Extraction

Feature extraction is the process of analysing the large volume of data generally contained within a single window and reducing it to a set of descriptive features which are generally much less numerous. There are a large number of techniques for feature extraction from time series data. Based on previous studies of human activity classification the most commonly used techniques are time domain features such as average and standard deviation or features calculated from a discrete wavelet transform (DWT) or discrete Fourier transform (DFT) performed on a window of data. The following sections introduces the feature extraction techniques used from each of the above domains starting first with features extracted from the DWT of the windows, then the features from the DFT of the windows and finally the features extracted from the time-domain signal.

4.3.1 Wavelet Features

With this approach, the original time-domain signal (maximum frequency \( f \)) is initially decomposed into detail and coarse approximation information by high-pass filtering \([f/2, f]\) and low-pass filtering \([0, f/2]\) respectively \([123]\). This process can be repeated through subsequent levels of decomposition during which the approximation signal from the previous level is split into a second approximation and a detail coefficient. This process is repeated to the desired decomposition level. Features were then extracted from the detail coefficients at differing levels, each referred to as \( cDa_n \) where \( n \) is the level used and \( a \) is the axis.

Equation (4.2) is an example of a feature set based on the discrete wavelet transform proposed in \([186]\). Features such as these are common amongst activity classification literature (e.g. \([143, 157, 186, 205]\)). The features are defined as the sum of the squared detail coefficients at levels 4 and 5. As this process is applied to each of the acceleration signals individually, this produces a total of 6 features for each window of data

\[
\|cDX_4\|^2 \& \|cDX_5\|^2 \& \|cDY_4\|^2 \& \|cDY_5\|^2 \& \|cDZ_4\|^2 \& \|cDZ_5\|^2
\]

(4.2)
4.3.2 Frequency Domain Features

A Fourier transform converts time to frequency and vice versa. With this approach a discrete Fourier transform (DFT) is performed using a fast Fourier transform (FFT) algorithm. When using an FFT algorithm it is usually more efficient, depending on the implementation, to process a number of samples that is a power of 2. Often windows of data are padded with zero values at the end to bring the size to a power of 2 which greatly reduces the computation time of the process.

A typical example of a frequency domain feature set would be spectral energy, proposed in [184]. Spectral energy, which is defined to be the sum of the squared FFT coefficient provides only a single feature for each of the axes.

4.3.3 Time Domain Features

Time Domain features are commonly employed in the classification of daily activities in humans such as [6, 59, 101, 129, 150, 157]. A typical example would be the calculation of mean (average) and standard deviation (SD) as well as the median (50th percentile), 25th and 75th percentiles of the values for each axis [59].

4.4 Classifier

In pattern recognition the algorithm that is used to determine the class of a given set of features is referred to as a classifier [49]. These generally function by generating a model based on an amount of previously classified data, which can then be used to determine what class any new unclassified data belongs to [82]. In the field of human activity classification a number of different classification algorithms have been tested. Amongst these Decision Trees [170], Naive Bayes [167] and Nearest Neighbour [96] are most commonly utilised. The following sections will give a brief introduction to each of the aforementioned algorithms. Other notable algorithms that are less commonly applied in human activity classification, such as Support Vector Machines [27] or neural networks [78] are omitted.

4.4.1 Decision Trees

A decision tree is a flowchart-like structure in which internal node represents test on an attribute, each branch represents outcome of test and each leaf node represents class label. A path from root to leaf represents classification rules. Figure 4.2 shows a trivial example of a decision tree, it operates only on a single value of the temperature and based on the tests shown in the boxes classifies the temperature as either cold, warm, hot or very hot.
Decision trees, as the name implies have a tree structure, this means that there cannot be any convergence. As decision trees are often used where there is more than one variable and the classes are much harder to distinguish, decision trees can grow very large. Decision trees have proven useful in activity classification (E.g. [9, 59, 101, 129, 150]) providing accuracies of up to 97% [150]. Despite its often good results, it is possible that the decision tree may be shaped too well by the data that is used to train it, this process is referred to as overfitting [115].

4.4.2 Naive Bayes

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes’ theorem with strong (naive) independence assumptions. This assumption of independence means that the classifier will assume that the value of a feature is unconnected to any other feature. Table 4.1 shows an example set of training data for a naive bayes classifier taken from [130], it shows three attributes describing cars and a classification as to whether the car was stolen or not.

If we want to classify red domestic SUV, then we need to compute the probability that the class is yes and the probability that the class is no. We calculate the following probabilities;

- $P(\text{Red}|\text{Yes}) = .56$
- $P(\text{SUV}|\text{Yes}) = .31$
- $P(\text{Domestic}|\text{Yes}) = .43$
Table 4.1: Example Data Set

<table>
<thead>
<tr>
<th>Color</th>
<th>Type</th>
<th>Origin</th>
<th>Stolen?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Sports</td>
<td>Domestic</td>
<td>Yes</td>
</tr>
<tr>
<td>Red</td>
<td>Sports</td>
<td>Domestic</td>
<td>No</td>
</tr>
<tr>
<td>Red</td>
<td>Sports</td>
<td>Domestic</td>
<td>Yes</td>
</tr>
<tr>
<td>Yellow</td>
<td>Sports</td>
<td>Domestic</td>
<td>No</td>
</tr>
<tr>
<td>Yellow</td>
<td>Sports</td>
<td>Imported</td>
<td>Yes</td>
</tr>
<tr>
<td>Yellow</td>
<td>SUV</td>
<td>Imported</td>
<td>No</td>
</tr>
<tr>
<td>Yellow</td>
<td>SUV</td>
<td>Imported</td>
<td>Yes</td>
</tr>
<tr>
<td>Yellow</td>
<td>SUV</td>
<td>Domestic</td>
<td>No</td>
</tr>
<tr>
<td>Red</td>
<td>SUV</td>
<td>Imported</td>
<td>No</td>
</tr>
<tr>
<td>Red</td>
<td>Sports</td>
<td>Imported</td>
<td>Yes</td>
</tr>
</tbody>
</table>

- $P(\text{Red}|\text{No}) = 0.43$
- $P(\text{SUV}|\text{No}) = 0.56$
- $P(\text{Domestic}|\text{No}) = 0.56$

Given these values we calculate the

$$P(\text{Yes}) * P(\text{Red}|\text{Yes}) * P(\text{SUV}|\text{Yes}) * P(\text{Domestic}|\text{Yes}) = 0.037$$

and

$$P(\text{No}) * P(\text{Red}|\text{No}) * P(\text{SUV}|\text{No}) * P(\text{Domestic}|\text{No}) = 0.069$$

Given that 0.069 is the larger value we classify this example as No.

Naive Bayes classifiers have been used within human activity classification [9, 129], it is however in much research not chosen for use in favour of better performing algorithms [129]. Other studies have reported accuracy as high as 52% when using a naive bayes classifier [9].

4.4.3 Nearest Neighbour

Nearest neighbour classifies new data using a distance function to measure the difference between the new example and those in memory. The class of the most similar example is then used to classify the new one. Nearest neighbour is a method that originated in statistics. It was first considered for rule production by Fix and Hodges [64], who performed an initial analysis of the properties of k-nearest neighbour systems, and established the consistency of the method as k varies from one to infinity.

A distance function is used to determine similarity. For numeric attributes this is usually based on Euclidean distance, where each example is treated as a point in an n-dimensional feature space. It assumes that for a given point in the feature space the surrounding area will share the same class. For the Euclidean distance function to work well the examples must be clustered into relatively few dense regions in feature space that share a common class.
A specific variant of the nearest neighbour algorithm known as k-nearest neighbour (kNN) has been used successfully in a number of studies of activity classification [9, 108, 129, 151, 157]. Amongst the reasons sighted are the reliability, accuracy and the ease of implementation, furthermore the inclusion of a nearest neighbour based algorithm, nearest neighbour generalised exemplars (NNge) [126], within the machine learning toolkit WEKA [80] was the reason this algorithm was chosen for use within this thesis.

4.5 Testing

When finding the efficacy of different classifiers of feature extraction techniques, it is necessary to perform a number of tests based on the training data. Typically the training data is randomly separated into two sections not necessarily equal in size. Then one of the sections is used to train the classifier and the other portion is used as test data. As all the data is previously classified, often by hand, the predicted class can be prepared with the actual class of the data.

Cross validation is a popular statistical re-sampling procedure [49], in $n$-fold cross validation data is split randomly amongst $n$ equally sized subsets. Each subset is then used once as to test the classifier trained with the remaining $n - 1$ subsets. The empirical accuracy is given by the average of the accuracies of these $n$ classifiers. This improves thoroughness of the testing when compared with the more traditional method of sub-sampling where the data is split into only two sets, one for training and one for testing.

4.6 Pattern Recognition in Intelligent Transportation

A large volume of research exists within the confluence of intelligent transportation and Pattern recognition. However most of this research is on topics outside the scope of this thesis, examples such as automatic licence plate recognition [7, 43, 193], traffic sign recognition [60] and travel time prediction [201] while interesting do not relate to the work of this thesis.

Only one similar study was identified which performed a similar classification of object state, the study by Sarangan et al. [24, 176] focused on objects moving whilst in transit rather than being manipulated by humans. The study was based on four states, wobbling, collision, tilting and sliding, and using wavelet decomposition reported accuracies of 100% for the first three states and 85% for the final state.

The study utilised biaxial accelerometers at a sampling rate of 233Hz to record data for evaluation, but the authors believe that a sampling rate in excess of 2500Hz is needed for effective container monitoring. As the focus of the work was the device developed, very little detail about exactly what features were extracted and what classification techniques were applied was presented.
4.7 Discussion

This chapter gives an overview of Pattern recognition and the steps within this process. A number of preprocessing techniques are described in Section 4.2, these are generally used to convert large amounts of raw data to manageable chunks or provide a means to reduce the volume of data. The concept of extracting features from data is discussed and three of the most common types of techniques used within human activity recognition are introduced.

The purpose of classifiers is discussed and three of the algorithms most commonly applied to human activity tracking are introduced. Based on the proven results achieved in other studies [9, 108, 129, 151, 157], the nearest neighbour algorithm was chosen for use within this work. Specifically a version provided within the machine learning toolkit Weka [80], nearest neighbour generalised exemplars (NNge) [126], was chosen.

The process of evaluating the performance of classifiers is discussed, and the method of cross validation was chosen for use in this work because it provides an increased thoroughness when compared to more traditional sub-sampling. Areas in which the research of Pattern recognition and intelligent transportation are combined are discussed, a particular focus is placed on an example which is closely related to this work but not directly comparable.
Chapter 5

Wireless Sensor Networks

Wireless Sensor Networks (WSNs) have received a much attention in the research community, doubtless due to both the potential of such devices to enhance our lives in the future and the technical challenges that must be faced. The vision of the future alluded to in the beginning of this thesis may be a long way off, however since the realisation of the first devices a large number of applications have been developed.

Many areas of wireless sensor networks receive attention within the research community, however applications with a military application also receive the attention of governments. Example of military applications of wireless sensor networks are localising a shooter within an urban environment [124], battlefield surveillance [109] as well as a host of other examples [51].

Other applications of wireless sensor networks exist within other domains such as healthcare, education and environmental monitoring. These devices enable the monitoring of a patients vital signs remotely [70] or can aid in the effective deployment of emergency responders in the event of a mass casualty event [69].

In education, researchers have utilised wireless sensor networks within schools in an attempt to encourage the development of problem solving skills. Environmental based wireless sensor network research has been applied in a number of varied ways from monitoring volcanic eruptions [194], assessing agricultural enterprises [26] and in tracking wildlife and their habitats [17, 98, 122].

This chapter discusses the concept of wireless sensor networks and the middlewares that enable their use. Wireless sensor networks are utilised in the work of this thesis, but as they are not the focus of this work, the descriptions in this chapter will be very high level. In order to facilitate the usage of wireless sensor networks within the work of this thesis, a middleware was utilised. The following sections will introduce the terminology of wireless sensor networks and more specifically a number of middleware solutions will be detailed.
5.1 Wireless Sensor Networks

The concept of wireless sensor networks evolved naturally from technological advances that led to a reduction in size, power requirements and cost of digital circuitry [4]. This enabled the creation of very compact mobile computing platforms containing a number of sensors, computation and communication abilities and a power supply. These wireless sensors, when considered in isolation may seem underpowered and of little use. However, when used to collectively solve a problem, a network of these devices are more than capable. The idealised concept of what these devices will become when reaching millimetre scale is referred to as “Smart Dust”, where the device is so small it may be suspended in the air [99].

Figure 5.1 shows a representation of a typical wireless sensor network, in which all data is forwarded to a single node connected to a computer. This connected node is referred to as a gateway sensor node, sink or basestation. The devices, when used within a network are referred to as sensor nodes or simply nodes. At present technological levels are capable of sensing numerous phenomenon, initially devices such as the Crossbow TelosB [156] focused on temperature and humidity, further devices appeared with sensors for light, passive infra-red and acceleration. The latest generation of devices are more likely to allow the attachment of heterogeneous sensors to motes, platforms such as the Wasp mote by Libelium [111] allow the use of a wide variety of gas, liquid, soil and presence sensors.

Figure 5.1: Representation of wireless sensor networks

5.2 Challenges of Wireless Sensor Networks

Wireless sensor networks offer much in the way of capabilities, however there is a price to be paid for this; the complexity in designing and utilising a wireless sensor network is high. Users must be comfortable with the design of low level device programming in languages such as TinyOS [71], complex routing protocols [5], duty cycles [149] and power conservation [74] in order to successfully utilise the devices. All this is before the user designs and builds whatever system they intend to use this data. This challenge has led to the design and creation of a number of middleware solutions with the goal of minimising the effort and making the creation
of applications on a wireless sensor network as easy as possible. A select number of these will be discussed in Section 5.3.

One of the greatest challenges faced is the limited energy reserve or recovery ability. Researchers wishing to maximise the life-time of networks have focused much attention on this problem. An often used strategy, referred to as duty cycling, is the use of redundant nodes within the network, these nodes can then alternate in a coordinated manner between active and sleeping states such that the network can perform its task and the resources of the devices are extended.

A further challenge faced is that of routing information through the network, there are many concerns to this process such as Quality of Service, timeliness or redundancy, however the challenge of limited power is again paramount. Research by Pottie and Kaiser calculated the cost of transmission of 1Kb of data a distance of 100 meters was approximately equivalent to processing 3 million instructions [154].

Flexibility of wireless sensor networks is also an important requirement. Once deployed, it is unlikely that the nodes of the network can be easily regathered for reprogramming. As such wireless sensor networks must have a degree of flexibility to allow the network operation be altered to take advantage of a favourable change in the environment or recover from an unfavourable one. Reconfiguring or retasking a wireless sensor network at runtime can, if unprepared for, require the reprogramming of the entire network. This process can be done wirelessly but at a great energy cost [79].

5.3 Middleware for Wireless Sensor Networks

A number of applications have been produced for wireless sensor networks, the majority of these applications are designed and built from the ground up without making use of a higher level middleware abstraction [133]. There are many possible reasons for this but one posited is the resource limited nature of devices being used where ”For better resource utilization, layers blur and blend together, to the point that placing a middleware layer in a wireless sensor network design becomes difficult, even conceptually”.

Within the work of this thesis it was decided that a middleware would be used for a number of reasons; primarily to remove the need for low level programming of devices, which can require as much effort if not more as the application itself. Secondly use of a middleware can potentially allow the application be independent of any particular type of sensing devices. Finally this decisions keeps the focus of the work on the application itself.

There are a large number of middlewares for wireless sensor networks [42, 67, 110, 117, 137, 181]. This section reviews a number of these wireless sensor network middleware solutions. Following a review of the area, a select number of middlewares were considered for use. Based on the available information describing each solution, inclusion within the review was decided
on the perceived suitability of the middleware to the proposed tasks. Factors considered were: the maturity of the project, the structure of the middleware and its interoperability.

In order to determine which middleware would be utilised a number of factors were considered. The following abilities were considered if not necessary, at the very least desirable:

- Supports multiple sensing platforms;
- Runtime reconfiguration or retasking;
- Integrated data acquisition and processing;
- Allows integration with agent-oriented programming languages;

5.3.1 Global Sensor Network

Global Sensor Network (GSN) is a flexible middleware layer which abstracts from the underlying, heterogeneous sensor network technologies [173]. Figure 5.2 shows the model used by GSN, where the network delivers all data to a sink node which is connected to a more powerful base computer. The GSN middleware is run on the base computer, which may participate in a network of base computers.

![Figure 5.2: GSN Model (Taken from [173])](image)

GSN models sensors as Virtual Sensors which are defined using an XML format. This virtual representation to aid both runtime reconfiguration and data acquisition and processing. GSN provides APIs and documentation describing the process of using the data received within another application but does not provide a mechanism for integration with agent-oriented programming languages. While capable of supporting local applications, the core focus of GSN is the distribution of data over the internet in a myriad of ways [172].

5.3.2 TinyDB

TinyDB is designed to hide the difficult and time consuming process of low level device programming on the TinyOS platform [120]. This is achieved through the provision of a SQL-
like query language named *Tiny-SQL*, which is used to gather data from the sensor network. The goal of the system is the provision of an interface for getting data without specifying how to get it. Figure 5.3 shows a sample query in which each node of the network should report its own identifier (*nodeid*), light and temperature readings once per second for 10 seconds.

```
1 SELECT nodeid, light, temp
2 FROM sensors
3 SAMPLE PERIOD 1s FOR 10s
```

Figure 5.3: Sample TinyDB Query (taken from[120])

Results of this query stream to the root of the network, where they may be logged or output to the user. The output consists of a stream of tuples, clustered into 1-s time intervals. Each tuple includes a time stamp corresponding to the time it was produced.

As TinyDB is written in Java, it could easily be used as the core of a wireless sensor network application and the requisite integration with agent-oriented programming languages could be built upon this. The functionality provided is exactly what is required for the work of this thesis, however it can only be used with TinyOS devices and the application described herein utilises Oracle SunSPOTs for the acquisition of data.

### 5.3.3 Reconfigurable Ubiquitous Network Embedded Systems (RUNES)

RUNES supporting middleware is based on a two-level architecture: the foundation is a language-independent, component-based programming model that is sufficiently minimal to run on any of the devices typically found in networked embedded environments. Above this is a layer of software components that offer the necessary middleware functionality. The middleware functionality is separated into self-contained components that can be selectively and individually deployed according to current resource constraints and application needs.

RUNES is implemented for a number of different development environments including Java and C and is compatible with devices running the Contiki [50] operating system. Figure 5.4 shows the software architecture of RUNES, in which a large emphasis is placed on separating application and middleware from low level implementation. All application components are independent of the operating system through the use of the kernel API.

The Contiki operating system supports a number of devices produced by different manufacturers, however the equipment used within this work is not supported.

### 5.3.4 Cougar

Cougar is quite similar in operation to TinyDB, providing a query language through which data can be gathered. The sensor network is programmed through declarative queries which abstract the functionality of a large class of applications into a common interface of expressive
queries [46, 203]. The view taken in the design of Cougar is that a number of applications will share use of a wireless sensor network. As such Cougar automatically optimises and combines queries from different applications in order to minimise the energy usage in satisfying a query.

Cougar makes a few assumptions about the wireless sensor network in order to fully optimise its use. It is assumed that the nodes in the network are stationary and laid out in a grid like fashion. This enables the most efficient dissemination of queries and receipt of data, but imposes an inflexibility that is not desirable.

5.3.5 SIXTH

SIXTH [30] is a Java-based Sensor Web Middleware incorporating sensed data from diverse data sources both physical and cyber [28, 31, 147]. Through the adaptor layer abstraction the middleware can be connected with any data source that is accessible programatically. Examples of data sources include those from Wireless Sensor Networks composed of SunSpots, WaspMotes, Shimmers or smartphones and online resources such as Twitter, Xively, Foursquare and Facebook.

SIXTH uses the concept of an *adaptor*, which typically represents a particular sensor network. For example, in the case of SunSPOT motes, an adaptor would run on a machine connected to the SunSPOT base station and provide access to all sensors within the network. SIXTH enables the retasking of sensor through a Tasking Service, which routes the request to the pertinent adaptor for execution.

As SIXTH is an OSGi-based system implemented by means of various component bundles, the choice of this middleware for use within the work of this thesis is partially motivated by previous work done in the area of the management of component-based systems using intelligent agents [113]. SIXTH forms the core of all work within this thesis and as such will be discussed in greater detail later.
5.4 Technology Choice

A number of middlewares are discussed as candidate technologies for use within this work. Table 5.1 show a comparison of the middlewares based on the desirable features identified earlier. None of the middlewares offer compatibility with agent programming languages. However, the component based nature of both RUNES and SIXTH facilitate the integration of such a component. Only two middlewares offer the same level of interoperability with heterogeneous devices, namely GSN and SIXTH. Both systems require the implementation of device specific code to allow the use of new devices.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Device Compatibility</th>
<th>Run-Time Retasking</th>
<th>Agent Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Sensor Network</td>
<td>Acquisition and Processing</td>
<td>All(^1)</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>TinyDB</td>
<td>Acquisition</td>
<td>TinyOS</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>RUNES</td>
<td>Acquisition and Processing</td>
<td>Contiki</td>
<td>No</td>
<td>None(^2)</td>
</tr>
<tr>
<td>Cougar</td>
<td>Acquisition</td>
<td>TinyOS</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>SIXTH</td>
<td>Acquisition and Processing</td>
<td>All(^1)</td>
<td>Yes</td>
<td>None(^2)</td>
</tr>
</tbody>
</table>

As only one middleware satisfies all criteria, namely SIXTH, it was the natural choice for use with this work. Additionally, this is further solidified as SIXTH is developed within the same research group, allowing its development be influenced by the author. The following section will give an introduction to the core terminology used within SIXTH.

5.5 SIXTH

Within SIXTH a number of core concepts are necessary to represent key abstractions of important wireless sensor network components. It should be noted that this information is present here for reference only and was primarily the work of others. A more in depth description of the functionality of SIXTH is available in the SIXTH Middleware Guide [29].

5.5.1 Terminology

The following sections explain core concepts within SIXTH which are relevant to this work and describe how they are used to build wireless sensor network applications.

\(^1\)Requires the implementation of adaptor/wrapper.
\(^2\)Not supported, but extensible design allows addition
Sensor Data

There is a high degree of heterogeneity in the data formats presented by sensor hardware, this problem is further compounded by multiple users with varying goals extracting different data. Within SIXTH sensed data is encapsulated into a common interface in order for an application to reason about data from different sources more easily. The ISensorData interface once implemented, allows the object be viewed as data and passed within the middleware.

Adaptors

An adaptor is a means to collect sensor data from the producer or intermediaries. It acts as a software wrapper around the code required to gather the data, so that this heterogeneous data can be made available to the middleware. Sensor network adaptors present a uniform interface to heterogeneous sensing platforms, allowing the graceful integration of sensor networks into the middleware. For each type of sensor platform utilised an adaptor is required. The heterogeneous data formats of these platforms are transformed within the adaptors into objects which extend the ISensorData interface discussed earlier, thus enabling the goal of data independence from sensor platform implementation. An adaptor should also implement a retasking method which allows for reconfiguration of the sensor network or single nodes within the network.

Receivers

SIXTH utilises the publish/subscribe model for data dissemination, receivers typically represent the simplest version of the subscriber role where the sensor or the respective adaptor can be viewed as the producer. There are a number of different types of receivers that are designed to separate classes of data.

- **Data receivers** are utilised for the data streams originating from the adaptors. These types of receivers are frequently incorporated into an application directly. There are two types, the first is the IDataReceiver where all data is received and the second is the IQueryDataReceiver where the data is filtered before reception.

- **Retasking receivers** receive *tasking messages* from the adaptors when a retasking command has been issued. The message received allows the receiver to determine the effected sensors and the type of retasking to be performed.

Querying Data

Basic data receivers by default receive all of the sensor data that is made available to the middleware, this may often be useful, but for complex applications, or for SIXTH deployments supporting multiple applications, it may be desirable to gain access to only a subset of
interesting sensor data. Queries are created that match against the required data and when matching data is sent to the core by a sensor, the Data Broker sends it to the application that registered the query.

In SIXTH, a query is simply an object that implements the `IQuery` interface. For common query types a collection of prepared implementations are available. The simplest examples contain boolean expressions that check if some aspect of the data, such as the modality, is as required. Provided also are objects which emulate conjunction and disjunction over `IQuery` objects, this allows the representation of queries with arbitrary complexity such as 

\((\text{Modality} = \text{Temperature} \land \text{value} > 12) \lor \text{Modality} = \text{Humidity})\).

**Discovery Service and Data Broker**

The discovery service provides access to the sensor nodes, adaptors and the Data Broker where queries can be registered for filtered data receivers. The data broker is responsible for monitoring incoming sensor data and passing on data that matches queries.

**5.5.2 Using SIXTH**

The previous section has introduced the terminology associated with SIXTH, this section serves to highlight the functionality utilised in realising the WAIST system.

**Adding Data**

Where data needs to be introduced into the middleware an adaptor is used. There exists an abstract implementation of a sensor network adapter, called the `AbstractSensorNetworkAdaptor`, which provides required functionality such as adding data and sensor information. This class is extended to add the required specific functionality for a particular bundle.

Basing all adaptors within the system on this class removes all difficulty with respect to introducing data. Once the data is stored within an object implementing the `ISensorData` interface, the user needs only to pass it as a parameter to the addData method of the above class and the middleware will handle the rest.

**Getting Data**

Once added to the middleware, accessing data requires the use of a receiver. There are a number of different ways in which this can be done, such as receiving all the raw data in a class implementing the `IDataReceiver` interface, however using the data broker is more efficient.
In order to use the data broker, a class implementing the IQueryDataReceiver is required. The process for registering interest in data is illustrated in Figure 5.5. Initially a query is required, this can be of arbitrary complexity, secondly the class offers its credentials (not discussed in this work) and requests the data broker. Finally the query is registered with the data broker and all matching data will trigger the receive callback.

```java
IQuery query = new ModalityQuery(new Modality("GPS Data"));
IDataBroker db = SIXTHMonitor.getDiscovery(getCredentials()).getDataBroker();
db.registerInterest(query, this);

public void receive(final ISensorData data, final IQuery query) {
    ...
}
```

Figure 5.5: Registering for filtered data in SIXTH

The integration of the environment interface described in Section 3.4 and SIXTH was completed as part of this work and will be detailed in Chapter 7.

5.6 In-Network Intelligence

The work within this thesis investigates the introduction of intelligence into an intelligent transportation application. This investigation does not include the introduction of in-network intelligence, as there exist many examples of this research [3, 66, 105, 135].

The primary motivation of in-network intelligence is to reduce the energy usage by having reasoning carried out on nodes rather than outside the network after being transmitted. While it may appear as the natural progression of this work, concerns over admissibility of the data instigated a choice to transmit and store all gathered data in this application. As such no energy saving can be achieved. Furthermore the ability of agents within the applications to exert control of the devices in the network through the middleware is considered sufficient.

5.7 Discussion

This chapter introduces the concept of wireless sensor networks at a high level. This is primarily to familiarise the reader with the terminology that will be used within this thesis. A concept of wireless sensor network middlewares is introduced, and a number of popular middlewares are described. Based on a number of criteria, the SIXTH middleware was chosen for use within this work.

Following this, the terminology and functionality of SIXTH that is relevant to this application was detailed. Finally as this thesis utilises both the concepts of agent-oriented programming and wireless sensor networks, some work where the two areas are combined was introduced. This concept while interesting, is not utilised in this work primarily on the basis of concerns over data admissibility.
Part II

Requirements and Design
Chapter 6

Waste Augmentation and Integrated Shipment Tracking

Chapter 1 introduced us to the information overload that is apparent with the use of wireless sensor networks. This was exemplified by the a vision of the future in which the predictions of researchers and Micro Electro Mechanical Systems (MEMS) manufacturers have been achieved or surpassed, a vision in which sophisticated programs, termed as AIs are required to process the data.

This thesis is not intended to address this issue, however it presents a solution to small portion of the greater problem. The problem, which is detailed in Section 6.1, is that of the monitoring and validation of the transport of waste materials and more generally the creation of intelligent transportation systems for waste management. A potential solution to this problem is introduced in Section 6.1.2.

Following from this, Section 6.2 describes the overall structure of the Waste Augmentation and Shipment Tracking (WAIST) system. The system is described in terms of the three applications that together form the system. These applications are then described in Sections 6.4 to 6.6.

6.1 The Waste Management Problem

Within Ireland and other Organisation for Economic Co-operation and Development (OECD) countries there has been a growing problem of unauthorised waste activity. The Environmental Protection Agency of Ireland (EPA) commissioned a report into this activity [56]. The report identified a number of problem areas of which four are:

- Illegal movement of waste to Northern Ireland;
- Illegal movement of waste to Europe and Beyond;
The illegal disposal of waste by householders has been declining in the period since the reports publication. In 2011 this was 13,700 tonnes reduced from 14,958 tonnes in 2010 and 16,573 tonnes in 2009 [55]. This practice will likely be reduced further by the guilty until proven innocent approach taken in legislation, whereby all householders will be obliged to demonstrate that they are availing of an authorised waste collection service or are otherwise managing their waste in an environmentally acceptable manner [47]. The problem of unauthorised collection and fly-tipping of waste is of particular concern as it has the potential to cause long term pollution and health problems. From a national perspective, it risks compromising the “green” image that is of fundamental importance to Irish agriculture and tourism. The illegal movement of waste, while lacking the same ramifications locally, is as much of a concern because materials may not be treated in an environmentally friendly manner and expenses incurred during their safe disposal are payable by the originating state.

One of the solutions proposed to address these problems was the “development and implementation of a national waste tracking system to allow for the pre-notification and tracking of all waste movements within and outside the state”. Section 2.5 describes a number of systems which have been designed for similar goals, however as mentioned these systems are not designed for the use of governmental agencies as validation tools but by companies to aid in compliance.

The WTF system described in Section 2.5.2 moves partly towards this goal, however it is only applied to the transportation of hazardous waste. The application of the WTF system to all waste would be associated with considerable costs and overhead in terms of manpower, authorising and verifying users of the system, and would even then be vulnerable to exploitation.

6.1.1 Exploitation

There exists a number of scenarios in which the WTF system, which is probably the most cost effective solution, is not fit for purpose. The simplest scenario is when the producer of the waste is also the carrier. As such, the volume of waste can be under-represented and the excess dumped in an illegal manner. On arrival, the correct volume of the waste is delivered to the receiving facility and the costs of the waste producer are reduced.

In a more likely scenario, waste from multiple sources will be collected and transported by a carrier to an intermediate location, perhaps for sorting or consolidation of multiple sources. Each of these individual transports would be verified by the receiving facility (the intermediate location). Once aggregated, the materials are then transported to their final destination and a new WTF form is created. This provides opportunity for a company to take advantage of the first scenario as there is no clear statement that incoming and outgoing waste levels
are compared, nor is it clear that such a system could discriminate between deliberate cost shaving in this manner or general inaccuracies in combined waste volumes.

In the event that such a system was discovered, it would be impossible to determine the resulting location of the illegally disposed waste. Liquid forms of hazardous wastes could contaminate natural resources of an area, such as a water table or reservoir, and cause large scale damage and danger to health without leaving easily identifiable evidence as is the case with solid waste.

6.1.2 Solution

Clearly, the development of a robust tracking system is fundamental to achieving this “national waste tracking system”. Many areas of research described in Chapter 2 can be built upon in the design and implementation of such a system, primarily the technologies described in Sections 2.2 and 2.3 can be applied to the problem.

This thesis presents a possible solution to this problem in the form of an intelligent transportation system for waste tracking known as Waste Augmentation and Integrated Shipment Tracking (WAIST). WAIST embodies the combination of a number of technologies described in Chapters 2 to 5, into a single application designed to enable the real-time monitoring of waste transportation and the validation that the materials have not been tampered with or partially disposed of.

In order to facilitate the design of this “national waste tracking system” a number of functional requirements were defined. These describe the desired functionality of the system. A national waste tracking system should have:

- The ability to monitor and record the location of waste shipments in transit.
- The ability to highlight possible instances of illegal dumping during a shipment.
- The ability to present all captured data in an easily understandable manner.
- The ability to manage the energy resources of the sensors to prolong system lifetime.

Beyond the functional requirements it is also beneficial to define a number of non-functional requirements. These describe criteria that can be used to judge the operation of the system. A national waste tracking system should be:

**Near real-time:** The system should perform logging and processing of data with minimal delay.

**Accurate:** The system should minimise the number of erroneously detected dumping events.

Finally further meta-requirements help inform the manner in which the system is designed and built. A national waste tracking system should be:

**Flexible:** The system should be able to grow or change in response to changing requirements.
**Robust:** The system should be capable of operating for sustained periods of time and react well to unexpected events.

**Intelligent:** The system should utilise and intelligent system for identifying illegal dumping.

### 6.2 Design

The WAIST system is composed of three applications; the in-situ application gathers data from the sensors, the central application processes and stores received data and the visualisation application. Figure 6.1 shows the overall architecture of the system, where the in-situ deployments, represented by trucks, transmit data to the central application which may be visualised in near real-time. With respect to the functional requirements each is achieved through the individual applications. The requirement that the system can monitor and record the location of waste shipments in transit contributes the primary functionality to the in-situ and central applications. The ability to present the data defines the primary functionality of the visualisation application. The requirements for detection and energy management are provided primarily by the central application.

The meta-requirements have a more profound influence on the design of the system. The requirement for flexibility encouraged the adoption of a modular design. In this way the requirement to alter a particular functionality requires the alteration of only a single component of the system. And the replacement of this component should have no material effect on the operation of other components. This also increases the robustness of the system by separating concerns and therefore simplifying the design of individual components.

The following sections will explain these applications, beginning with the *in-situ* application in Section 6.4, the central application in Section 6.5 and finally the visualisation application in Section 6.6. As one of the functional requirements was modularity and flexibility, each
application was designed as a number of components. Each component is implemented as an OSGi [77] bundle and has a specific functionality. Each bundle will be described separately. Before the description of the applications the design of the database is described. This is primarily because the database itself is independent of the individual applications.

6.3 Database

The database is designed to not only record the data sensed but also data computed and events generated by this data. Figure 6.2 shows the full schema of the database. Tables can be separated into three categories, raw data, generated data and organisational.

6.3.1 Raw Data

These tables are designed simply to store the raw data received from the sensors. While it would be possible to have a single table capable of storing all types of data, this would have increased the complexity of data insertion and retrieval. As such each type of data was stored in a separate table.

**Acceleration data:** This is stored in the `accvalues` table. It contains attributes for storing the id of the originating sensor, the time at which it was sensed as well as the x, y and z values sensed for acceleration.

**Light data:** This is stored in the `lightvalues` table. It contains attributes for storing the id of the originating sensor, the time at which it was sensed as well as the intensity of light.

**Power data:** This is stored in the `powervalues` table. It contains attributes for storing the id of the originating sensor, the time at which it was sensed as well as a measure of the current power in the battery of the sensor.

**Location data:** This is stored in the `locationvalues` table. It contains attributes for storing the id of the originating sensor, the time at which it was sensed as pertinent information relating to the location of the sensor. This includes the latitude, longitude, speed and a measure of the accuracy of the reading.

6.3.2 Generated Data

These tables are designed to capture elements of data that are generated based on the raw data. Typical examples would be a features generated for the classifiers, determinations made by the classifiers and events generated by the intelligent components of the system.
**Features:** This is stored in the `features` table. It contains attributes for storing the id of the sensor the data originated from, the time at which the window began, the type of feature set used and up to 30 individual features.

**Class determinations:** This is stored in the `classifications` table. It contains attributes for storing the id of the originating sensor, the time at which the window began, the type of classifier that made the determination, the feature set it used and the determined class.

**Events:** This is stored in the `events` table. It contains attributes for storing the source of the event, the time at which it was generated, the type of event and a message describing the event.

### 6.3.3 Organisational Data

These tables are designed to capture information that is generally used in a descriptive sense. Typical examples would be groupings of sensors into deployments, records of companies being tracked, the sensors available, modalities of these sensors as well as the types of classifiers and feature sets. As these hold little technical merit they will not be explained in detail.

### 6.4 In-situ Application

The main purpose of this application is to gather data from the sensors embedded within the transported waste. Typically the communication range of devices in a wireless sensor network is limited to approximately 100 metres. This requires that the hardware receiving data from these sensors must be secured along with the waste. Though not a vehicular sensor network in the traditional sense, as it has nothing to do with the operation of the vehicle we will refer to the collection of sensing devices and hardware running the application as such.

#### 6.4.1 Hardware

The hardware required to implement this vehicular sensor network was selected based on a number of requirements. Primarily it was required that the hardware could support the SIXTH middleware, which requires the Java runtime environment. Secondly it must be capable of operating on battery power while hidden within a waste consignment and finally it must be capable of interacting with the sensing devices being employed. In order to achieve these objectives, small computing devices referred to as plug computers are used. Figure 6.3 shows an example of such a computer, namely the SheevaPlug. Though not originally designed to be operated using battery power, these devices typically only require a supply of 5V and may consume as little as 5W. The modifications required to convert the devices to use battery power are reported to require only a hobbyist skill level for working with electronics. The
SheevaPlug comes with a distribution of Linux upon which the Java virtual machine can be installed, thus allowing the device to host the in-situ application and device drivers for the wireless sensor network.

Ideally should the WAIST system be utilised on a large scale, a custom platform based on a device such as the SheevaPlug or Raspberry Pi and designed specifically to be powered by battery would be developed. This would allow the encapsulation of all the required components within a single ruggedised form factor. It is quite likely that the environment in which the device operates (concealed within a consignment of waste materials) would require such a protective form factor.

Sensing is achieved through the use of Oracle SunSPOT devices, SunSpots offer the ability to sense temperature, light and acceleration and allow connection of other sensors. The devices execute a version of Java based on the Connected Limited Device Configuration (CLDC), and
can be programmed in an object-oriented manner. Figure 6.4a shows one of the sensing nodes with the sensors exposed.

![SunSpot device and basestation](image)

**Figure 6.4: SunSPOT WNS devices**

In order to communicate with the sensor nodes within the network, a computer must be capable of communicating over the medium used by the devices. This can vary between differing platforms many use the IEEE 802.15.4 standard, often referred to as ZigBee which is based on this standard, others use more common mediums such as Bluetooth or GSM. The SunSPOT devices communicate over IEEE 802.15.4, the attachment of a custom type of node, referred to as a basestation, is required to enable the reception of messages from the network. Figure 6.4b shows the basestation node, it contains the same basic hardware as the sensing node but does not have the attached sensors. Once this is connected to the SheevaPlug, only
the correct device drivers are required to communicate with the wireless sensor network.

The final requirements to be provided for are the ability to sense location and the ability to communicate over long distances. To enable determination of location a number of devices were considered; initially a Mica2 mote with attached GPS sensor was used for this function and the use of a Shimmer mote was also investigated. A smartphone was eventually chosen as the ideal device, as this would provide both location determining abilities and the ability to communicate over 3G/GSM.

### 6.4.2 Software

The application will be described in terms of the bundles contained within. The following bundles are required to deploy the in-situ application;

- SIXTH core
- WAIST core
- Database insert
- SunSPOT adaptor
- rxtx
- Data sender
- In-situ agents
- CArTAgO
- ASTRA
- MySQL connector

![Diagram of application structure and data flow](image)

Figure 6.5: Structure and data flow within in-situ application
Figure 6.5 show the connections between the bundles within the in-situ application, where libraries are shown in orange and WAIST components in green. The SIXTH core bundles provide the functionality that was described in Section 5.5 and the rxtx bundle is a library providing the connection between the adaptor and the SunSPOT basestation. The WAIST core bundle does not provide any functionality to the application, in this and all other uses it is a shared source of classes and constants common to the WAIST system. When one of these classes is changed, the bundle can be updated and all other bundles will have access to the new version of the class (after the framework is refreshed).

**SunSPOT Adaptor**

The SunSPOT adaptor contains all the necessary code for maintaining the wireless sensor network and passing data received into the middleware. Nodes within the network automatically transmit a reading of their remaining power every 5 minutes and light values every second. Acceleration data is not delivered automatically, as the act of sensing and delivering the data consumes too much power. These sensors are activated remotely for short periods of time by the receipt of a particular type of tasking message.

The decision to transmit all data to the in-situ application rather than perform analysis on the sensing device was driven by concerns over the admissibility of evidence gathered by the application without the raw data being available. As such all data is transmitted and recorded at both the local level within the in-situ application and in the central application.

**Data Sender**

This bundle is responsible for forwarding sensed data to the central application. This bundle maintains a socket connection, over which custom serialisable versions of the data objects are transmitted. While the name implies a strictly one way communication, this bundle also handles the receipt of data from the central application. Typically this could be tasking messages for nodes in the network or inter agent communications.

**Database Insert**

This bundle, as the name suggests, is responsible for storing sensed data within a local database. Data is stored locally for a two reasons; firstly it provides a sufficient store of data in case of a failure in the communication link and secondly it permits verification of the consignments integrity on arrival at its destination. This bundle requires the MySQL connector to function, which is a wrapper for the binary MySQL connector for Java.
In-situ Agents

This bundle and the required libraries of CArtAgO and ASTRA are responsible for managing the application. Agents within this bundle can receive data and events from the middleware and make intelligent decisions about the operation of both the application as a whole and the nodes within the sensor network. This topic will be discussed in detail in Chapter 7.

6.4.3 Determining Location

As location determination is being performed by a mobile phone, and this device also serves as the communications link within the vehicular sensor network, this data was gathered and transmitted directly to the central application. For this reason the Android application responsible for the functionality is not considered a component of the in-situ applications but is associated with it.

6.5 Central Application

The central application forms the core of the WAIST system, here data from all individual deployments of the in-situ application is received, stored and analysed. As with the previous section the individual bundles will be described. The following bundles are required to deploy the central application;

- SIXTH core
- WAIST core
- Database insert
- Database access
- Data receiver
- Server agents
- Classifiers
- Live stream
- CArtAgO
- ASTRA
- MySQL connector
- WEKA
- Window builder
Figure 6.6: Structure and data flow within central application

Figure 6.6 show the connections between the bundles within the central application, where again libraries are shown in orange and WAIST components in green. Arrows indicate the direction in which data flows through the application, highlighting unidirectional and bidirectional connections between components. All repeated bundles perform the same functionalities as in the above application, only new bundles will be described in this section.

### 6.5.1 Data Receiver

The data receiver is the mirror to the data sender from the in-situ application, the primary function of this bundle is to receive incoming data from the various deployments. Secondarily it also serves to send a limited amount of data in the opposite direction. Primarily this takes the form of tasking messages and communications between agents.

### 6.5.2 Server Agents

This bundle serves similar purpose to that of the in-situ agents bundle. Primarily intelligent management of the application itself, but also providing a means to make intelligent decisions based on data sensed and the results of analysis. A more detailed explanation of this bundle and examples of the agents are given in Chapters 7 and 9.
6.5.3 Live Stream

This bundle forwards data live, in a filtered manner to the visualisation application. The bundle is idle until a connection is initiated by the visualisation application, which can specify a query to filter the data.

6.5.4 Classifiers

A number of bundles performing classification of the acceleration data received by the server. The functionality of these bundles depends upon the WEKA bundle, built from a binary library, and the Window builder. These bundles will be discussed in greater detail in Chapters 8 and 9.

6.5.5 Database Access

This bundle, as the name suggest provides access to information in the database. Examples of such uses are the loading of training data from which the classifiers are trained and information about the grouping of sensors by the agents. Furthermore the bundle also presents an external interface through which the visualisation application may access historical data. This is achieved through a predefined protocol so as to protect the integrity of the data in the database. As with the live stream bundle the data can be filtered using SIXTH query objects.

6.6 Visualisation Application

Both previous components of the WAIST system perform vital roles, without which the visualisation application can not function. While from a technical standpoint this is true, the reality is that the large amount of data that can be gathered by those components cannot be easily digested and used without this component. Figure 6.7 shows the main window of the visualisation application, with a typical complement of views. As with all other components of the system, the visualisation application is also based on SIXTH and OSGi. In order to facilitate the development of this application, the eclipse platform was utilised, this provides not only a framework for building visual applications but also mechanisms for making these applications easily extensible. Further more the choice was affirmed by the fact that the framework itself is built using OSGi, thus removing compatibility problems between any potential framework and OSGi.

The application is again composed of a number of bundles, however in this case there are a large number of bundles required to support the framework. These bundles are required by the application but only those created for this work are mentioned here;
The application can consist of as little as the visualisation core and location bundle (and required bundles). All other bundles are optional and when omitted the visualisation or other content they provide is not within the application. Figure 6.8 shows the connections between bundles in the visualisation core and where the arrows represent the flow of data.

### 6.6.1 WAIST Visualisation Core

This is the heart of the application, it is responsible for defining the shape of the application. It does not provide any content, but exposes a number of extension points which the other bundles may contribute content to. Figure 6.9 shows the main window of the visualisation application, it is split vertically into three sections, the leftmost is reserved for controls, the
middle for data visualisation and the rightmost for events generated by the central application such as detecting a stationary vehicle or illegal dumping.

Figure 6.9: Empty Application

Figure 6.7 depicts two panels in the control section, one for filtering the streaming of live data and another for managing the playback of data from historical sources. The visualisation panel shows the default map view and the events panel is shown in the right most area. None of the panels are provided by this bundle, but are contributed by their respective bundles.

In addition to providing views to the application, bundles may also provide menu elements. These elements allow the selective use of the views provided by each bundle, giving the user a choice of which of the visualisation methods is preferred.
6.6.2 Location Bundle

The Location bundle is the only mandatory view in the application and by default it cannot be closed. The method of visualisation of location data in this application is formed by recent trends in this regard led by applications such as Google earth and mobile mapping applications which are now ubiquitous within society. As such, development became a question of what data sources or APIs to use.

OpenStreetMap [148] was chosen as the source of map data, this was primarily because it imposes no restriction on the usage of the data. Figure 6.10 shows a simulation of this view, in it there are a large number of vehicles represented in various locations across the map. As each deployment updates its location, the corresponding icon is moved to the new position on the map.

Alternatively, data can be filtered to display a single consignment in transit. Allowing the user to focus on the movements of a specific vehicle. The view is manipulable in the manner that is expected, zooming in providing more detail, however it must be noted that this data is crowd sourced and some areas may lack a high level of detail. To enable the long term analysis of location data, an alternate view is provided which emulates a heatmap of location readings. Figure 6.11 shows an example of this where repeated readings within the same area contribute to large blocks of colour.

6.6.3 Acceleration Bundle

This bundle contributes a number of views to the application. Primarily the contributions are that of visualisation views that use different techniques to present acceleration data in an easily digestible form. Figure 6.12 shows one such view, it uses the instantaneous acceleration values to estimate the roll and pitch of the crates. All containers within in a consignment of
waste that have acceleration sensors attached are visualised simultaneously. This provides an
easy method to identify when a single container is being handled in isolation.

The bundle provides two methods of graphing the data; the first, shown in Figure 6.13,
displays a graph for each acceleration sensor within the particular deployment. Within each
graph all three axes are displayed as a series. The alternate method of graphing displays three
graphs regardless of the number of sensors. Figure 6.14 shows this view, wherein each sensor
appears as a single series in each of the three graphs, grouped by axis. It is important to note
that in order for this alternate graphing method to provide a useful visualisation, either the
orientation of the sensors must be synchronised and maintained or a method of rebranding
these axes is required. For the purpose of this thesis it is assumed that the orientation of the devices are synchronised.

Figure 6.13: Acceleration graph grouped by sensor

Figure 6.14: Acceleration graph grouped by axis
6.6.4 Light Bundle

This bundle contributes two views to the application. The first view is a traditional method of graphing the light levels, shown in Figure 6.15. In this example all individual light sensors within a consignment are each represented as a single series. Figure 6.15 shows the representation of two sensors over a thirty second period, it can be easily seen that the sensors are within a low light area (such as the back of a truck) initially, however towards the end of the period the intensity of light sensed by one of the sensors peaks dramatically. This could be an indication that one container has been removed from the back of a truck into direct sunlight. The second view contributed by the bundle shows a graphical representation of the light levels. Figure 6.16 shows an example of the view. The size and intensity of each representation increase relative to the value of the light readings from a single sensor. In the example, it can be easily seen at a glance that one sensor is producing much higher reading than the other.
6.6.5 Database Bundle

This bundle provides control elements to the application, primarily this concerns the database view shown in Figure 6.17. Additionally functionality is provided allowing the connection to the remote database access bundle, through which data from all previous deployments can be replayed. The speed at which the this information is replayed can be adjusted, allowing the rapid digestion of data. Alternatively the data can be viewed at the speed in which it was sensed, this may be of use for viewing deployments which have had transmission problems during operation but updated all information on arrival from local database stores.

6.6.6 Live Connector Bundle

The live connector bundle provides a similar functionality to the database bundle, however this is applied to instantaneous data. Again there is functionality to communicate with the database access bundle on the server, but in this case it only provides information about the deployments that are currently in operation.

With regard to the filtering of data, the panel provides two options; location information for all deployments with other sensor information for only a selected deployment or only location and sensor information for the selected deployment. Once selected a SIXTH query is constructed to represent the option chosen and transmitted to the live stream bundle on the server. This query is then used as the basis as to whether any piece of data should be delivered to this application.
6.6.7 Event Bundle

This bundle provides for the visualisation of events that are generated by the application. A single view is supplied to the application and by default it displays all events. Figure 6.19 shows an example of the event panel, which is automatically constructed based on the number of events that are permissible within the system.

For each event type a check box is added to the panel, these allow the types of events that are displayed be easily altered. Additionally each event may be associated with a location, when such an event is selected the location at which the event occurred is displayed on the map.

6.7 Discussion

This chapter discussed the problem of waste management in Ireland and a solution posited by the Environmental Protection Agency. Scenarios in which the current practice for tracking hazardous waste could be exploited were outlined, highlighting the fact that even when illegal activity is identified, it cannot be localised. Finally, in Section 6.1.2 it was argued that there was a need for a robust tracking system for waste management and the ideal features of such a system were detailed.
The overall design of the WAIST system was described, followed by the description of the design of the three applications that form the WAIST system.
<table>
<thead>
<tr>
<th>ID</th>
<th>Event Type</th>
<th>Event Time</th>
<th>Has Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>5</td>
<td>Deployment Start...</td>
<td>16:08 19/06/14</td>
<td>false</td>
</tr>
<tr>
<td>5</td>
<td>Deployment Start...</td>
<td>16:08 19/06/14</td>
<td>false</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:08 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Stop for Duration</td>
<td>16:02 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:05 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:05 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:05 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:05 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:05 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:05 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:05 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:05 19/06/14</td>
<td>true</td>
</tr>
<tr>
<td>359188044095653</td>
<td>Momentary Stop</td>
<td>16:04 19/06/14</td>
<td>true</td>
</tr>
</tbody>
</table>

Figure 6.19: Event view
Part III

Implementation and Evaluation
Chapter 7

Facilitating Intelligent Behaviours using Agents

This chapter follows on from the introduction of agent-oriented programming in Chapter 3 and details the design, implementation and evaluation of the use of an agent-oriented programming language within a waste transportation application. The application, namely waste augmentation and integrated shipment tracking (WAIST) was introduced in Chapter 6.

Two objectives prompted this work: the ability to easily apply what might be termed as business logic to wireless sensor network applications and the addition of rational reasoning to a wireless sensor network application.

7.1 Intelligent Agents

Intelligent agents have been used extensively in combination with wireless sensor networks [3, 35, 65, 105, 114, 125, 175, 190, 191, 202]. This thesis introduces a novel method for the use of intelligent agents in combination with wireless sensor networks. Within this work, agents are facilitated through the use of an environment interface layer within WAIST.

This method of facilitating intelligence within the application does not situate agents inside the wireless sensor network. Situating agents within a wireless sensor network is discussed briefly in Section 5.6. This choice not to do so was motivated by a number of factors. Within this application, concerns over admissibility of data forced the transmission of all readings for storage, as such no energy saving can be made based on the use of agents. Additionally the use of in-network intelligent agents would likely tie the application to a particular type of sensor or sensor operating system as agent languages are typically compatible with only a single type of device.
7.1.1 Agent Programming Language

Emanating from the consideration of a number of agent programming languages presented in Chapter 7, the ASTRA agent programming language was selected for use in this work. ASTRA was chosen because it represents a new breed of agent programming languages that provides minimal runtime mechanisms such as debuggers and can be easily integrated with other technologies, such as OSGi. ASTRA also has a number of other features, including static typing, support for multiple inheritance, and language level integration with the CArtAgO and EIS interfaces discussed in Section 3.4.

Furthermore of the two environmental abstractions studied, CArtAgO was also chosen for use in this work. The use of the CArtAgO framework permits intelligent agents from a number of differing agent programming languages to be used. Consequently to adopt a plug and play metaphor it is possible to replace ASTRA with another agent-oriented programming language if deemed appropriate.

The following sections will describe first the general usage of ASTRA and CArtAgO in conjunction. Following from this the design and implementation of the integration of these technologies with SIXTH and WAIST will be detailed. Finally the evaluation of the realised system will be discussed.

7.2 Environment Abstraction

Section 3.4 detailed two alternate technologies for integrating agent oriented programming languages with their environment. This chapter describes the integration of the chosen technology CArtAgO [165] with the WAIST application.

CArtAgO has a number of means of supplying information to those agents that wish to receive it. The concept of focus is used to differentiate between an agent that wishes to utilise the functionality of an artifact and an agent that wishes to receive all the percepts made available by an artifact. An agent wishing to receive information from an artifact must first focus on it, and before that it must attain a reference to the artifact. An introduction to ASTRA can be found in Section 3.5.6. Figure 7.1 shows the code required to achieve the above steps when using ASTRA.

```java
try{
    CARTAGO.makeArtifact('BundleMan', 'ie.ucd.sixth.cartago.BundleArtifact');
} recover { console.println('The artifact bundleMan is already made');}
CARTAGO.lookupArtifact('bundleMan', object<ArtifactId> bundles);
CARTAGO.focus(bundles);
```

Figure 7.1: Example of artifact creation and focusing within ASTRA

Line 2 creates an instance of the artifact, this is contained within a try block, to recover from
an error generated by the artifact being present already. The artifact is associated with a unique name and is identified by its Java package and class name (shown on line 2). Line 4 shows the lookup of the artifact based on the name defined during creation. This associates the actual artifact with the variable called “bundles”. Finally, line 5 registers the interest of the agent in the percepts generated by the artifact using the “focus” operation.

Once an agent has focused on a bundle, it can access data within that bundle. This data is stored in what are called observable properties. In ASTRA, a distinction is made between beliefs of the agent and observable properties of an artifact. Figure 7.2 shows two examples of the usage of an observable property called count that contains a single integer parameter. Both examples take the form of an if statement where the following line is executed if a binding is found for the int variable. In the first example the search is undirected, which means that variable can be bound by any artifact that has a matching observable property. In the second example the search is restricted to only the artifact named “myCounter”.

```java
1 CARTAGO.lookupArtifact("myCounter", object<ArtifactId> myCounter);
2 CARTAGO.focus(myCounter);
3
4 if ( CARTAGO->count(int X) )
5   System.println("undirected from property: " + X);
6
7 if ( CARTAGO("myCounter")->count(int Y) )
8   System.println("from property by name: " + Y);
```

Figure 7.2: Observable property usage in ASTRA

Alternatively, an artifact may make information available in a form closer to that of an event. To this end the concept of a signal is introduced. Within ASTRA, a signal is interpreted as an event, similar to the treatment of observable properties a distinction is made between signals from CArtAgO and events from within the agent. Figure 7.3 shows an example of an ASTRA rule for responding to a CArtAgO signal. In this case the signal is a predicate containing location information and only the most recent for each sensor identifier is maintained within the belief base of the agent.

```java
1 rule @cartago("signal", location(long id, long t, double la, double lo, float v)) {
2   if (loc(id, long t0, double la0, double lo0, float v0)) {
3     if (t0 < t) {
4       -loc(id, t0, la0, lo0, v0);
5     } else {  
6       +loc(id, t, la, lo, v);
7     }
8   }
9 }
```

Figure 7.3: Signal interception in ASTRA
7.3 Integration with SIXTH

The integration of CArtAgO with the SIXTH middleware takes the form of a single bundle. This bundle contains no dependency or requirement on either the agent programming language in use or any technology it is based on. This bundle provides a number of artifacts that can be used not only to obtain data from the wireless sensor network, but also to perform operations within the wireless sensor network and within the application. In order to facilitate the extension of the basic artifacts with application-specific ones, the bundle has no more functionality than exporting the classes that are defined.

The following artifacts are available by default within SIXTH; AdaptorArtifact, BundleArtifact, DataStreamArtifact, RetaskerArtifact and LocalBundleSourceArtifact. These will be detailed in the coming sections.

7.3.1 AdaptorArtifact

The AdaptorArtifact serves to provide information relating to the Adaptors within SIXTH and their state. This information can allow an agent to infer what resources are available for use and determine the best course of action in a given situation. A simple example might be the installation and initiation of a bundle when a required service is lacking. The information supplied to the agent by this artifact is equivalent to the information given to an adaptor receiver discussed in Section 5.5.1.

This information is provided as a signal, which is a predicate with the name `adaptorEvent` and contains two parameters. The signal and it’s parameters are described below.

- `adaptorEvent(object(AdaptorDescription) ad, string type)`
  - `AdaptorDescription ad` An `AdaptorDescription` object that contains information such as the name, state or type of the adaptor.
  - `string type` A String that describes the type of the event.

7.3.2 BundleArtifact

The BundleArtifact provides a mechanism by which to inform an agent of the state of the OSGi deployment and also to enable the agent to effect change within this deployment. Such functionalities are less specific to the SIXTH middleware than the other artifacts as they are concerned only with the underlying OSGi Framework. The artifact provides information on the state of all bundles installed in the deployment as well as operations allowing the installation, starting, stopping and removal of bundles.

This information is provided as a signal, which is a predicate with the name `bundle`. The signal and it’s parameters are described below.
• bundle(string sname, object(Bundle) b, string version, string state)
  
  – string sname  The symbolic name of the bundle
  
  – Bundle b  The bundle object, this is required to perform any operations such as start and stop
  
  – string version  The version of the bundle represented as a string, this will traditionally be of the form “0.3” but may also be in a longer version “2.0.3.v20140305”
  
  – string state  The state of the bundle when the signal was sent. This can be any one of the following “uninstalled”, “installed”, “resolved”, “starting”, “stopping” or “active”.

This signal has two possible sources; the first is as a direct result of the getBundles operation at which point a signal will be sent for every bundle within the system. If the operation is repeated you will receive the same signals. The second source is a notification after a bundle has changed state, these may occur when any of the operations such as startBundle, stopBundle or uninstallBundle are used.

### 7.3.3 DataStreamArtifact

This artifact provides access to the stream of data passing through a given wireless sensor network application. As data types and modalities can be varied, the data is passed as an ISensorData object. This means that the contents must be accessed in order to reason about and make decisions based upon it. The ISensorData object is stored within an observable property named `data`. Every time that a new piece of data is received, the content of the object is updated. This process generates an event which can be used to trigger a rule.

### 7.3.4 LocalBundleSourceArtifact

The LocalBundleSourceArtifact artifact provides a mechanism for the discovery of bundles from a local source such as a directory on the local machine. A single operation is provided named `findBundles`, which requires a parameter of a location encoded as a string. All jar files present in the location are assumed to be bundles and their information is extracted from the file name and passed to the agent. This is done using a signal with the name `jar`. The signal and its parameters are described below.

• jar(string fname, string sname, string version)
  
  – string fname  The file name and location of the jar file
  
  – string sname  The symbolic name of the bundle
  
  – string version  The version of the bundle represented as a string

83
7.3.5 RetaskerArtifact

The RetaskerArtifact is the primary means for actuation within a wireless sensor network, which was discussed in Section 5.5. The artifact does not provide any percepts to the agent and provides only a single operation. This operation, named retask, allows agents to reconfigure sensors by means of tasking messages. These can be used for example to adjust the sampling frequency of a sensor or a group of sensors.

7.4 Extending the Environment

Expanding the functionality of the agents to receive application specific information and perform operations on the application is done by adding more artifacts. Figure 7.4 shows how the agents interact with both the application and the SIXTH middleware through the use of the Artifacts provided. The design of this system and the manner in which it may be extended are a consequence of the operation of classloaders within OSGi.

In this design, the same Classloader must be used to load the agents and the artifacts. The natural solution was to use the classloader of the bundle containing the agents and pass it as a parameter to both ASTRA and CArtAgO.

Extensions to the capabilities of the agents are achieved through the addition of new artifacts. Two approaches are available when considering the location of such artifacts. In the first approach we can group all artifacts within a single bundle. This allows the simplest use because the agent bundle requires only one extra dependency. The alternative approach is to
situate each artifact with the code that it relates to. In this case an artifact responsible for accessing data for the database, would be situated within the database access bundle. This approach complicates the usage of artifacts, as each bundle/package will have to be imported by the agents bundle separately. The main consequence of choosing the first approach is that there is a requirement for all bundles to be loaded by the intermediate CArtAgO bundle. This introduces a problem in that if we do not wish to use a bundle, we must still load it.

The approach taken in this work was to situate Artifacts within the bundle to which they most closely relate. The following artifacts are made available within the bundles of WAIST; ClassifierArtifact, DeploymentArtifact, CommunicatorArtifact, RemoteWaistReposArtifact and 4 data type specific artifacts for GPS, Light, power and event data. These will be detailed in the coming sections.

### 7.4.1 ClassifierArtifact

The classifier is designed to inform agents about the results generated by the classifiers within WAIST. These topics are discussed in greater detail in Chapter 8. The ClassifierArtifact has two main functions, firstly it delivers percepts to the agent for every class determination that is made and secondly it allows actuation into the application in a number of ways. The delivery of percepts is achieved through a signal named `classified` described below:

- `classified(long id, long timestamp, int class, int ctype, int fset)`
  - `long id` - The identifier of the sensor from which the readings originated.
  - `long timestamp` - The timestamp indicating the moment at which the window of data commenced.
  - `int class` - An integer representation of the determined class.
  - `int ctype` - An integer denoting the type of classifier the determination was made by.
  - `int fset` - An integer denoting the feature set that was used in the determination.

This signal is designed to allow the agents to reason about differing determinations made by different classifier and feature set combinations. Figure 7.5 shows excerpts taken from an agent reasoning about classifier determinations. Determinations are added to the belief base of the agent in the first rule. When the number of determinations that exist for a given sensor id and time period matches the number of classifiers, the agent adopts a goal of determining which of the class determinations is correct (line 12).

A second function of the artifact is that of actuation within the application. Primarily this relates to the generation of events, implementations of the ISensorData interface described previously, which are passed back into the application for recording and visualisation. The artifact also contains an operation for recreating windowed data for a specific period, and
passing it back into the application to allow determination by another classifier which could be installed and started by an agent using the BundleArtifact discussed above.

### 7.4.2 DeploymentArtifact

This artifact serves to provide information about the in-situ deployments in operation and the sensors that are contained within them. Information is returned as signals in response to operations. Primary among these is the signal `sensor` with the following parameters:

- **sensor(long id, int group)**
  - `long id` - The identifier of a sensor.
  - `int group` - An integer denoting the in-situ application that this sensor is associated with.

The capabilities of the sensors can be returned using the `modality` signal, each contains only the following two parameters:

- **modality(long id, string mod)**
  - `long id` - The identifier of the sensor.
  - `string mod` - A string representation of a modality that this sensor can perceive.

In order to fully specify the capabilities of a given sensor, a number of these signals are required. Figure 7.6 shows the signals required to describe the functionality of one of the SunSPOT sensors discussed in Section 6.4.2. The signals show the description of a single SunSPOT with the identifier 1234556. This information is designed to allow agents reason about the control of such sensors and their activation.

---

**Figure 7.5: Excerpts from a classifier agent**

```java
rule @cartago("signal", classified(long id, long time, int cl0, int cf0, int ft0)) {
  +class(id, time, cl0, cf0, ft0);
}

rule +class(long id, long time, int cl0, int ct0, int ft0): numSensors(int n){
  int i = 0;
  synchronized (t) {
    foreach(class(id, time, int c1l, int ct1, int ft1)){
      i= i + 1;
    }
    if(i == n & ~agreement(id, time, int cl2) & ~disagreement(id, time)){
      !getAgreement(id, time);
    }
  }
}
```
A final capability of the artifact is the creation of the TaskingMessage objects that are required to retask sensors within the in-situ deployments. Figure 7.7 shows an excerpt from an agent that is managing sensors. The triggering event of this rule is the addition of the goal deploy, implying that the deployment should be started. The context of the event contains the id of the deployment. If the deployment has not already been started, the agent iterates through the attached sensors and creates a tasking message for this sensor which is then sent using the RetaskerArtifact.

These actions are performed in order to reduce the power usage of the acceleration sensors, having them only sense while the vehicle they are placed within is stationary.

### 7.4.3 CommunicatorArtifact

The ability of intelligent agents to interact socially is widely acknowledged as a key feature of multi-agent systems [179, 199]. As such the majority of agent-oriented programming languages and toolkits incorporate some communication functionality. To facilitate this inter-agent communication, great efforts have been made to standardise Agent Communication Languages (ACLs): most notably the ACL developed by the Foundation for Intelligent and Physical Agents (FIPA)[68].

To enable any agent programming language function within WAIST, it was necessary to include a means for agent communication. This may not be strictly necessary when all agents...
are on the same platform, as many agent programming languages include the functionality of an agent communication language, however when communication is required with an agent in a remote deployment delivery of messages may be difficult if not impossible.

Based on the principles of speech act theory [107], a very simple mechanism for communication is provided by the CommunicatorArtifact. Operations are provided that allow request and inform type messages to be sent. Limiting the number of performatives in this way is designed to simplify the communication process as has previously been done in [87]. The messages are then converted into objects that implement the SIXTH ISensorData interface and can be distributed using the SIXTH middleware.

On arrival the messages are delivered to the agents using signals named req and inf which contain the name of the sender followed by a variable number of parameters defining the contents of the message.

### 7.4.4 RemoteWaistReposArtifact

This artifact is similar to the LocalBundleSourceArtifact from SIXTH described in Section 7.3. This however is designed to discover the bundles available on a web based location. The location that is connected to is simply a folder containing all the bundles and a PHP file that lists the file named separated by line breaks. Figure 7.8 shows a rule taken from an agent responsible for managing a deployment. Every 5 minutes the agent queries a local folder and remote server for bundles. This artifact returns the same signal as the LocalBundleSourceArtifact with the absolute path for the file replaced by the URL of the remote file. This can be used to update the bundles within a deployment, but more importantly allow the agent to reason about when this should be done.

```
rule +!checkForUpdates() {
    while (true) {
        system.sleep(300000);
        if (remoteHost(string url)) {
            CARTAGO.getBundlesOnHost(url);
        }
        if (bundleLocation(string l)) {
            CARTAGO.findBundles(l);
        }
    }
}
```

Figure 7.8: Excerpt from agent checking for new bundles

### 7.4.5 GPSDataArtifact, LightDataArtifact, PowerDataArtifact and EventDataArtifact

These four artifacts all serve the same purpose, the delivery of data in a format that is easier for an agent to reason about. Each artifact is responsible for a single type of data, e.g. the
GPSDataArtifact only sends location data. All data is provided using the signal mechanisms, GPS data is forwarded using signal named location with the following parameters:

- location(long id, long timestamp, double lat, double lon, float sp)
  
  - long id - The identifier of the GPS sensor.
  
  - long timestamp - The timestamp indicating the moment at which the reading was taken.
  
  - double lat - A double precision floating point representation of the latitude reading.
  
  - double lon - A double precision floating point representation of the longitude reading.
  
  - float sp - A single precision floating point representation of the current speed reading.

The signals representing the power levels of the sensors and the light levels they read are forwarded using signals named power and light respectively and contain the same parameters:

- power(long id, double val, long timestamp)

- light(long id, double val, long timestamp)
  
  - long id - The identifier of the sensor.
  
  - double val - A double precision floating point representation of the reading (power or luminosity).
  
  - long timestamp - The timestamp indicating the moment at which the reading was taken.

The addition of new sensors and sensing modalities requires the addition of new artifacts if agents are to reason about such readings easily. The traditional DataStreamArtifact can be used with any data that does not have an associated artifact, however accessing the data for usage requires the use of further Java libraries to extract the required data from the ISensorData object.

### 7.5 Evaluation

In order to demonstrate the effectiveness and usefulness of the integration of ASTRA and SIXTH, this section describes a user trial that was conducted using students from the MSc in Advanced Software Engineering programme in University College Dublin. Twenty participants were asked to solve two problems relating to the use of wireless sensor networks and the SIXTH middleware. Half of the students attempted the first problem using ASTRA and the
other half using Java. This was then reversed for the second problem such that all students had attempted one problem in each language.

The solutions delivered were analysed using both objective numeric metrics and subjective analysis. The aim of this work was to reduce the required effort when implementing intelligent behaviours in a complex system. Naturally it can be assumed that a programmer sufficiently competent with the technologies in use could implement the behaviours using those technologies.

However, the intention of this work is to show that programmers with a lesser understanding of tools such as SIXTH and OSGi benefit from the higher level abstractions provided through CArtAgO. This study was intended to ascertain whether the use of the system reduced the effort required by a developer to implement a solution.

7.5.1 Background

Evaluating programming languages, methodologies, paradigms and toolkits is a matter of some discussion and debate within the wider software engineering community. For systems like this, the principal aim is to make it easier for developers to perform particular tasks. Specifically, it should facilitate developers in implementing reliable, predictable, understandable and manageable wireless sensor network applications.

Commonly, the notion of “programmer effort” is used in attempting to quantify the amount of work a programmer must undertake to complete some programming task. A number of studies have used objective measures to quantify programmer effort. Although not specifically based in the agent domain, these metrics can help inform a choice of metric for agent-oriented programming and have previously been used as such [112]. Typical examples of these metrics include the number of non-comment, non-empty lines of code [119] or non-commented code statements [192]. Another common measurement of effort is the time taken to implement a solution to a standard problem [90, 119].

When performing an evaluation such as this, a common approach is to use two groups of participants. Each group is presented with a common problem to solve, with all factors other than the subject of the evaluation being kept equal [90, 119, 192].

7.5.2 Problems

Two problems were developed, each containing a number of distinct tasks. The first problem was based on analysing incoming GPS readings and performing actions and the second was based on managing an OGSi application. This section will describe both of these problems, however for further detail the full problem specifications have been included as Appendices A and B. These include the materials describing how the same precesses can be achieved using Java as were described in the preceding sections.
The design of the problems as a number of small tasks allows the use of a more concrete metric than those based on lines of code. The solutions can be analysed in terms of the number of tasks completed, the principle is the same as in [90, 119] where time taken is used. However, in this case the number of task completed within a fixed time period is used rather than time taken to complete. This technique has previously been used in studies within the field of agent-oriented programming [112]. Students had one hour to complete each of the problems.

**Problem 1**

This problem was designed such that the students would undertake a number of tasks that would conceivably be a part of the WAIST application. While it was not strictly necessary to solve all problems in the order provided, most tasks have a reliance on the previous task. Based on the concept of GPS and acceleration sensors within a number of trucks, the students were required to complete the following tasks:

1. Access the sensor information;
2. Maintain the most recently recorded position of each truck;
3. Notify the system whenever the speed of a truck falls below a threshold value (1 m/s);
4. While the speed remains below the threshold it is considered stationary. When the speed moves above this threshold the system should be notified that the truck has resumed driving;
5. When a truck is first detected as stationary, all acceleration sensors within that truck should be activated;

**Problem 2**

This problem was designed such that the students would undertake a number of tasks based on the management of an OSGi application. Based on this premise, the students were required to carry out the following tasks:

1. Access the bundle information
2. Maintain a record of all bundles within the system
3. Query a local source of bundles and if any of the bundles are required within the system and are not already installed (list was supplied) they were to be installed.
4. Whenever a bundle is installed, if it is on the required list it should be started.
5. If a bundle from the local source is a new version of an existing installed bundle, the older bundle is to be stopped and uninstalled, then the new bundle should be installed and started.
The students were to attempt both problems, one using ASTRA and the other using Java. To prevent any bias towards the first language used, half the class attempted the first problem in ASTRA and the other half in Java. This was then reversed for the second problem. The students had one hour to complete each problem after which they were required to submit the work they had completed.

### 7.5.3 Results

The source code of the solutions was analysed in order to get a measure of the number of tasks completed within the allotted time. Table 7.1 shows the average number of tasks completed for each problem and language combination. The class consisted of 20 students studying Agent-Oriented Software Engineering as part of an MSc in Advanced Software Engineering. The programme is completed part time over two years and each module takes place over an intense one week period of classes. The evaluation took place on a Friday afternoon and was the last commitment of the students. As such, a number of students did not stay to complete the second problem, which was perceived as the more difficult problem.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Language</th>
<th>Number of submissions</th>
<th>Number of tasks</th>
<th>Average tasks solved</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ASTRA</td>
<td>11</td>
<td>5</td>
<td>2.63</td>
<td>1.15</td>
</tr>
<tr>
<td>1</td>
<td>Java</td>
<td>9</td>
<td>5</td>
<td>0.67</td>
<td>0.47</td>
</tr>
<tr>
<td>2</td>
<td>ASTRA</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>0.70</td>
</tr>
<tr>
<td>2</td>
<td>Java</td>
<td>7</td>
<td>5</td>
<td>0.86</td>
<td>1.72</td>
</tr>
</tbody>
</table>

Table 7.1 displays an obvious improvement in the number of tasks completed when using ASTRA over the number of tasks completed using Java particularly for the first problem. The improvement in the task completion within the second problem is not as pronounced but is nevertheless still apparent.

A unpaired student t test [206] was performed on the above results comparing the number of tasks completed over the same problem. For the first problem the number of tasks completed was significantly higher and yields a p value of 0.0002. When analysing the second problem in the same manner the slight difference in the number of tasks complete yields a p value of 0.89.

### 7.5.4 Survey

In order to attain a subjective analysis of the programmer effort involved in the completion of the problems, the students were asked to respond to a survey. Students were asked to indicate their level of agreement with statements relating to their experience level with a number of technologies and their experiences with the problems they completed. The results
were captured on a 5 point Likert scale. Appendix C includes the full detail of the questions asked in the survey. The average number of years experience in industry of the students was reported as $7.4 \pm 5.1$ years. The following statements relating to the problems were made:

Q1 The supporting information for solving the problem in ASTRA was adequate.

Q2 The supporting information for solving the problem in Java was adequate.

Q3 Using ASTRA made the problems easier to complete.

Q4 ASTRA was easier to use than Java.

Q5 ASTRA was easier to modify than Java.

Q6 My ASTRA solution was more readable than my Java solution.

The responses to the questions are contained in Table 7.2 where the percentage is given for each of the responses and Figure 7.9, which show a graphical representation of the same data. The average and standard deviation of the responses is calculated, where strongly disagree is 1, agree is 2, neither agree or disagree was 3, agree was 4 and strongly agree is 5. The responses for a number of statements were quite positive, primarily questions 3, 4 & 6 showing that students believe that the problems were easier to complete and code was easier to use and more readable when using ASTRA.

<table>
<thead>
<tr>
<th>Question</th>
<th>Strongly disagree</th>
<th>Disagree</th>
<th>Neither agree or disagree</th>
<th>Agree</th>
<th>Strongly agree</th>
<th>Mean</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>35</td>
<td>20</td>
<td>45</td>
<td>0</td>
<td>3.1</td>
<td>0.89</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>30</td>
<td>20</td>
<td>40</td>
<td>0</td>
<td>2.9</td>
<td>1.04</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>10</td>
<td>30</td>
<td>45</td>
<td>15</td>
<td>3.65</td>
<td>0.85</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>30</td>
<td>30</td>
<td>45</td>
<td>10</td>
<td>3.2</td>
<td>0.98</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>35</td>
<td>45</td>
<td>45</td>
<td>0</td>
<td>2.85</td>
<td>0.73</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>20</td>
<td>30</td>
<td>30</td>
<td>15</td>
<td>3.45</td>
<td>0.97</td>
</tr>
</tbody>
</table>

### 7.5.5 Summary

A number of methods were utilised to evaluate the efficacy of this work as a means to simplify the implementation of intelligent behaviours. These were based on a class of masters students studying Agent-Oriented Software Engineering as part of an MSc in Advanced Software Engineering.

The first method measured the number of tasks completed by the students within a fixed amount of time. The results of the two problems were detailed, and show that students completed a greater number of tasks when using ASTRA compared to the same tasks completed in Java. In the first problem the improvement is quite significant where on average 2 more tasks were completed. In the second problem this was not significant, however this may be
due to the poor completion rate in the second problem. This difference could be ascribed to 
a perceived difference in adequacy of the supporting information for completing the problem 
in Java.

The subjective analysis carried out re-enforces the objective assessment. The results show 
that there was a perception amongst the students, that using ASTRA rather than Java made 
the problems easier to complete. When surveyed, 85% of the students indicated that they 
had no experience using OSGi, this together with the positive feedback about using ASTRA 
supports the theory posited in Section 7.5. There is also positive responses for the ease of use 
and readability of ASTRA code when compared with Java.

7.6 Discussion

This chapter discussed the integration of an agent programming language with the SIXTH 
middleware and WAIST system as a means to reduce programmer effort when developing 
behaviours. The implementation of this integration was detailed and an analysis was carried 
out. This analysis was carried out using a user trial and was based on objective and subjective 
measures.

The results indicate that objectively, students completed on average 1.45 more tasks when 
using ASTRA than Java and additionally there was a perception that the problems were easier 
to complete when using ASTRA. It should be pointed out that due to the limited sample size 
it may not be possible to generalise these findings.
Chapter 8

Classification of Object State

This chapter details the design and implementation of a system for determining the state of objects within a transportation context using data received from acceleration sensors. As discussed in Chapter 4, this area has received little attention within the research community. It was required that a study be performed in order to determine the applicability of techniques that have proven useful in similar classification problems. This chapter discusses the completion of this study and the results achieved.

8.1 Method

8.1.1 Data Collection

Accelerometer data was collected using Sun SPOT (rev 8) devices developed by Sun/Oracle. Each device contains a MMA7455L triaxial accelerometer sampling with a range of $\pm 8g$, which was sampled with a 10-bit resolution. These devices are capable of transmitting sensed data wirelessly over a short range using 802.15.4 low rate wireless personal area network.

A custom application was developed for the Sun SPOT device to simplify the capture of training data, this application was controlled using two buttons. The first button initiated a configurable fixed time period of sampling with visual and auditory indicators announcing that sampling was occurring and when the period was complete and the second button instructed the device to attempt to transmit the readings to the computer. Readings were not removed from the device until they had been successfully received, this configuration facilitated the capture of data without the need for a constant connection to a computer to store the data. This allowed the training data to be captured without the need for a computer in the vicinity to record it, as data could be transferred at a later point.

Before each set of readings was gathered, the physical device was synchronized with the clock of a laptop computer, this was repeated after the experiment to ensure that the unit remained
accurate. As the samples were taken over short periods of time the difference was never greater than 0.004 seconds.

Little is known about the minimum sampling frequency required to attain good accuracy from a classifier, as such previous studies aiming to utilize accelerometer data have chosen high rates such as 300 - 380 Hz [58, 132]. Another study expressed the belief that sampling rates in excess of 2500 Hz were required to accurately monitor container integrity, however the system introduced itself was only capable of sampling at 233 Hz [176]. With respect to this 250 Hz was chosen for this study, as this is the maximum achievable using the above technology, this frequency should be above what is required to assess object state and is significantly above the 20 Hz sampling posited as the minimum required for classification of human activity[20].

For the data collection two types of container were used, a 60 litre rectangular tight head plastic drum and a 120 litre circular open top plastic drum, both of which are UN approved for Dangerous Goods. These specific volumes were chosen as they would commonly be used in the transportation of liquid wastes, are both manipulable by a single person and possibly provide disparate readings through some actions such as rolling and falling.

The sensors were attached to the top of each of the containers, on the 60 L in a fixed position and orientation and in the centre of the 120 L container. Figure 8.1a shows the positioning of the sensor on the 60 L drum, through all testing and generating data the orientation of the sensor with respect to the container remained constant. Figure 8.1b shows the positioning of the sensor on the 120 L container, this was positioned as close to the centre as possible and the orientations of the sensor were annotated in order to maintain the ability to mimic the experiments carried out with the 60 L container. The sensors were firmly affixed using VELCRO® Heavy Duty strips which keep the sensors attached through all motion in the
generation of the sample data but allow removal of the sensors for reprogramming or charging.

### 8.1.2 Data Preprocessing

As noted in Section 5.2, one of the greatest issues associated with wireless sensor networks is the lack of power resources. In order to prolong the lifetime of the sensors transmitting the data, a number of techniques were investigated that would reduce the amount of data transmitted wirelessly and consequently reduce power usage.

The first technique, which was discussed in Section 4.2.2, reduces the amount of data to be transmitted by up to one third. This is achieved by removing directionality from the sensor data. The second technique, which was discussed in Section 4.2.3, is the reduction of the sampling rate. This was studied to help determine a lower bound for frequency and also as a potential measure to reduce power consumption.

Table 8.1 shows the frequencies that were evaluated along with the number of readings that were in each window of data and the number of readings that were omitted between each consecutive reading. This study was carried out using the best performing feature set from the initial evaluation, namely the mean, standard deviation and quartiles. The signal combination technique discussed in Section 4.2.2 was not used.

<table>
<thead>
<tr>
<th>Frequency (hz)</th>
<th>Readings per window</th>
<th>Gap between Readings</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>500</td>
<td>0</td>
</tr>
<tr>
<td>125</td>
<td>250</td>
<td>1</td>
</tr>
<tr>
<td>50</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>25</td>
<td>50</td>
<td>9</td>
</tr>
<tr>
<td>12.5</td>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>40</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>49</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>61</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>82</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>125</td>
</tr>
</tbody>
</table>

### 8.1.3 Windows

An initial pilot study was carried out using only three states, motionless, in transit and falling. This pilot was based around the concept of a small-scale research platform [106] where a remote control truck was used to simulate the in transit state and other states were gathered as normal. Figure 8.2 shows the truck containing two sunspot sensors, which was driven around the office to simulate containers in transit.
After viewing the data generated for the falling state, a window of two seconds was chosen from which to extract features. Two seconds was chosen as this represented approximately twice the duration of a typical fall. Figure 8.3 shows a plot of the data recorded during a typical fall. The period in which there is activity makes up approximately one second. This coupled with a 50% overlap between windows maximized the extraction of data from each event. As discussed in Section 4.2.1, this configuration of window overlap is quite common in human activity classification.
8.1.4 States

Following from the pilot study, a larger study was performed in which containers were manipulated through eight different states (motionless, in transit, falling, tilting, dragging, lifting, carrying and rolling). The description of each of the classes are given below with the details of how the data was captured. To ensure that there was sufficient data to address the research question, the experiments were repeated four times, once with each container while both empty and while containing 30 litres of water. Each of the differing states was captured over differing periods and repeated a numbers of times in order to attain approximately the same number of windows of data.

Motionless: The motionless state was measured in 5 different positions, upright as well as laying on each of the containers four sides. The materials in the container were allowed to settle for 60 seconds before the measurements were taken to prevent the internal movement of the contents from distorting the acceleration values recorded. Data was recorded in 5 second increments which were repeated 5 times in each of the positions mentioned above.

Falling: This state is defined as the toppling over of the container onto one of it’s sides where it comes to rest. Data was recorded in which the containers were gently pushed until they reached the point at which they fell over. Data was gathered over three second periods giving a view of each of the measurements from differing temporal positions. The process was carried out equally falling onto each of the sides of the containers 13 times.

Tilting: This state is defined as the motion of gently tilting the container from an upright position through a minimum of 45 degrees where it was then held. Data was gathered over three second periods equally spread between tilting each of the four sides of the container towards the ground both 45 degrees and 90 degrees 7 times each.

In Transit: This state is defined as the container being secured in a stationary position within a moving vehicle, measurements were taken with the container equally placed upright as well as resting on each of its sides. The container was placed in a position such that it could not move or fall over. Data was captured over six second periods and repeated 4 times for each position (upright and laying on each side). Data was captured while the container was within the passenger compartment of a mid-sized (D-segment) vehicle, the vehicle was then driven through an area containing inclines, left and right turns and was driven at speeds up to 40 km per hour.

Dragging: This state is described as the action of moving the container in a horizontal direction while remaining in contact with the ground. This simulates the movement of loads too heavy to lift. Data was recorded over an eleven second period, and were repeated twice while the container was upright as well as on each of the containers sides.

Lifting: This state describes the elevation of the container by approximately 1 metre and
its positioning on a solid surface directly in front of it. The data was recorded in two second periods, each one the length of a single window, and was repeated twenty times while upright and while lying on each of the containers four sides.

**Carrying:** This state is the movement of the container by a person during which the container’s weight is wholly borne by that person. Measurements were taken in eleven second periods during which the container was carried forwards at normal walking pace. The process was carried out twice while positioned upright as well as on each of the containers sides.

**Rolling:** This state is the process of moving the container by rolling along the ground without lifting or dragging. This was carried out with both containers to generate data with both smooth rotation, generated by the almost cylindrical 120 L container, and violent rotation generated by the continual impacts from the rectangular 60 L container. Data was gathered in eleven second periods which were carried out five times both clockwise and anti-clockwise.

### 8.2 Features

Differing approaches have been used to extract features from time series in studies of human activity classification. Most commonly used are time domain features or features calculated from a discrete wavelet transform (DWT) or discrete Fourier transform (DFT) performed on a window of data, which were introduced in Sections 4.3.1 to 4.3.3. A selection of methods from each of the above techniques are utilised in this study. This section introduces the feature extraction techniques used from each of the above domains starting first with features extracted from the DWT of the windows, then the features from the DFT of the windows and finally the features extracted from the time-domain signal. Each set of feature sets discussed is labelled with a number to enable easy identification.

#### 8.2.1 Wavelet Features

With this approach, the original time-domain signal (maximum frequency $f$) is initially decomposed into detail and coarse approximation information by high-pass filtering ($[f/2, f]$) and low-pass filtering ($[0, f/2]$) respectively [123]. This process can be repeated through subsequent levels of decomposition during which the approximation signal from the previous level is split into a second approximation and a detail coefficient. This process is repeated to the desired decomposition level. Features were then extracted from the detail coefficients at differing levels, each referred to as $cD_n$ where $n$ is the level used.

Based upon previous studies [157, 186], three sets of wavelet features were selected, these feature sets were then used in two ways. First the features were extracted from each of the three acceleration signals individually and used in the order X, Y, Z and secondly a
single signal was calculated using the root-sum-of-squares method described in Section 4.2.2. Features were then extracted using the same techniques based on the combined magnitude signal. As such if an extraction technique produces three features, the first method of analysis produces 9 features for classification and the second only three.

Table 8.2 details the wavelet mother used during the decomposition process, the number of features returned from each signal decomposed and details the treatment of these values afterwards.

<table>
<thead>
<tr>
<th>#</th>
<th>Wavelet mother</th>
<th>Features per axis</th>
<th>Description of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Daubechies 3</td>
<td>2</td>
<td>$|cD_4|^2 &amp; |cD_5|^2$</td>
</tr>
<tr>
<td>2</td>
<td>Daubechies 2</td>
<td>5</td>
<td>$|cD_1|^2, |cD_2|^2, |cD_3|^2; |cD_4|^2 &amp; |cD_5|^2$</td>
</tr>
<tr>
<td>3</td>
<td>Daubechies 2</td>
<td>5</td>
<td>$|cD_1|_1, |cD_2|_1, |cD_3|_1; |cD_4|_1 &amp; |cD_5|_1$</td>
</tr>
</tbody>
</table>

Feature set 1 was first proposed by Tamura et al. [186], as the study utilized the same sampling frequency as this one, it was a natural choice and required no change to the original features. The features are defined as the sum of the squared detail coefficients at levels 4 and 5. Feature sets 2 and 3 were both proposed by Preece et al. [157] and are defined as the sum of the squared detail coefficients at levels 1 to 5 and the sums of the absolute values of the detail coefficients at levels 1 to 5 respectively. The same detail coefficients are used in both feature sets, however the use of differing methods of summing provide different results. Both of these feature sets were originally calculated for a sampling rate of 64 Hz, as such to obtain detail coefficients from approximately the same frequency bands, the detail coefficients used within this study were the levels 3 to 7.

### 8.2.2 Frequency Domain Features

For additional comparison three frequency domain features were also employed, To enable this a discrete Fourier transform (DFT) was performed using the fast Fourier transform (FFT) algorithm within Matlab. For efficiency reasons each 2 second window containing 500 values was padded with 12 zero values to bring the size to a power of 2 which greatly reduces the computation time of the process.

<table>
<thead>
<tr>
<th>#</th>
<th>Features per axis</th>
<th>Description of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>Spectral energy</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>Principal frequency</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>Magnitude of the first 10 components of DFT</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>Power of the first 10 components of DFT</td>
</tr>
</tbody>
</table>
Table 8.3 summarizes the frequency domain features analysed. The first set of frequency domain features (set number 4) used was spectral energy, which is defined as the sum of the squared DFT coefficients divided by the number of samples, has be utilized amongst other domain features [9] within a greater feature set and has been used as a standalone set of features [157]. The same is true for feature set number 5 the principal frequency evident within the DFT, as this study is not assessing the combination of differing sets these features were evaluated as standalone extraction techniques, however when extracting features from the magnitude acceleration signal both of the above were combined as they would produce only one feature. Feature sets 6 and 7 are based on the first 10 components of the power and magnitude spectrums of the DFT, both have been used in previous studies and while the pilot study indicated that features derived from the magnitude spectrum performed better both were included in the final analysis.

8.2.3 Time Domain Features

Time Domain features are commonly employed in the classification of daily activities in humans, Table 8.4 summarizes the time domain features evaluated. Feature set 8 is the mean (average) and standard deviation (SD) of the values for each axis, these features have been used in previous studies [151] as has feature set 9 which includes some extensions, additionally including the median (50th percentile), 25th and 75th percentiles[59].

The final set of features (number 10) is also based on extending mean and standard deviation with the values cumulative and final change in orientation. The mean and standard deviation are calculated for each axis individually and a single value is derived for both orientation based features. The final change in orientation is defined as the angle between the 3D vectors represented by the first and last values for each axis and the cumulative change in orientation is the summation of angle between each adjacent set of values which is then normalized by dividing by the number of samples - 1.

If we view the signal as a sequence of vectors of length n named V, each with an x, y and z component, then the final change in orientation can be viewed as

\[
atan2(\|v_1 \times v_n\|, v_1 \cdot v_n)
\]

. Using the same assumptions the cumulative change in orientation can be defined as

\[
\frac{\sum_{i=1}^{n-1} atan2(\|v_i \times v_{i+1}\|, v_i \cdot v_{i+1})}{n - 1}
\]

These calculations are based on the calculation of the angle between two vectors, namely \(a\) and \(b\), calculated using the formula \(\theta = atan2(\|a \times b\|, a \cdot b)\).
### Table 8.4: Summary of Time Domain Features

<table>
<thead>
<tr>
<th>#</th>
<th>Features per axis</th>
<th>Description of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>5</td>
<td>Mean and Standard Deviation</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>Mean, Standard Deviation and 25th, 50th and 75th percentiles</td>
</tr>
<tr>
<td>10</td>
<td>2 + 2</td>
<td>Mean, Standard Deviation and cumulative and final change in orientation</td>
</tr>
</tbody>
</table>

#### 8.2.4 Classifier

In order to compare the performance of each of the different feature sets, a classifier must be used. As discussed in Chapter 4 a number of different algorithms for classification exist. In this research an off the shelf k-nearest neighbour (kNN) based classifier provided with the Weka (Waikato Environment for Knowledge Analysis) workbench [80] was utilised. The nearest-neighbour-like algorithm using non-nested generalized exemplars (NNge) improves the classification performance of nearest neighbour systems and reduces classification time [126]. Nearest neighbour based classifiers have been shown to be effective in previous activity recognition studies [9, 129], this and the provision of Java libraries within Weka motivated the choice of NNge as the recognition engine as this required little work to implement the classifier.

#### 8.2.5 Testing

A number of methods of evaluation were discussed in Section 4.5, of these cross validation was chosen for use. In order to thoroughly gauge the efficacy of the differing feature sets each was analysed using 10 fold cross validation. Cross validation is a popular statistical re-sampling procedure [49], in n-fold cross validation data is split randomly amongst n equally sized subsets. Each subset is then used once as to test the classifier trained with the remaining n - 1 subsets. The empirical accuracy is given by the average of the accuracies of these n classifiers. This improves thoroughness of the testing when compared with the more traditional method of sub-sampling where the data is split into only two sets, one for training and one for testing.

#### 8.3 Results

The primary question posed in this chapter is to determine if research carried out in human activity studies can be used in the classification of object state, and within this question to identify which, if any, of the domains used in this study offer a better performance. Table 8.5 displays the classification accuracies achieved by the 10 feature sets from the three domains. In order to relate these results to the original feature sets described in Section 8.2, the same numbering scheme has been maintained, furthermore the details of the total number of features
extracted for each feature set before giving the average results and standard deviation over the 10 tests carried out in the cross validation process.

Table 8.5: Individual axis results

<table>
<thead>
<tr>
<th>#</th>
<th>Total no of Features</th>
<th>% Accuracy</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>81.6</td>
<td>2.1</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>87.1</td>
<td>1.2</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>85.4</td>
<td>2.0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>83.7</td>
<td>1.2</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>25.2</td>
<td>2.5</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>87.1</td>
<td>2.8</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>87.8</td>
<td>1.9</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>95.7</td>
<td>1.2</td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td>95.8</td>
<td>1.3</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>93.7</td>
<td>1.7</td>
</tr>
</tbody>
</table>

Previously it has been shown that features from the time and frequency domain significantly outperform features derived from wavelet analysis when applied to the classification of human activity, in the study Preece et al [157]. determined the best performing feature set was the magnitude of the first 5 components of the DFT. This study separates the time and frequency domains in order to give a more thorough determination as to the best performing domain when applied to the classification of object state.

The highest classification accuracy was achieved by feature set 9, which uses the mean and standard deviation complemented with the 25th, 50th and 75th percentiles, with an average classification accuracy of 95.8% ±1.3%, this result was however not significantly different to the performance achieved by feature set 8 upon which it was based (p = 0.85).

In order to establish whether, in general, the time domain features outperformed the wavelet and frequency domain features, a number of statistical tests were performed. First, the performance of the best feature set was compared to all feature sets from wavelet and frequency domains, the results show that feature set 9 significantly outperforms feature sets (p < 0.005) from these domains. Further testing was then carried out to compare the remaining time domain features in the same manner. Results again show that feature sets 8 and 10 outperform all feature sets from wavelet and frequency domains (p < 0.005).

These comparisons showed that the time domain feature sets significantly outperformed both the wavelet and frequency domain feature sets, however the accuracies reported in Table 8.5 represent the average across all activities and it is not clear that this is true for all states. To further investigate this, sensitivity and specificity were calculated for the best performing feature set in each of the domains. The comparison shown in Table 8.6 shows that the time domain feature outperforms the frequency domain feature set for five out of the eight categories and the wavelet feature set for seven out of the eight categories.

A secondary research question was with relation to the efficacy of the same feature extraction
techniques when applied not to each axis individually, but, to the combined magnitude of all three axes as a single series which is defined as the square root of the sum of the squares of the x, y and z values at each sampling point. To this end the magnitude series was calculated from the same data used in the primary study and the same feature extraction techniques were applied. There are two exceptions to this, the combination of two feature sets into one as mentioned in Section 8.2.2 and the final set of features which while reduced still requires all three axes to calculate the change in orientation based features.

The results, given in Table 8.7, show that there is a significant decrease in performance \((p < 0.01)\) when the results of the original feature sets are compared against the magnitude feature sets derived from them, with the exception of feature sets 5 and 10 which show no significant change. The best performing feature set is the mean and standard deviation of the magnitude series coupled with the cumulative and final change in orientation which significantly outperforms all others \((p < 0.001)\). This feature however differs from the others as it requires data from all axes to calculate the final two features thus negating possible reduction in data transmitted and therefore energy usage by a wireless device. The required calculations could be performed on the device but this would increase the complexity of deploying multiple sensors as they would require knowledge of the windows used.

The next best performing feature sets were 9, 2 and 3 which are the best performing feature set from the previous and two differing normalizations of the detail coefficients at levels 1 to
5 respectively. These three feature sets were not significantly different from one another and provide a reasonable accuracy of approximately 81%.

Finally the best feature set from the individual results (set 9) was used to perform an investigation into the variance of accuracy with the reduction in frequency. Table 8.8 shows the approximate energy usage as well as the accuracy achieved for the frequencies that the data was converted to.

<table>
<thead>
<tr>
<th>Frequency (hz)</th>
<th>% Energy usage</th>
<th>% Accuracy</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>100</td>
<td>95.8</td>
<td>1.3</td>
</tr>
<tr>
<td>125</td>
<td>50</td>
<td>95.5</td>
<td>0.9</td>
</tr>
<tr>
<td>50</td>
<td>20</td>
<td>95.1</td>
<td>0.8</td>
</tr>
<tr>
<td>25</td>
<td>10</td>
<td>94.9</td>
<td>1.1</td>
</tr>
<tr>
<td>12.5</td>
<td>5</td>
<td>93.2</td>
<td>1.0</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>91.9</td>
<td>1.0</td>
</tr>
<tr>
<td>8</td>
<td>3.23</td>
<td>91.9</td>
<td>1.4</td>
</tr>
<tr>
<td>6</td>
<td>2.44</td>
<td>89.1</td>
<td>2.2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>87.4</td>
<td>1.6</td>
</tr>
<tr>
<td>4</td>
<td>1.61</td>
<td>86.0</td>
<td>1.8</td>
</tr>
<tr>
<td>3</td>
<td>1.2</td>
<td>84.1</td>
<td>1.4</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
<td>76.2</td>
<td>1.9</td>
</tr>
</tbody>
</table>

It can be seen clearly that it is possible to drastically reduce the sampling rate of the accelerometers and still attain a high degree of accuracy. The chosen sampling rate for a given application depends on the desired level of accuracy and the required lifetime of the sensors. As the reduction in accuracy observed after a ten fold reduction in the sampling rate is less than 1% and the power saving is approximately 90%, 25 hz was chosen as the sampling rate to be used within the WAIST project.

### 8.4 Discussion

Only one study was identified which performed a similar classification of object state, the study by Sarangan et al. [176] focused on objects moving whilst in transit rather than being manipulated by humans. This study was based on four states, wobbling, collision, tilting and sliding, and using wavelet decomposition reported accuracies of 100% for the first three states and 85% for the final state. The study utilised biaxial accelerometers at a sampling rate of 233 Hz to record data for evaluation, but the authors believe that a sampling rate in excess of 2500 Hz is needed for effective container monitoring. As the focus of their work was the device developed, very little detail regarding exactly what features were extracted was presented as such there is no scope for comparison with this study.

The maximum achieved accuracy was 95.8%, this was from a combination of mean, standard deviation and 3 percentiles. As it was not evident whether any particular class was limiting
performance, the corresponding confusion matrix was investigated (Table 8.9). While no single state could be identified as the limiting factor there are several pairs which had above average number of confusions, most notably dragging and lifting (30 times), carrying and lifting (23 times) and falling and tilting (21 times). As the largest of these values represents less than 4% of the samples for the given pair, no single state is considered limiting.

Table 8.9: Confusion matrix for best performing feature set

<table>
<thead>
<tr>
<th></th>
<th>Falling</th>
<th>Motionless</th>
<th>In Transit</th>
<th>Tilting</th>
<th>Dragging</th>
<th>Lifting</th>
<th>Carrying</th>
<th>Rolling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Falling</td>
<td>382.0</td>
<td>0.0</td>
<td>0.0</td>
<td>15.0</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Motionless</td>
<td>0.0</td>
<td>399.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>In Transit</td>
<td>0.0</td>
<td>0.0</td>
<td>399.0</td>
<td>0.0</td>
<td>3.0</td>
<td>2.0</td>
<td>2.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Tilting</td>
<td>6.0</td>
<td>1.0</td>
<td>12.0</td>
<td>413.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Dragging</td>
<td>1.0</td>
<td>0.0</td>
<td>7.0</td>
<td>0.0</td>
<td>371.0</td>
<td>13.0</td>
<td>6.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Lifting</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>17.0</td>
<td>372.0</td>
<td>10.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Carrying</td>
<td>0.0</td>
<td>0.0</td>
<td>16.0</td>
<td>0.0</td>
<td>4.0</td>
<td>13.0</td>
<td>366.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Rolling</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>399.0</td>
</tr>
</tbody>
</table>

There are a number of limitations in this study. First all data was gathered while the containers were manipulated by a single experimenter. Differing levels of strength and fitness may lead to different readings for activities such as lifting, carrying, tilting and dragging, as such it is not possible to generalise these findings. While certain assumptions may be made about the strength and fitness of employees handling the transportation of waste, which is where this will be implemented, further research would be required in order to determine if the findings apply to a larger group of subjects.

8.5 Summary

This chapter describes the application of pattern recognition technologies to the classification of object state in a transit context. The methodology is described and details were given about the manner of data collection as well as preprocessing steps taken and how the data was grouped. Each of the states were described in detail and the conditions under which the data was gathered for these states were enumerated.

Three different domains of feature extraction were introduced and the feature sets utilised within the study were detailed, finally the classifier and testing methods were discussed.

This study sought to answer a number of questions, primarily the efficacy of previously proven feature sets from the domain of human activity classification when applied to this area was examined. Furthermore a number of preprocessing techniques were evaluated, in order to determine the impact on accuracy when the techniques were applied.
As these techniques both involved the removal of data, it was expected that there would be a reduction in accuracy. The first method was combining the three acceleration series into a single series using the root sum of squares method. The reduction in accuracy was too great for all but feature set 10, which included two features based on directionality. As this would require a greater amount of coordination of the nodes it is not considered.

The second method sought to address the question of what is the minimum required frequency. The original data was converted to lower frequencies and successively tested using the best performing frequency set. Based on the results a frequency of 25 Hz was chosen for use within this research, as it reduces the energy usage of the component by 90% but incurs a penalty of less than 1%.
Chapter 9

Implementation

The previous chapters have detailed the overall design of WAIST and of a number of components within the application. This chapter will complete the description of WAIST by detailing the implementation of the main components of the system.

9.1 Introduction

As noted in Chapter 6, the modular design of WAIST means that components perform discrete tasks. When combined these individual pieces form the whole application. SIXTH is the mechanism by which much of the interaction between bundles occurs. The most commonly used SIXTH components within WAIST are the adaptors and receivers discussed in Chapter 6. The role of these components is the introduction of data into the middleware and its reception at relevant points as required. Once again the system will be discussed in terms of the three applications it is composed of. The in-situ application that resides on hardware within the consignment of waste, the central application running on a centralised server and the visualisation application, which can be utilised on any desktop machine with the Java runtime environment.

9.2 In-situ

The main purpose of this application is to gather data from the sensors embedded within the transported waste. The hardware that is utilised to achieve this is described in detail in Section 6.4.1, the following sections will describe the software components that realise this purpose.

In order to gather data from the chosen physical devices (SunSPOTs) there is a requirement not only for programming the device, but also the matching components within the application. SIXTH provides examples of these cooperating units of software, however because the
demands of different applications are often unique they are not an exhaustive solution. As such additional components were developed based upon these examples.

9.2.1 SunSPOT Operation

The operation of the SunSPOT devices is designed to leverage the internal operation of the reduced Java virtual machine on which the code is executed. The application is based on a number of threads, one responsible for communication and one for each sensor. By default a single thread samples the current power level every 5 minutes, but all other sensing threads are configurable through messages sent from the application.

The power management is handled automatically by the virtual machine and devices can enter two levels of sleep, shallow and deep. A device will sleep if all threads are sleeping, the choice between deep and shallow is based on the duration of the shortest sleep. Assuming all other threads are sleeping, duty cycling is achieved by the communication thread. This thread repeats the same cycle of listen to wireless medium for 1 second and sleep for 10 seconds.

Each sensor can communicate, either directly or indirectly, with the basestation that it has been assigned. It is expected that the sensors will always be within a short distance of their respective basestation. However, obstructions may make the wireless channel unreliable and in such a case multi hop routing is used to deliver readings.

Use of the wireless channel is minimised as much as possible. This is due to the high usage of power in transmitting when compared to performing calculations (transmission of 1Kb of data is approximately equivalent to processing 3 million instructions [154]). Under normal circumstances it would be prudent to perform much of the calculations on the devices and transmit only when an event has occurred. However in this particular application, concerns over the admissibility of this derived information in a court could be an issue. As such all data is transmitted and recorded.

A number of methods are used to reduce the impact of this decision; firstly the time of the sensor readings are transmitted relative to an initially agreed time. This allows the same precision be attained but data be transmitted as a 4 byte integer rather than 8 byte. Additionally acceleration values are reduced to single precision floating point numbers instead of double precision and light and power values are converted to 2 byte shorts from 4 byte integers. In all cases but acceleration, the conversion is lossless however in that case a small loss of precision is considered acceptable.

9.2.2 SunSPOT Adapter

Data from the SunSPOTs is received within a Connection object, one Connection exists for each device attached to the deployment. Each connection contains a translator which processes the raw bytes transmitted and converts them into objects compatible with the ISensor-
Data interface. This requires the adjustment of the transmission time as well as the creation of a suitable object type. To represent the power and light readings, the default SensorData class is used. However in order to represent the triplet of acceleration values received for the three axes, a custom extension of this class called AccelerationData was created that provides accessors for all of the relevant data.

Once data is realised in this form, it is inserted in to the middleware through an extension of the AbstractSensorNetworkAdaptor class. From this point on, in order to access the data other components must simply query for the data they require. Data can be distinguished in a number of ways, default queries exist within SIXTH that can be used to get data that matches; a particular modality (e.g. “Luminosity” or “Tri-Axial Acceleration”), a given set of sensor ids, contains a number, is bigger/smaller than a given value or was generated within a certain period.

The final requirement of this adaptor is the retasking of the sensor when required. When a retasking message is received, the adaptor will establish a separate connection with the device in question and transmit instructions which are encoded within 3 bytes. Within the first byte, the first three bits identify whether it is a command to start, stop or change the frequency of a particular sensor which is identified by the remaining bits. The rest of the message contains information relating to the frequency and duration of the sensing period. Once acknowledged this connection is closed, leaving only the data reception facilities operational.

\subsection{9.2.3 Data Sender}

The data sender operates in a simple manner, it filters and transmits matching data. Figure 9.1 shows a representation of the query that is used to filter the data. Data needs only to match one of the modalities within the conjugate in order to satisfy the query, or be an event. Once matched data is then transmitted to the server over a socket which is established on startup and reconnected when failed.

Initially when connected the data sender transmits an identifier representing this deployment. This is primarily to aid identifying the correct deployment to send tasking messages and agent communications to. All data sensed by the SunSPOTs and messages sent by the agents are forwarded through the connection to the central application. In response, the server may send tasking messages or communication from the agents on the server back to the deployment. These items are then passed into the middleware through an adaptor.

This adaptor also generates events related to the status of the connection. An event is generated whenever a connection is established or disconnected. These events are another implementation of the ISensorData interface and utilise SIXTH for dissemination. Typically these events are either acted on by agents or visualised. When disconnected messages are not queued, this is because all the relevant data is stored locally within the database and can be accessed later.
9.2.4 Database Insert

This bundle is shared between both the in-situ application and the central application, as such it contains all the same functionality. However as only a subset of the data types exist in the in-situ application, a fuller explanation is given in Section 9.3.5. For the in-situ deployment the following data types are stored within tables within the local database and can be recovered later: “Luminosity”, “Power percentage”, “Tri-Axial Acceleration” and “Event”. Each different type of data is then inserted into specific tables within the database. If the tables do not exist, the bundle will create them before inserting any data.

9.2.5 In-situ Agents

This bundle contains the intelligent agents within the in-situ application. Figure 9.2 shows an example of how a number of agents could be deployed within WAIST. Agents create and use artifacts that are defined within a number of bundles depending on their intended functionality. There are three agents in this bundle;

- the SensorManager responsible for controlling the activation of the sensors,
- the ConnectionMonitor responsible for monitoring the connection to the server and
- the DeploymentManager responsible for managing the bundles within the application.

The role of the SensorManager is activating the sensors when it is appropriate. This is generally done upon receipt of a message either from the ConnectionMonitor when there is
Figure 9.2: Depiction of how agents interact with each other as well as SIXTH and WAIST.

connection problems or from an agent in the central application. Figure 9.3 shows an excerpt of the SensorManager, when it receives a request with the content “start deployment” it adopts the goal deploy. The goal deploy can be achieved by a number of plans depending on the context. The rule beginning on line 7 is chosen when there are no concerns for power in which case all attached sensors are activated.

The ConnectionMonitor agent is responsible for monitoring the status of the remote connection and responding to problems associated with it. When there is a problem with the connection, a request is sent to the SensorManager to activate the deployment. This is done because no messages can be received from the server at that time and activation requests may be missed.

The DeploymentManager agent ensures that all the required bundles are installed and up to date. The agent checks all sources for updated bundles every thirty seconds and is capable of updating all bundles excepting itself and it’s dependencies. This bundle makes use of the RemoteWaistReposArtifact and the LocalBundleSourceArtifact both as sources of information. This allows a single deployment be updated by the addition of a new jar file in a given folder on the machine or all deployments by the addition of a jar file to the remote repository.

9.2.6 Determining Location

Determining the location of the in-situ deployments is carried out by a custom Android application on a device that is capable of utilising GPS satellites to determine location. The application determines its location once every second and is transmitted to the server. When
network resources are unavailable, the location values are stored in a buffer and transmitted on resumption. In the same manner as the data transmitted between the SunSPOT devices and the in-situ application, data is reduced as much as possible to reduce bandwidth requirements. The application transmits the id of the sensor, latitude, longitude, accuracy and speed as well as the timestamp, which totals only 40 bytes.

The operation of this link contrasts with that of the Data Sender mentioned earlier. This is because the small and infrequent nature of the data being sent means that any backlog can be quickly cleared without causing a meaningful delay to the following packages.

A further consideration of the application is the weakness and susceptibility of GPS signals to jamming [76, 92]. In response to this potential problem the application also includes location estimates based on network cell tower triangulation. While this method of determining location is not as accurate as GPS, it provides a redundant source of information with a lower risk of interference.

### 9.3 Central Application

This is the core of the WAIST system, all data is stored and processed here. All sensor data enters the application through the data receiver bundle and from that point is entered into the middleware and distributed through the application.
9.3.1 Data Receiver

The data receiver performs two primary roles, the reception of sensor data from in-situ applications and of location data from the GPS sensors coupled with them. The receiver listens over two separate sockets for incoming connections, one for the in-situ deployments and another for location updates.

The in-situ connection once established by a client is maintained, this is to facilitate bidirectional communication between the agents of the two applications. The nature of this connection is a result of rules of mobile telecommunication networks, where establishing a connection to a client is not permitted. Therefore the client must establish the connection and the server must utilise the same socket to send messages back.

Each deployment identifies itself using a unique id representing the deployment, this can be then used to route communications to the correct deployments. When received all serialised data objects are introduced to the middleware using a single adaptor within the bundle.

The operation of the location reception is somewhat different, no connection is maintained and sockets are closed and disposed of as soon as the value has been received. This is in part due to the fact that there is very little bandwidth use and also the fact that the connection is unidirectional. The data is converted into an implementation of the ISensorData interface designed specifically for location data. This converted object is then added to the middleware through the same adaptor as is used above.

9.3.2 Window Builder

The window builder bundle is designed to support the operation of the classifier bundles discussed in the next section. Its purpose is to gather together individual acceleration readings into windows containing a two second period. The window builder was added to the system to centralise a potentially processor intensive operation and reduce redundancy. This solution offers greater efficiency when compared to the alternative of having each classifier perform the same duty. Figure 9.4 illustrates the way data flows through the related bundles.

The bundle listens for all acceleration values from the middleware and creates a single instance of the Windower class for each sensor. All incoming data is added to the correct windower and stored within a Map of Lists. This method was chosen as it provides the most efficient means of building windows. When a reading is added the timestamp is copied and divided by 1000 and then multiplied by 1000. The resultant value is then used as the key to choose the correct list in the map. This leads to a situation whereby a number of readings in the first and last seconds of an activation period may not be used to form a window. This loss of data is not considered significant as at most a single second of data can be lost.

The bundle also contains a thread which interrogates the windowers every ten seconds prompting windows to be formed of newly received data. In order to optimise the quality of the
classifier determinations a lower bound was placed on the number of readings that must be present to from a window. This accommodates the occasional loss of data due to transmission problems but keeps the overall quality high. The limit is imposed on each one second component of the window to have at least 22 of the 25 expected readings. Once formed a Window is wrapped in a custom object implementing the ISensorData interface, namely the **AccelWindowData** object, and added to the middleware and can be accessed by the classifiers.

### 9.3.3 Classifiers

Based on the research carried out in Chapter 8, a number of components were developed to enable the classification of the state of waste containers. These determinations were based on the readings of accelerometer sensors attached to the containers. All classifiers start by loading the training data from the database and training their classifiers. Once complete the bundle begins to listen for data generated by the window builder.

All classifiers receive the same window data and perform their own analysis and their relevant set of features are extracted. As these classifiers are built using the Weka framework [80], these extracted features must then be converted to the correct type and then passed to the classifier. All bundles currently utilise the same classifier, the nearest-neighbour-like algorithm using non-nested generalized exemplars (NNge).

There are currently two classifier bundles within the system, ```classifier.nnge.msdq``` and ```classifier.nnge.msddodt``` . The first represents the combination of the NNge classifier and the feature set that performed with the highest accuracy in the study conducted in Chapter 8. This feature set contains the average, standard deviation as well as 25\(^{th}\), 50\(^{th}\) and 75\(^{th}\) percentiles of the acceleration values for each axis.
The second bundle combines the NNge classifier with a novel feature set comprising the mean and standard deviation for each axis with the average and total change in orientation of the sensor. This is calculated based on the assumption that all acceleration forces are due to gravity and the angle between successive vectors is calculated. These features are discussed in more detail in Section 8.2.3.

Once a determination of state has been made for a window of data an `AccelClassData` object is created. This object contains the id of the sensor, time of the window, an identifiers for the classifier used and the feature set used and the determined class. This object implements the ISensorData interface and once created, is passed to the middleware to be stored or reasoned about.

### 9.3.4 Server Agents

This bundle contains the intelligent agents within the centeral application. There are three agents in the bundle;

- the `DeploymentManager` responsible for managing the bundles within the application,
- the `GPSMonitor` responsible for monitoring the incoming location data and
- the `ClassMonitor` responsible for monitoring the generated classifier data.

The `DeploymentManager` agent performs the same role as the similarly named agent in the in-situ agents bundle. There are only two differences between the agents. Firstly the bundles that the agents are required to maintain are different. Secondly, as the agent is deployed on the server, there is no need for RemoteWaistReposArtifact to be used. This is because it monitors the same location as the LocalBundleSourceArtifact.

The `GPSMonitor` agent listens to all the location data generated by the deployments and performs a number of functions. The primary function is to activate the acceleration sensors of a deployment whenever the speed of the vehicle it is within falls below a certain threshold. This process is achieved by passing a message to the SensorManager on the deployment through the communicator artifact.

As a secondary function the agent also generates events using the DeploymentArtifact. Figure 9.5 shows the two event types that are generated. The first is a low level event for every reading in which the speed is below the threshold (line 3). The second is a higher level event that is generated once for every stop and includes duration information (line 11). These events include the location information such that there will be a record of every location that the vehicle was stationary.

The `ClassMonitor` agent is responsible for monitoring the output of the classifiers. Depending on the number of classifiers deployed, it may attempt to determine the class of a particular window of data based on the results of all the classifiers. This currently takes the form of assuming the majority is correct but could utilise a weighting system where certain classifier
and feature set combinations perform better for different classes. Assuming that a determination has been made as to the class of a particular window of data for a given sensor, the agent will then do two things. First look for readings that might indicate dumping has taken place and secondly look for inconsistencies between readings from the same deployment.

The first is simply checking for one of the states that show it is being manipulated by a person, such as tilting, lifting, carrying, dragging or rolling. This is by no means definitive proof that illegal dumping has occurred. However, an event will be generated and the associated location information recorded such that the site may be inspected if it is considered a possible dump site.

Additionally the agent compares the class determinations made for different sensors within the same deployment recorded at the same time. This is an attempt to identify situations where a single container is being manipulated in a different way to the others. Again this may simply indicate the containers are being repositioned but as above an event will be generated such that the site may be investigated if necessary.
Figure 9.6 shows a rule from the agent which is responsible for comparing two readings once agreement has been reached between the classifiers. This rule only applies to the context where there are two sensors within the deployment. Initially a list is compiled containing all the results and when the they are not the same an event is generated.

### 9.3.5 Database

Both database bundles operate to enable the storage of raw and processed sensor data within a MySQL database. The bundles insertion and access were separated to allow the reuse of the insertion bundle within the in-situ application, however the two bundles are considered a cohesive unit. This is due to the natural consequence that a change to how the data is stored will most likely result in a change to how that data is accessed.

The manner in which all other bundles interact with the database means that the technology on which the database is based can be easily changed without any impact on the operation of other bundles. Therefore SQLite, Oracle or Microsoft SQL Server could be utilised as long as the same external interface is provided.

Figure 9.7 shows a representation of the WAIST database. All sensed data is stored within the tables `powervalues`, `accvalues`, `lightvalues` and `locationvalues`, this is supported with information relating to the sensors, their modalities and the deployments they are connected to. To record the outcomes of the classifiers, both the calculated features and the resultant classes are stored. Finally all events within the system are stored in order to help understand the actions of the application.

The information stored within the database is made available to the visualisation application by the access bundle. The bundle listens for requests over a socket connection and forwards the results to the application. The requests take the form of a Java object describing the required data, this prevents the database being exposed to the outside and risking data loss.

### 9.3.6 Live Stream

Similar in operation to the database access bundle, the live stream bundle maintains an open socket connection and waits for requests. Requests are sent in the form of the Query objects discussed in Section 5.5.1. Once received, all data that matches the query is forwarded over the established connection. Once created the query can not be altered, as such connections are closed and reopened when this is required.

The bundle also includes a reduced version of the interface used in the database access bundle, allowing the identification of deployments that are currently active.
9.4 Visualisation

The visualisation application is built using the Eclipse framework and OSGi. It is designed in a modular way so as to be as extensible as possible. The application itself is defined by the visualisation core bundle, which presents an empty application window split into three components. All other views and functionalities are added by their respective bundles.

9.4.1 Visualisation Core

This bundle adds only the shape of the main application, however through the exposure of a number of interfaces it allows the contribution of contents. The design of application is defined in an XML format, which allows the specification of visual elements in a hierarchical fashion. Figure 9.8 shows a reduced excerpt from the main definition file, in this line 1 defines
the application window and lines 2 and 3 add a single perspective (grouping of visualisation components). The perspective contains a single PartSashContainer which breaks the window into a number of sections. Within this there are three PartStacks defined which can contain multiple tabbed views (parts). The ids of these partstacks is what allows the extension of the application.

![XML excerpt](image)

Figure 9.8: Excerpt from XML definition of visualisation application.

In order to contribute to one of the exposed components from another bundle, a similar XML definition is required. Figure 9.9 shows the addition of two visualisation methods from the light bundle. The first line defines where the contribution is to, in this case the visual partstack, and the following two lines each contribute a single view (part). These contributions are linked to a concrete implementation through the contributionURI tag which defines the location of the class.

![XML excerpt](image)

Figure 9.9: Excerpt from XML definition of view contributions.

The system does not require that a view implements a particular interface in order to contribute to the user interface. Instead the system uses dependency injection [155] and annotations to allow the specification of the methods that contribute. Figure 9.10 shows excerpts of the LightAccelerationPart class, of particular note are the annotations @PostConstruct and @Inject. The first indicates that this method should be called after construction to generate the user interface contributions and the second is an example of simply requesting a variable that is supplied by the framework.
public class LightAnimationPart implements DeploymentViewer {
    ...
    @PostConstruct
    public void createPartControl(final Composite compositeParent) {
        ...
    }
    @Inject UISynchronize sync;
    protected void render() {
        ...
    }
}

Figure 9.10: Excerpt from an example class contributing a view.

9.4.2 Location Bundle

As stated in Section 6.6, this bundle is required by default in the application. All contributions from this bundle are based on the MapPanel project [169] which provides an implementation for a manipulable Java based map panel using the data from OpenStreetMap [148].

In order to display the locations of deployments (represented as blue truck icons) it was necessary to modify the operation of the panels paint method. A number of problems had to be solved before the icons could be painted. First the location information of the vehicles needed to be accessed, and then the coordinates required translating into a relative position on the panel.

The first problem is solved by registering a listener for data with the modality “GPS Data” and adding some variables to store the data. The second problem required some calculation and is achieved the following formulae:

\[
ypos = \left\lfloor \frac{(1 - \log(\tan(lat) + (1/\cos(lat))) / \pi)) / 2 \right\rfloor \times ymax
\]

and

\[
xpos = \left\lfloor \frac{(lon + 180) / 360) \times xmax \right\rfloor
\]

where \(lat\) is the latitude, \(lon\) is the longitude and \(ymax\) and \(xmax\) are representations of the tile size and zoom level.

The bundle offers two different views of location information. The first shows only the most recent location for every vehicle and the second shows every reported position of a single truck simultaneously. The second is achieved by keeping old location readings and by drawing a circle of colour in place of an icon. However in order to switch between the two modes of operation, the bundle needs to contribute to the menu of the application. This is done in the same manner as the contribution of a view.

Figure 9.11 shows excerpts of the XML definition which contributes an item to the menu. The framework utilises Commands and Handlers to specify menu operations. A command is a declarative description of a component and is independent from the implementation details and a handler is the class which is executed once the command is called. Similar to the definition of views, the handler does not need to implement an interface or use a particular name for a method. It is only required that the chosen method is annotated with @Execute.

122
The first three lines show the definition of a command for disabling the heat map functionality. The command is associated with the application through the parentElementId which specifies the id of the application it belongs to. Lines 5 to 7 show the definition of a handler. Within this definition there is a link to the id of the command it satisfies as well as a link to the class that implements the functionality. Finally lines 9 to 13 show the addition of a menu and menu item to the application. The menu item is linked to the command it should call and the handler is resolved when required.

Figure 9.11: Excerpt from XML definition of menu contribution.

9.4.3 Acceleration

The acceleration bundle contributes three different views to the application as well as a number of commands to the menu. The manner of contributing these elements does not differ from the steps described in Section 9.4.2, as such this will not be discussed further.

Animated View

The animated acceleration view offers an instantaneous representation of the containers within a single deployment. The view is developed using the Java bindings for the OpenGL API. The containers are represented as three dimensional crates floating in a black background. The positioning of the crates is altered depending on the configuration of the deployment, and can support between 1 and 4 crates being represented.
The orientation of each crate is defined by the readings received from the sensor, based on these the roll and pitch of the crate are calculated. The formulae to calculate these are:

\[
\text{roll} = \text{atan}2(Y, Z) \quad \text{and} \quad \text{pitch} = \text{atan}2(X, \sqrt{Y \cdot Y + Z \cdot Z})
\]

where \(X, Y\) and \(Z\) represent the readings of the x, y and z axes respectively. These calculated values of roll and pitch are then used to rotate the crate about the z and x axes respectively.

To smooth the animation of the crates the data is passed through a low pass filter. A low pass filter eliminates high frequency noise that would cause jittering in the animation. This is achieved by combining the previous value with a percentage of the difference between it and the current value.

This view operates on an assumption that the crate is only under the acceleration due to gravity and as such may not reflect the actual position of the container.

**Graph Views**

Both graph views are implemented based on the library provided by the SWTChart project [158]. This provides a number of utilities for producing graphs in the Standard Widget Toolkit which is used by the Eclipse framework. Two types of graph are produced as part of the bundle, each visualised the data in a different manner. The first groups data together based on the sensor it came from, thus showing a graph for each sensor within a deployment and each graph having a series for each of the axes.

The alternate view groups data based on the axis, in this way there will always be three graphs, one for x, y and z values. For every acceleration sensor within the deployment, a series is added to the three graphs. This allows a different method of comparison where anomalies may be easier to identify. This is however susceptible to distortion when the orientation of the sensors is not the same. This problem could be solved by identifying the difference in orientation and adjusting the readings, but has not been attempted here.

The charts are refreshed 10 times a second and only the previous 5 second period is graphed. This is to reduce the resource demand that generating the charts causes.

**9.4.4 Light**

The light bundle offers two visualisation methods; the first is the more traditional graph approach and the second is an experimental animated visualisation. The light graph, because it updates at a much slower rate is able to utilise a more fully featured graphing library JFreeChart [145]. This library was originally used with the acceleration views but was not capable of graphing the data in real-time. However as light data has a much slower sampling rate, it is perfectly suited to this application. The view shows values sensed during the period of the previous 5 minutes. All sensors are depicted on the same graph as a series and differences are easy to identify.
The alternate visualisation of light values is built using the Java bindings for the OpenGL API. It uses a single white light source to represent each of the sensors on a black background. The light value of the latest sensor reading determines the brightness of the light source and thus the size. Linear interpolation is used to colour the screen such that the larger the light value the further it expands.

9.4.5 Database

The database bundle contributes a single view to the control section of the application. This view allows control of the connection to the server and the data stored there. A protocol was developed to allow the bundle communicate with the server without exposing the underlying data storage to the outside. This prevents any actuation of the database and severely restricts the data that can be accessed. This was motivated primarily by concerns over data protection.

Once connection is established the bundle queries the start and end time of all the deployments and displays only the completed deployments. The user can select the deployment from the list and stream the data from it. The data transmission process begins immediately and a large buffer is built up in the visualisation application, this allows the speed to be controlled from the view. There are two options for speed; the readings will play in real-time or the speed can be altered using a slider.

The slider offers a linear sliding control of the speed by deciding what percentage of the actual time between readings is observed. At the lowest setting, this is equivalent to operating in real-time mode and as the highest setting the readings are processed without a gap between them. In this case the processing power and network capabilities of the device running the operation are the limiting factors for the top speed.

9.4.6 Live Data

This bundle also contributes to the control section of the application and operates in a similar manner. As the focus of this view is only live data, it utilises a reduced version of the communication protocol defined for the database. This only allows access to the details of the deployments in the system and prevents alteration of the database. It also allows the user find the status of a particular deployment, where it is defined as being offline if it has not transmitted a reading in the last 5 minutes and online otherwise.

The user can select any deployment from the returned list, this includes both online and offline deployments. The user then has two options with regard to what data is streamed to the application, each represented by a button. The first requests location readings from all of the deployments in progress, but detailed sensor readings from only the selected deployment. The second button restricts both the location and other sensed data to just the selected deployment.
When the connection is established a SIXTH query object is transmitted to filter the data before transmission. This query is automatically generated based on the deployment selected and the option chosen. When the first option is chosen a conjugate query is created. This contains queries to match either the modality “GPS Data”, the ids of the sensors in the deployment or an Event Query matched to the deployments id. In the case where the second option is chosen, the modality query is removed and the id of the GPS sensor of the deployment is added to the id query.

Before transmission, all unnecessary pieces of data are removed to reduce the bandwidth requirements. This is achieved through the use of a custom serializable object to represent each of the data types transmitted. Once received the data is then converted to its original form and introduced into the application. This process is also repeated in the above Database streaming.

Because SIXTH is a full featured middleware, there is a large amount of meta-data associated with every reading, including objects describing the source of the data within the middleware. In the memory of the application, there would be a single instance of any given object which would be shared amongst the data objects. However when transmitting the entire object and its contents are serialised, leading to a larger than necessary size. Through this conversion, there can be a reduction in size of as much as 82% for acceleration values and 88% for light values.

9.4.7 Events

The final bundle contributes a single view to the events section of the application. This is designed as a catch all method of depicting events. As such, on creation it queries the core of the application and generates the view based on the events defined within the system. For every type of event a check box is added to the view and initially defaulted to false. Thus forcing the user to actively choose to display events of a given type.

For every event displayed, the following information is given;

- an identifier representing either the id of a sensor or deployment depending on the type of event,
- the type of the event,
- the time at which the event occurred and
- whether or not the event is associated with a particular location.

When an event has an associated location, this can be mapped by selecting the given event. On selection an event icon is added and the map is centred on the location of the event. This provides an easy method for reviewing all the locations where a vehicle was stationary for a period of time in an efficient manner.
9.5 Discussion

This chapter details the implementation of the WAIST system focusing on each application in turn. The capture of sensor data by the in-situ application is described with a particular focus on the reasoning component based on the work described in Chapter 7. The central component of the system is described, detailing the reception storage and access of the data. Of particular focus in this section is the description of the classifier components based on the work described in Chapter 8 and the reasoning components based on the work in Chapter 7. Finally the implementation of the visualisation application is described in detail with an emphasis placed on the extensibility of the architecture.
Chapter 10

Evaluation

10.1 Introduction

Within this thesis, technologies from machine learning and agent-oriented programming are applied to the domain of intelligent transportation systems for waste management. Three objectives were discussed in Section 1.3.

Ob. 1, to investigate the applicability of techniques for human activity recognition to object state within a waste tracking application, was detailed in Chapter 8. Ob. 2, to study the use of agent-oriented programming languages as a means easily to augment the functionality of a waste tracking application and control the application itself, was evaluated in Chapter 7.

Finally, Ob. 3 was to develop a waste tracking application demonstrating the use of pattern recognition and agent-oriented programming as part of an intelligent transportation system. Chapters 6 and 9 have detailed the design and implementation of this application.

This chapter seeks to undertake an evaluation of the complete WAIST system. Evaluation of such a system is a non-trivial task. As the purpose of the application was to enable the monitoring and validation of the transportation of waste shipments in Ireland, it was decided that the system should be evaluated with respect to its efficacy in performing this task.

10.2 Evaluation

In order to evaluate the application, an in-situ deployment was set up and two sensorised containers deployed within a van. The van simulated six typical waste materials pickup and transfers and stopped in a number of locations. The driver was instructed to stop and simulate dumping of the contents of a container during at least one of the simulations. The number of dumping simulations and their associated locations was not confirmed until after analysis.

While the simulation was in progress, all readings were streamed in real-time to the central
application for visualisation. In order to assess the performance of the application in this evaluation, a number of criteria were developed by which the actions of the application and the outcomes could be judged:

1. The timeliness of the arrival of the sensor readings at the central application was assessed.

2. The ability to identify all the stopping locations used in the simulation.

3. The identification of the location in which the illegal disposal of waste was simulated.

4. The ability of the agents within the system to effectively manage the energy resources of the sensors i.e. avoid unnecessary acceleration sampling.

These criteria are discussed in the following sections.

10.2.1 Timeliness

The assessment of the timeliness of the arrival of sensor data at the central application is relevant in order to ascertain whether the application responds in real-time. As the real-time responsiveness is considered an important quality, assessing the difference in time between sensing and visualisation is carried out.

Initially all devices were synchronised as closely as possible to the same time, approximately less than a quarter of a second in difference, this is achieved through the use of network time protocol [131] (NTP) on the server, computer running the visualisation application and the machine hosting the deployment. The phone providing the location service was also updated as accurately as possible using information from the NTP servers.

This level of synchronisation, a difference of less than a quarter of a second is considered as sufficient preparation. This is due to the nature of the application, as it is monitoring human activity, the timing of a reading being ± 250 milliseconds has little real effect. A driver being monitored can not be expected to complete dumping and speed away in this amount of time.

The bounds as to what is considered near real-time [183] responsiveness cannot be less than this synchronisation value. Based on the usage scenarios for this application a response time of less than 1 minute for the delivery of location data and visualisation is considered acceptable.

The evaluation was implemented by measuring the difference between the timestamp associated with the sensor reading and the current system time measured on the server. These values were later analysed based on the acceptable value established here.

10.2.2 Stop Identification

In evaluating the application this criterion is essential: since the ability to enumerate and map all the locations at which the vehicle stops is fundamental. As fundamental as the performance
of this function is, the evaluation is particularly simple to complete. Following the end of the simulated waste transfer, the event list was examined in order to ascertain whether all of the stop off points were listed.

In this instance only the simulated stops are included in the evaluation, excluding all traffic related stops. This is primarily because maintaining a log of every traffic light stopped at would place an onerous burden on the driver. Additionally the determination as to whether the vehicle fell below the speed threshold, which translates to 3.6 Km/hour, may be difficult to estimate.

10.2.3 Illegal Disposal Identification

This section of the evaluation focuses on the operation of both the classifiers and the agents reasoning about their outcomes. As the purpose of this application is to monitor and validate waste disposal, it is important that the core function of identifying illegal dumping during the simulation is assessed.

To this end, the application will be evaluated by the ability to determine at which of the stop locations tampering has occurred. As there are a number of ways in which it may be inferred that a container has been tampered with, instances where one or more of the states tilting, carrying, lifting, rolling or dragging have occurred will be specifically looked for.

10.2.4 Energy Management

Minimising the energy usage of sensors has been at the heart of the design of WAIST. The primary method of achieving energy savings is through managing when the sensors are actively sensing acceleration. This function is carried out by the agents within the system and acceleration sensors should be activated when the container is stationary or when no connection to the server is available.

As events are generated for both of these conditions and also the activation and deactivation of the sensors/deployment, the event log will be analysed to determine that this function has been carried out correctly.

10.3 Results

During the evaluation period a single deployment was operated. Six simulated shipments were made between a number of differing start and end points. Figure 10.1 shows screen captures taken from the visualisation application in heatmap mode detailing the routes taken on each of the simulated shipments. During each simulation there were a number of stops of differing duration and during an unknown number of these a single container was removed from the
vehicle and dumping of the contents was simulated. The following subsections describe the results based on the criteria outlined in Section 10.2.

Figure 10.1: Visualisation representation of all 6 simulated shipments

10.3.1 Timeliness

The ability to report the location of the deployment in real-time was assessed. Table 10.1 shows the results compiled from the raw timing data. The average delay between sensing and reception was 1703 ms. This value is well within the limits defined as to what was considered
an acceptable result.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>1703</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>7548</td>
</tr>
<tr>
<td>Minimum</td>
<td>40 ms</td>
</tr>
<tr>
<td>Maximum</td>
<td>95868 ms</td>
</tr>
<tr>
<td>% below 60 s</td>
<td>99.5 %</td>
</tr>
<tr>
<td>% below 1000 ms</td>
<td>84 %</td>
</tr>
</tbody>
</table>

Figure 10.2 shows a plot of the difference in sensing and reception with respect to time, of particular note are the large spikes. These outliers explain the standard deviation of the results and are assumed to be as a result of loss of communication coverage. When coverage is restored the readings are delivered in quick succession causing dramatic drops.

Further analysis shows that 99.5% of the readings were delivered within the limit established in Section 10.2.1. The fact that 84% of the readings were received within 1000 ms of sensing illustrates the general running time to be expected of the application. These results far exceed what is necessary to achieve the aims of the application.

While the final step of transmitting the data to the visualisation application would incur further delays, these are considered small enough to disregard. Additionally this added delay is dependant on the location of the device running the application.
10.3.2 Stop Identification

Following the simulated deployment the operator provided annotation of the dataset to confirm the locations in which the deployment had stopped. These were then compared against the database playback of the deployment. Figure 10.3 shows a screen shot of the process, as each stationary event occurred it was highlighted on the map with an event icon.

![Figure 10.3: Screenshot of stop location review](image)

The evaluation process was somewhat hampered by the granularity of the results, as there were in total 118 instances where the deployment was stationary for a period of time. These results motivate the addition of filtering functionality as a means to simplify analysis.

The lack of a filter based on the duration of the stop (which was recorded) meant that the events had to be iterated individually in order to check an event was recorded. The addition of such a filter would greatly improve this process as short stops such as traffic lights, which make up the bulk of the readings, could be easily eliminated from the results.

Based on the functioning of the GPS sensor within the deployment, every location where the speed of the vehicle dropped below 1 metre per second was recorded. There were in total 3377 readings where this condition was met, all of which were formed into the 118 instances described above. Failure in such a system could only come as a result of failure in both GPS and network based location determination. As such all stop locations recorded by the operator of the deployment were represented by events and associated with the time and location they occurred. This delivers on the system requirements.
10.3.3 Illegal Disposal Identification

In order to evaluate the ability of the system to identify locations in which illegal dumping may have occurred, the events generated by the classifiers were examined. This was done for each of the simulated shipments separately. This process was completed in a number of phases: first the events from the live system were viewed and then based on the information contained within the in-situ deployment, some events were eliminated.

Two types of events are generated by the classifier: one where a window of data was classified in a state that implied it was being manipulated by a person and another where two containers are in different states at the same time. As the acceleration sensors are only activated when the deployment was stationary or had no connection to the server, the number of events to be examined was small.

There are a number of reasons for the elimination of events generated. Primarily this was during periods that were too short such as stops at traffic lights, secondarily any events that took place while GPS readings were delayed but later show the vehicle to be in motion. Where events were eliminated typically they consisted of only the first type of event. This highlights that both containers were in the same state at the same time.

Table 10.2 shows the results of this analysis. In the first, third and fourth shipments there were no events generated by the classifier as all windows were classified as either idle or in transit. During the course of the second shipment 10 events were generated. These are attributed to two periods where the deployment had begun to move but the acceleration sensors were still active. In this situation (gentle acceleration) the windows of data were erroneously classified as tilting.

Table 10.2: Events breakdown

<table>
<thead>
<tr>
<th>Shipment number</th>
<th>Number of classifier based events</th>
<th>Events disregarded</th>
<th>Unexplained events</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>78</td>
<td>18</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>12</td>
<td>0</td>
</tr>
</tbody>
</table>

In the fifth deployment there was a larger number of events generated. As with the second deployment, a number were eliminated because they occurred where the deployment was moving but the sensors had yet to deactivate. There was however a number of events that took place while the vehicle was stationary, which could not be eliminated. Thirty of these events related to the classification of a single container within the deployment as the state of either carrying, lifting or tilting. The remaining events related to the disparity of state between the different containers, where the other container was classified as motionless. When viewed through the visualisation application there was very apparent tilting of the container.
a number of times.

Figure 10.4 (also available here: https://www.youtube.com/watch?v=Gh2_e5pNUlY) shows the relevant time period when viewed through the visualisation application. The cause of the generated events is very clearly displayed by the changes in roll and pitch of the container, which occurs approximately 90 seconds into the video.

Finally during the sixth deployment there was again a number of events generated. On further analysis these could be attributed to readings taken while in motion during a period of unstable connection to the server.

Based on the events generated by the agent within the central application and further analysis, a single instance of dumping was identified. This followed the elimination of a number of the events as being erroneous. Following the analysis only one tampering site was highlighted by the operator, this corresponded with the location and time of the 30 unexplained events captured by the system during deployment 5.

10.3.4 Energy Management

The energy management of the system was analysed based on the events generated, specifically by comparing events describing stops and their duration with deployment start and stop events. Additionally, events about the status of the connection were also compared to the start and stop events of the deployment.

To analyse the effectiveness of the system, the average time between the first occurrence of a stationary location reading and the subsequent initiation of a deployment was measured. On average there was a delay of 972 ms between the two. Additionally to see if the switching off
of the deployment was carried out at the correct time the difference between the end of the stationary period and turning off the sensors was calculated. The average of these values was 962 ms.

There was only two clear instances where a disconnection caused the deployment to start, the average delay between the disconnection event and the deployment starting was 566ms and deployments were stopped on average 126 ms after the connection was re-established. Based on the delays visible in Figure 10.2, there appears to be more than just two periods in which connection was unavailable. A possible explanation for the discrepancy is the fact that an error will not be registered until an attempt is made to send data over the link.

Sensors are activated and deactivated with little or no delay, as such there is not a great deal of time where the deployment is either stationary and not sensing or moving and sensing.

### 10.4 Discussion

This chapter describes the method and results of an evaluation carried out on the WAIST system. The methodology of the data collection was introduced and the criteria by which the system will be evaluated were described. The results were then described in terms of the criteria of timeliness, stop identification, tamper detection and energy management.

The average delay in reception of location readings was calculated as 1703 ms, for the purposes of this application this level of delay is excellent. Further analysis of the data shows that over 84% of the readings are received within one second. While the final step of transmitting the data to the visualisation application would incur further delays, these are considered small enough to disregard. Additionally, this added delay is dependent on the location of the device running the application.

Identification of the stop locations of the deployment performed perfectly. However, it was apparent during the validation process that the addition of functionality to filter the information based on duration of stop would simplify the process of reviewing stop locations.

The location of the dumping simulation was correctly identified by the system. However the system also generated a number of events indicating tampering may have occurred in another location on the same simulation as well as in two other simulations. These false positives could only be disregarded after manual analysis of the data received by the server and the data stored on the in-situ deployment. In order to mitigate the problem, the agents within the system would require the addition of a number of behaviours in order to achieve a greater level of intelligence. Possible improvements will be discussed in Section 11.2.3.

While false positives should be expected, the average accuracy of classified windows was 90% rather than the maximum in testing of 95.8%. This appears primarily due to a disproportionate number of windows erroneously classified as tilting. It is thought that the number of
windows where the vehicle was accelerating from stand still is the root cause, as the readings look similar to a slight tilt.

Finally the agents responsible for energy management activate and deactivate the system with little or no delay, meaning there is not much time where the deployment is stationary and not sensing and there is little time where the deployment is moving and wasting energy by sensing. There is a discrepancy between the connection log and the location delay shown in figure 10.2 as a result the deployment was only started twice for lost connection.
Part IV

Closing
Chapter 11

Critique and Further Work

This chapter begins the closing part of this thesis with some critiques of and possible extensions to the work presented in the previous chapters. Section 11.1 acknowledges limitations to aspects of the work described in this thesis. Following this Section 11.2 discusses possible avenues for further research to build upon the work described in the preceding chapters.

11.1 Critique

The work presented within this thesis represents a novel contribution to the state of the art in the area of intelligent transportation systems for waste management. There are, however, a number of limitations relating to the wider applicability of techniques discussed herein.

Principle amongst these is the findings of the overall system evaluation. Results based on the simulation of six simulated shipments were excellent. For each of the criteria assessed the results met or surpassed the desired levels.

The methodology utilised in evaluating the overall system could be considered limited. The study attempts to approximate the same conditions as a typical waste management or haulage company. Alternatively an examination would have analysed results based on sensors situated within actual waste shipments. The latter method of evaluation, which is closer to the real use of the system, can be considered more like a product trial than an evaluation.

When comparing both methods, it can be argued that the results achieved for a number of criteria would likely have been duplicated. However, the criterion of stop locations and illegal dumping identification would most likely not have been analysed. The first because employees of the haulage company would not be expected to perform extra duties such as recording stop locations. The second criterion would not have been evaluated because it would not be expected that a haulage company would perform any illegal dumping with prior knowledge of the monitoring occurring.

The classifier components of this work have some small limitations. All data gathered while the
containers were manipulated by a single experimenter. Differing levels of strength and fitness may lead to different readings for activities such as lifting, carrying, tilting and dragging, as such it is not possible to generalise these findings. While certain assumptions may be made about the strength and fitness of employees handling the transportation of waste, further research may be required in order to determine if the findings apply to a larger group of subjects.

During the evaluation of the overall system in Chapter 10, it was highlighted that a number of windows of data captured while under acceleration were classified as tilting. The prevalence of these occurrences reduced the effective accuracy of the classifier to 90% rather than the 95.8% reported in testing. Of note is the fact that the confusion matrix presented in Section 8.4 indicated no overlap between these states.

The disparity between these findings may be the result of a single factor which was not constant. In the collection of training data, signals were captured while the container was within the passenger compartment of a mid-sized (D-segment) vehicle. Whereas during the evaluation of the system data was recorded while the containers were within the storage compartment of a cargo van. Additionally, both vehicles were piloted by different drivers.

Further investigation is required to ascertain whether recapturing the data for the in transit state would improve the working accuracy of the classifier.

Finally, there were some limitations to the evaluation of the agent-oriented programming integration. These stem from the number of participants within the study and the conditions of the experiment. The participants were students of an agent-oriented programming course asked to voluntarily take part in the study. The study was run in a laboratory setting following a week long intensive course. This was the final commitment of the students and took place on a Friday afternoon.

After completing the first problem, seven of the students chose to leave without completing the second. Some may be attributed to a perception that the problem was more difficult or simply that they wished to arrive home early.

With respect to the generalisation of the results, sample size alone is a limiting factor. Locating participants for this type of study is quite difficult as the participants must be familiar with agent-oriented programming. While the results are quite positive, further study is required in order to ascertain the wider applicability of the work.

11.2 Further Work

This section discusses a number of ways in which the work presented in this thesis may be extended. The possible avenues discussed in this section are by no means an exhaustive list, but represent the most likely direction for work to progress.
11.2.1 Further Study of Object State Classification

Chapter 7 describes a study into the discriminative ability of a number of feature sets when classifying object state within a transportation context. The features used were varied and from domains that are commonly employed in similar classification problems. However the study did not examine whether differing combinations of classifiers and feature sets would produce improved results.

A further study could be carried out to examine the discriminative ability of different classifiers when applied to the same problem. Studies have utilised hidden Markov models [138], support vector machines [204], decision trees [127] and other methods which may provide a greater discriminative ability.

In the event that no other classifier provides improved results, research could be applied to the combination of a number of classifiers into an ensemble classifier. Situations where some classifiers are more accurate with different states may allow an improvement in this way.

Additionally this study should address the limitations discussed in Section 11.1, by capturing training data using more vehicles and a number of subjects.

11.2.2 Investigation of Generalisation of Application

The WAIST system by it’s very nature is modularised, however little attention has been applied to utilising the design and components within the context of another transport based application. Further attention could be focused on the task of generalising the components of the system in such a way as to make them reusable in a greater number of situations.

The core of this application with such generalisations could form the basis of other activities such as content tracking in order to detect damage in transit similar to [24, 176], support the locating of containers as in [1] or monitor the atmospheric conditions within a shipment of perishable goods [93].

11.2.3 Improvements in the Functionality of Agents and the Events they Generate

The agents within the WAIST applications successfully performed the roles they were assigned, however there exist room for improvement. The number of false positives by the agents responsible for detecting illegal dumping could be improved through the addition of new rules to identify the situations causing them and more event types representing greater reasoning.

An example of such a behaviour would be an agent which examines the classifier based events generated and compares them with the location based events. This would allow, with a slight delay, a determination be made as to whether a stop was of too short a duration, or if the location readings were simply delayed and the deployment was in transit at the time.
By design the agents within the system were restricted to the middleware, this choice was explained in Section 5.6. However in other applications without the same constraints agents could also be situated within the network itself. The investigation of the cooperation of different agent types, some operating on devices and some on the middleware, could prove interesting.

As the middleware already includes a basic platform independent communication functionalities there is no requirement to have the same agent platform utilised by agents on devices and within the middleware. The use of different agent-oriented programming languages within the system to cooperatively complete tasks could introduce an interesting dynamic, where agents of differing types perform roles that they are most suited for.

11.3 Summary

This chapter described the limitations that are evident within the findings of this work. Following from this possible avenues for the continuation of the research present in this thesis.
Chapter 12

Conclusions

This thesis has presented a comprehensive description of the Waste Augmentation and Integrated Shipment Tracking (WAIST) system and more specifically the technologies underpinning it. The contributions of this work were detailed in Section 1.3, this section discusses how these have been achieved.

Chapter 4 introduced the background of pattern recognition, highlighting the techniques that are used generally in the area. Chapter 8 continued from this to describe a study in which proven techniques from the field of human activity classification were applied to the classification of object state within a transportation context.

Chapter 8 also contained evaluation which indicated that many of the previously defined feature sets utilised for classifying human activity were suitable for use in the determination of object state. The highest accuracy achieved in testing was 95.8% when discriminating between eight different states.

Additionally the study examined the efficacy of a method of signal combination which reduced transmitted data by up to two thirds and the discriminative ability of the best performing classifier was examined with varied sampling rates in order to identify a suitable lower bound for use within the WAIST system. Based on this the sampling rate of 25 Hz was chosen for use within the system.

Finally the implementation of these classifiers as a component of the WAIST system was described in Sections 9.3.2 and 9.3.3.

Chapter 3 described the area of agent-oriented programming and defines what constitutes an agent in the context of this thesis. The manner in which agents may be integrated into a software system was also described and a number of environment interfaces were reviewed. Relevant agent-oriented programming languages were introduced and the choice of environment interface (CArtAgO) and programming language (ASTRA) was explained.

Chapter 7 outlined the manner in which the chosen interface standard and agent-oriented programming language operate together. This was followed first by the detail of how CArtAgO
was integrated as a component of the middleware being used; and then specifically what additions were made to WAIST to enable agents access and configure components of the application.

Chapter 7 also included an evaluation of the described work in terms of it’s ability to simplify the process of applying behaviours to a sensor network application. The evaluation was carried out by a class of Msc students enrolled in the Advanced Software Engineering programme in University College Dublin.

Participants were required to solve two problems, one using an agent programming language and one using Java. The participants where split into two groups where one group used Java in the first problem and the rest used the agent programming language, this was then alternated for the second problem. Participants undertaking the same programming tasks using different implementation technologies is a common approach in the software engineering community.

After the allotted time, students were required to submit their work for analysis, whether finished or unfinished. In the first problem, significantly more tasks were achieved by the students using the agent programming language than the those required to perform the same tasks in Java. The second problem also featured a slight increase in the average number of tasks completed, however the increase was not significant. T-tests were applied to the results and p values of 0.0002 and 0.89 were calculated for problem 1 and 2 respectively.

In order to gain a fuller understanding of the perceptions of the students, a subjective analysis was also performed. The students were required to complete a survey containing several questions relating to their perception of the utility of the agent language in solving the problem. This results strengthen the outcome observed in the first problem, indicating that students felt that the agent language was easier to use and read than Java and also made solving the problems easier.

Building on previous results, the WAIST application, an end to end wireless sensor network application for the monitoring and validation of waste transportation, was then developed. The background information upon which this application is built is detailed in Chapters 2 to 5.

Chapter 6 details the problem that is to be solved and highlights inadequacies of the current practices and systems in place. Following from this the functional requirements of a potential solution are described. The Chapter is then completed with the description of the overall design of WAIST. The system is detailed in terms of the three functional component applications. These are in turn described in terms of the components that combine to form them. As noted earlier, the design, implementation and evaluation of two major components of the system, namely the pattern recognition and agent-oriented programming integrations, are described in isolation in Chapters 7 and 8.

The implementation of these components are then described along with the rest of the WAIST system in Chapter 9. This chapter provides detail about the operation of all components of
the system and highlights any interaction between components.

Finally the system was evaluated as a whole in Chapter 10. The evaluation was performed using a number of criteria chosen because they represent the core functionality or desirable features of such a system. The methodology used in evaluating each of the criteria and the results achieved were detailed.

The performance of the system in the criteria of timeliness and stop identification was excellent, all locations at which the vehicle was stopped were identified and on average location was reported within 1.7 seconds. The system not only identified all the locations at which the deployment was stopped intentionally, but also every momentary stop in traffic along the routes.

The identification of illegal dumping was similarly successful, as the location at which illegal dumping was simulated was identified. There was however a number of false positives as a result of the accuracy of the classifier which was approximately 90% during the simulation. These false positives could be eliminated manually by comparing the generated event and the full record of all readings after the deployment has finished. The work involved in this process could be mitigated by the addition of intelligent behaviours to the agents which take these situations into consideration.

The final criteria an evaluation of the operation of the agents as a means to intelligently control the application. Within this application the behaviours were designed to control the activation and deactivation of the most energy intensive sensing, in this case the sensing of acceleration. The principle was simple, we do not need to check the sensors if we are in transit. As such the acceleration sensors were only activated when a request was sent to the controlling agent, the SensorManager.

The manager agent received requests under two conditions; a loss of communication with the server or a location reading indicating the deployment was stationary. For every stop the system was activated on average within one second, the same was true of the deactivation when moving. There were only a couple of examples of activations because of connection loss and the system responded in less than a second as above.

In terms of the objectives outlined in Section 1.3, this thesis has achieved all aims through the contents of the thesis.

Ob. 1 was the evaluation of certain pattern recognition techniques when applied to determination of object state. This objective is addressed by the study detailed in Chapter 8. The techniques are described in detail and evaluated in an eight class problem. Results of up to 95.8% are achieved during evaluation.

Ob. 2 was the demonstration of agent-oriented programming languages as a high level abstraction in order to minimise programmer effort. Chapter 7 details the study carried out to assess this. Results show that students completed more tasks when using ASTRA than Java in the same problems.
Finally Ob. 3 was the development and evaluation of a waste tracking application. The design, implementation and evaluation of this system was described in Chapters 6 to 10. The WAIST system addressed all issues of the objective by successfully completing the goals of tracking the location of waste shipments, identifying all stopping locations, highlighting where tampering may have occurred and managing the energy resources of the sensors.
Glossary

Agent-oriented programming  Agent-oriented programming (AOP) is a programming paradigm where the construction of the software is centred on the concept of software agents. These agents act autonomously, and often collaboratively, to achieve their goals. 12

Classification  is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known. 5, 35, 36

Feature  a single representative value calculated from a collection of data. 36

Feature extraction  The process of calculating a feature or set of features based on data. 36

Instance  is a set of features representing single observation or collection of data within the context of classification. 36

Machine Learning  is a scientific discipline that explores the construction and study of algorithms that can learn from data. Such algorithms operate by building a model based on inputs and using that to make predictions or decisions, rather than following only explicitly programmed instructions. 35

Near real-time  in telecommunications and computing, refers to the time delay introduced, by automated data processing or network transmission, between the occurrence of an event and the use of the processed data, such as for display or feedback and control purposes. 3, 4

Pattern recognition  is a branch of machine learning that focuses on the recognition of patterns and regularities in data, although is in some cases considered to be nearly synonymous with machine learning. 1, 5, 7, 8, 12, 35, 42, 43, 130, 145–147

SIXTH  SIXTH is a Java-based middleware for the Sensor Web. It allows sensor-driven applications to be abstracted from the sensors they depend on by providing a unified
interface by which a variety of sensor types can be integrated with the middleware, along with a standardised way for interacting with them.. 3

**Window** is a period of data from a continuous source. Classifiers attempt to classify each window in order to determine what class the window belongs to.. 36, 37

**Wireless sensor networks** A wireless sensor network (WSN) is a collection of spatially distributed autonomous sensors collaborating to monitor physical or environmental conditions, such as temperature, sound, pressure, etc. and to cooperatively pass their data through the network to a main location.. 12
Bibliography


[83] Philip Heck, Jan Bellin, Martin Matousek, Stefan Wonneberger, Ondrej Sychrovsky, Radim Sara, and Markus Maurer. “Collision mitigation for crossing traffic in urban scenarios”. In: Intelligent Vehicles Symposium (IV), 2013 IEEE. IEEE. 2013, pp. 559–566.


[185] Andrea Tabasová, Jií Kropáč, Vít Kermes, Andreja Nemet, and Petr Stehlík. “Waste-
to-energy technologies: Impact on environment”. In: Energy 44.1 (Aug. 2012), pp. 146–
155. issn: 03605442. DOI: 10.1016/j.energy.2012.01.014.

[186] T Tamura, M Sekine, M Ogawa, T Togawa, and Y Fukui. “Classification of accelera-
tion waveforms during walking by wavelet transform”. In: Methods of information in

siting of an incineration plant for municipal solid waste.” In: Waste management (New

collection routes for minimum fuel consumption using 3D GIS modelling.” In: Waste
management (New York, N.Y.) 29.3 (Mar. 2009), pp. 1176–85. issn: 0956-053X. DOI:

460.

[190] Richard Tynan, David Marsh, Donal O’Kane, and G.M.P O’Hare. “Agents for wire-
less sensor network power management”. In: Parallel Processing, 2005. ICPP 2005
Workshops. International Conference Workshops on. June 2005, pp. 413–418. DOI:
10.1109/ICPW.2005.19.

for Wireless Sensor Networks”. In: Proceedings of the Fourth International Joint Con-
ference on Autonomous Agents and Multiagent Systems. AAMAS ’05. New York, NY,
1082682.

lel programming languages”. In: Proceedings Second International Workshop on High-
Level Parallel Programming Models and Supportive Environments. IEEE Computer.
URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=
582951.

“An Algorithm for License Plate Recognition Applied to Intelligent Transportation
System”. In: IEEE Transactions on Intelligent Transportation Systems 12.3 (Sept.

[194] G Werner-Allen, J Johnson, M Ruiz, J Lees, and M Welsh. “Monitoring volcanic eru-
2005.1462003.


Appendix A

AOSE Worksheet: Programming for wireless sensor networks using agents

A.1 Description

This worksheet will require you to develop some agents to achieve some tasks within a piece of software called WAIST: Waste Augmentation and Integrated Shipment Tracking. The project aim is to provide a hardware and software combination that allows for the monitoring and validation of waste in transit in Ireland. The project is built within OSGi and provides support for control and added logic through the use of the CArtAgO interface.

A.2 Problem 1

Trucks within the system have a GPS sensor attached as well as two sensors capable of sensing light and Acceleration that are attached to containers within the truck. Using the acceleration sensors requires a lot of power, so we wish to only utilize them while the truck is stationary. Using the artifacts GPSStreamArtifact, DeploymentArtifact and ReTaskerArtifact write an agent with the capabilities to do the following:

1. Gain access to the sensor readings
2. Maintain the most recently recorded position (as a belief) of each truck (you can assume that the values will be received in chronological order)
3. Notify the system whenever the speed of a truck falls below a certain threshold (1 m/s) by using the operation stopped from GPSStreamArtifact.
4. While the speed remains below the threshold we will consider that the vehicle is stationary, when the speed moves above the threshold value we should notify the system by using the operation stoppedDuration from GPSStreamArtifact.

5. When a truck is first detected as stationary the agent should use the DeploymentArtifact to find the ids of the acceleration sensors and use the RetaskerArtifact to activate the sensors

A.3 Problem 2

WAIST is built upon a component management system, OSGi. OSGi allows the use of bundles (encapsulated as jar files) to provide functionality. Using the operations and signals of the BundleArtifact write an agent with the capabilities to do the following:

1. Gain access to the bundle information
2. Maintain a record of all the bundles currently within the system
3. Based on the bundles available (as jar files), the bundles required for operation, (listed below) and the currently running bundles the agent should install any required bundle when it is available but not already installed.

4. Any bundle that is installed by the agents should also be started.

5. If a bundle becomes available (as a jar) that is of a newer version of a bundle within the system the agent should first stop the older bundle and uninstall it and secondly install the newer version and start it.

The following bundles are required for operation;

- ie.ucd.waist.database.access
- ie.ucd.waist.receiver

A.4 Working with artifacts

A.4.1 Getting access to an artifact

There are three steps required to access an artifact:

1. Creating the artifact, this should either be done by only one agent or attempted within a try recover statement. Alternatively each agent may create their own version of an artifact so long as the name is unique.
where the first parameter is the name we will give the artifact and the second is the full path to the artifact.

2. Lookup the Artifact, ask the system for a reference to the artifact as a variable

```java
CARTAGO.lookupArtifact("bundles", object<ArtifactId> bund);
```

where the first parameter is the name we gave the artifact when it was created and the second is an unbound variable of the type object<ArtifactId>, which will contain the required reference after the statement has been executed. NOTE: you will need to add an import statement at the beginning of the file for the ArtifactId class.

```java
import cartago.ArtifactId;
```

3. Focus on the Artifact, this will give us all signals and observable properties of the Artifact.

```java
CARTAGO.focus(bund);
```

The only parameter is the ArtifactId object you wish to focus on

### A.4.2 Using artifact operations

To use the operations of one of the artifacts easy enough, assuming that the Artifact is already focused all you must do is pass the required parameters in the following form

```java
CARTAGO.nameOfOperation(parameters)
```

```java
CARTAGO.getBundles();
```

If the operation returns a value to the agent through the use of the OpFeedbackParam(T), you must pass in an unbound Variable of the type T for that parameter. Afterwards the return value will be store within the variable.

If the definition shows: getDeploymentForSensor(long id, OpFeedbackParam<Integer> dep), then the code for using the operation would be:
CARTAGO. getDeploymentForSensor(id, int G);

Where id is a value passed and G is the value returned to the agent.

A.5 GPSStreamArtifact

Path: waist.cartago.GPSStreamArtifact

This artifact is responsible for forwarding the GPS data from the system to the agent and system events from the agent to the system.

Operations

• stopped(long id, long t, double lat, double lon)
  – long id - The id of the sensor
  – long t - The time that the reading took place represented as a long
  – double lat The latitude of the truck at the time
  – double lon The longitude of the truck at the time

Stopped generates an event within the WAIST system to signify that a truck was detected as stationary.

• stoppedDuration(long id, long t, double lat, double lon, long duration)
  – long id - The id of the sensor
  – long t - The time that the stationary period began
  – double lat The latitude of the truck at the beginning of the period
  – double lon The longitude of the truck at the beginning of the period
  – long time The duration for which the vehicle was stationary in ms

StoppedDuration generates an event within the WAIST system to signify that a truck was stationary for a period of time.

A.5.1 Signals

These signals are sent as and when they arrive from the waist system. You can assume that the data will arrive in chronological order.
• location(long id, long t, double lat, double lon, float sp)
  – long id - The id of the sensor
  – long t - The time that the reading took place represented as a long
  – double lat The latitude of the truck at the time
  – double lon The longitude of the truck at the time
  – float sp The speed of the truck at the time

A.6 DeploymentArtifact

Path: waist.cartago.DeploymentArtifact

Combinations of sensors are grouped together into deployments, that is a combination of a single GPS sensor and a number of other sensors capable of sensing acceleration and other things. Each deployment is given an identifying number (an integer) and sensors can be associated with only one deployment.

For the purpose of this you may assume that there are exactly 2 sensors as well as the GPS sensor in any deployment.

A.6.1 Operations

• getDeploymentInfo() This operation will cause the generation of the sensor signal described below for each sensor identified within the WAIST system.

• getModalities(long id)
  – long id The id of the sensor you wish to know the modalities of

This operation will cause the generation of the modality signal described below for each modality of the given sensor id.

• getDeploymentForSensor(long id, OpFeedbackParam⟨Integer⟩ dep)
  – long id The id of the sensor we wish to know the deployment of
  – int dep The return value of the type int will contain the deployment id number of the sensor

This operation allow you to query the WAIST system to find what deployment a given sensor belongs to, the value is returned as the second parameter.

• getAccTaskingMessageforDep(long id, OpFeedbackParam⟨TaskingMessage⟩ tm)
  – long id The id of the sensor you wish to activate
object\_TaskingMessage tm The return value of containing the object required to activate the sensor

This operation allows you to generate the default acceleration sensor activation message, which can be used to activate the sensor for a period of 2 min.

A.6.2 Signals

- sensor(long id, int deployment)
  - long id - The id of the sensor
  - int deployment The identifier of the deployment that the sensor belongs to
  
  This signal will be generated for every sensor whenever the getDeploymentInfo operation is called.

- modality(long id, string m)
  - long id - The id of the sensor
  - string m A string containing the modality
  
  This signal will be generated for every modality of a sensor after the use of the getModalities Operation.

A.7 RetaskerArtifact

Path: \textit{ie.ucd.sixth.cartago.RetaskerArtifact}

The WAIST system is build on top of a sensor network middleware named SIXTH, this artifact is a part of SIXTH which allows sensors to be retasked (activated/ sensing frequency altered/ deactivated) by agents. We do not need to worry about how this is achieved, simply use the operation provided.

A.7.1 Operations

- retask(TaskingMessage\textsuperscript{1} t)
  - TaskingMessage t - The tasking message generated earlier which will be used to reconfigure the sensor

  retask passes the TaskingMessage into the middleware in order to activate a sensor.

\textsuperscript{1}The steps involved in obtaining a TaskingMessage object are explained above
BundleArtifact

Path: `ie.ucd.sixth.BundleArtifact` This artifact is designed for managing the operation of the WAIST and SIXTH bundles (and OSGi system bundles too).

Operations

- `getBundles()` This operation will cause the generation of the bundle signal described below for each bundle installed in the system.

- `getJars()` This operation will cause the generation of the jar signal described below for each jar file found within the current working directory.

- `installBundle(String location)`
  - `String location` The path and file name of the jar on the file system
  This operation allows a bundle to be installed into the system based on the location of the jar file.

- `startBundle(object⟨Bundle⟩ b)`
  - `Bundle b` The bundle that you wish to start
  This operation starts the specified bundle.

- `stopBundle(object⟨Bundle⟩ b)`
  - `Bundle b` The bundle that you wish to stop
  This operation stops the specified bundle.

- `uninstallBundle(object⟨Bundle⟩ b)`
  - `Bundle b` The bundle that you wish to uninstall
  This operation uninstalls the specified bundle from the system.

A.7.2 Signals

- `bundle(string sname, object⟨Bundle⟩ b, string version, string state)`
  - `string sname` The symbolic name of the bundle
  - `Bundle b` The bundle object, this is required to perform any operations such as start and stop
  - `string version` The version of the bundle represented as a string, this will traditionally be of the form “.0.3” but may also be in a longer version “2.0.3.v20140305”
string state  The state of the bundle when the signal was sent. This can be any
one of the following “uninstalled”, “installed”, “resolved”, “starting”, “stopping”
or “active”.

This signal has two possible sources; the first is as a direct result of the getBundles
operation at which point a signal will be sent for every bundle within the system. If
the operation is repeated you will receive the same signals. The second source is a
notification after a bundle has changed state, these may occur any of the operations
such as startBundle, stopBundle or uninstallBundle are used. You may assume that no
software will be affecting the bundles.

jar(string fname, string sname, string version)

- string fname  The file name and location of the jar file
- string sname  The symbolic name of the bundle
- string version  The version of the bundle represented as a string

This signal is generated by the operation getJars only and will send a signal for every
jar file in the current directory.
Appendix B

AOSE Worksheet: Programming for wireless sensor networks using Java

B.1 Description

This worksheet will require you to develop some classes in order to achieve some tasks within a piece of software called WAIST: Waste Augmentation and Integrated Shipment Tracking. The project aim is to provide a hardware and software combination that allows for the monitoring and validation of waste in transit in Ireland. The project is built within OSGi and additional functionality has been provided to support development (described below).

You will need to install the two parts of the SIXTH core to develop using eclipse the eclipse update site is (http://sixth.ucd.ie/eclipse)

B.2 Problem 1

Trucks within the system have a GPS sensor attached as well as two sensors capable of sensing light and Acceleration that are attached to containers within the truck. Using the acceleration sensors requires a lot of power, so we wish to only utilize them while the truck is stationary. Using functionality described below for the OSGi and SIXTH systems write code to complete the following requirements.

1. Gain access to the sensor readings
2. Maintain the most recently recorded position (as a belief) of each truck (you can assume that the values will be received in chronological order)
3. Notify the system whenever the speed of a truck falls below a certain threshold (1 m/s) by using the static method stopped from the class SystemEventManager.
4. While the speed remains below the threshold we will consider that the vehicle is stationary, when the speed moves above the threshold value we should notify the system by using the static method stoppedDuration from the class SystemEventManager.

5. When a truck is first detected as stationary the agent should use the DeploymentManager to find the ids of the acceleration sensors and use the RetaskingService to activate the sensors.

B.3 Problem 2

WAIST is built upon a component management system, OSGi. OSGi allows the use of bundles (encapsulated as jar files) to provide functionality. Using Bundle operations described below and some File operations complete the following tasks:

1. Gain access to the bundle information
2. Maintain a record of all the bundles currently within the system
3. Based on the bundles available (as jar files), the bundles required for operation, (listed below) and the currently running bundles you should install any required bundle when it is available but not already installed.
4. Any bundle that you installed should also be started.
5. If a bundle becomes available (as a jar) that is of a newer version of a bundle within the system you should first stop the older bundle and uninstall it and secondly install the newer version and start it.

The following bundles are required for operation:

- ie.ucd.waist.database.access
- ie.ucd.waist.receiver

B.4 SIXTH, WAIST and OSGi functionality

B.4.1 Getting data

In order to get data from the system we must register our interest in this data with the data broker of the SIXTH middleware, it will then pass us all data we have the credentials to see.

In order to receive the data we must implement the interface IQueryDataReceiver, this will require implementing the following methods getCredentials, unregister, getIdentity, and two types of receive methods. Code for some of the methods is given below, you are only required to implement a constructor which will ask for the data by passing a query to the middleware,
a method named receive that will receive the data and a method called unregister that will remove the query from the system when called.

Creating the artifact, this should either be done by only one agent or attempted within a try recover statement. Alternatively each agent may create their own version of an artifact so long as the name is unique. SIXTH uses queries to determine what data will be forwarded to you. For our purposes we will only be using one type of query, a ModalityQuery, this filters all of the data that is received for a single modality such as "GPS" or "Acceleration".

A ModalityQuery can be constructed by passing the parameter of a Modality to the constructor, and the Modality constructor only requires a String representing the modality as a parameter. Registering for the data requires access to the SIXTH DataBroker, this can be accessed statically using the following code. This object is used to both register our interest as well as unregistering it through the use of the methods registerInterest and unregisterInterest respectively. The first method requires an IQuery and an IQueryReceiver as parameters and the latter requires only the original IQuery used when registering.

```
public class GPSIntel implements IQueryDataReceiver {
    private UUID id = UUID.randomUUID();

    public Credentials getCredentials() {
        return new Credentials(GPSIntel.class.getName(), "IQueryDataReceiver", id);
    }

    public String getIdentity() {
        return GPSIntel.class.getName();
    }

    public void receive(String arg0, IQuery arg1) {}}
```

```
ModalityQuery query = new ModalityQuery(new Modality("GPS"));
```

```
SIXTHMonitor.getDiscovery(getCredentials()).getDataBroker()
```

### B.4.2 Converting Data

Once registered, the receive method in the class will be used as a callback providing an ISensorData object which contains our data. When reading GPS Data the object should be cast to a LocationData object. The LocationData object contains the following methods;
• getID() returns the id of the sensor as a long
• getLatitude() returns the latitude of the reading
• getLongitude() returns the latitude of the reading
• getVelocity() returns the speed of the vehicle
• getTimestamp returns the time the measurement was taken

### B.4.3 Getting Deployment Information

In order to perform some functions you will require knowing which sensors are grouped together in deployments. This information is provided using the static class DeploymentManager;

• getSensors() returns a list of the sensor ids as longs
• getModalities(long id) returns a list of the modalities as strings for the given sensor id
• getDeploymentForSensor(long id) returns the deployment number of the given sensor id
• getIdsForDeployment(int dep) returns a list of the sensor ids for the given deployment number
• getAccTaskingMessageForDep(long id) returns a TaskingMessage configured to activate the acceleration sensors for the given sensor id.

### B.5 Sending WAIST System Events

The work carried out by this component is to inform other portions of the system, keeping a record of all events and where necessary displaying the events in a visual representation for the users. This is done using static methods from the SystemEventManager class. The events are;

• sendDepStartEvent(int dep) propagates into the WAIST system an event stating that the given deployment was activated
• stopped(long id, long t, double lat, double lon) propagates into the WAIST system an event noting the location, time and id of a truck that was believed to be stationary
• stoppedDuration(long id, long t, double lat, double lon, long duration) propagates into the WAIST system an event noting the location, start time, duration and id of a truck that was stationary for a period of time

In some cases we will require pushing events back into the system to cause changes. On such event is the use of tasking messages to activate sensors. To do this we need access to
the RetaskingService, which is accessed in SIXTH in the same way as the DataBroker. This

```
SIXTHMonitor.getDiscovery(getCredentials()).getRetaskingService()
```

service provides the method task, which accepts a TaskingMessage as parameter.

## Accessing Bundle Information

In order to make any changes to the OSGi component system you must first have access to
the bundles that exist in that system.

### Accessing Existing bundles

In order to get access to the bundle lists we must first get access to the BundleContext, this can
be passed as an argument from the activator or accessed by means of the following code. You
pass the a class as a parameter and you will get the BundleContext object associated with that
class in this example I have simply used the class that the code was in named BundleManager.
You can then use the method getBundles() which returns an Array of Bundle objects. Bundle

```
BundleContext bc = FrameworkUtil.getBundle(BundleManager.class).
            getBundleContext();

Bundle[] bundles = bc.getBundles();
```

objects contain a number of methods of use, particularly we will make use of the following;

- `getSymbolicName()` returns the name of the bundle (generally given in reverse domain
  naming scheme) without the version number or file type

- `getVersion()` Returns a Version object representing the version number of the bundle,
  can be converted to a string using `toString` method

- `getState()` returns an integer representation of the current state of the bundle. These
  values are stored in the Bundle class as constants UNINSTALLED, INSTALLED, RESOLVED,
  STARTING, STARTED, STOPPED, STOPPING, ACTIVE

The following methods in the bundle class are used to control the operation of the bundles;

- `start()` this will start the bundle, throws BundleException

179
• stop() this will stop the bundle, throws BundleException

• uninstall() this will remove the bundle from the system

Installing a bundle

In order to install a bundle you require access to the BundleContext, this contains a method called installBundle. installBundle requires a String as a parameter representing the source and protocol of the Bundle. We will assume only use of the local file system to install bundles, as such the absolute or relative path to the jar file must be prepended with "file:". This can also throw a BundleException.
Appendix C

AOSE Class Survey

C.1 Questions

C.1.1 Personal Information

1. Please describe your experience with programming. Ideally the number of years within the industry or since graduation. ____________

2. Please list the programming languages you have experience using. ______________

3. What is your level of experience with the Java programming language?

4. What is your level of experience with OSGi?

5. What is your level of experience with event-driven programming?

6. What is your level of experience with distributed systems?

<table>
<thead>
<tr>
<th></th>
<th>No Experience</th>
<th>Little Experience</th>
<th>Moderate Experience</th>
<th>Very Experienced</th>
<th>Extremely Experienced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C.1.2 Agents in General

Please indicate your level of agreement with the following statements.

1. Agents are a useful level of abstraction.

2. I would consider using an AOP language in my (future) work.
3. AOP languages make distributed programming easier.

4. AOP languages make concurrent programming easier.

5. Studying AOP languages enhanced my understanding of distributed computing

<table>
<thead>
<tr>
<th></th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neither Agree nor Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

C.1.3 ASTRA

Please indicate your level of agreement with the following statements.

1. Static typing, and the verification this enables, are important.

2. Static typing is a necessary feature of AOP languages.

3. Static typing makes ASTRA code easy to read.

4. ASTRA was easy to learn.

5. I found it easy to apply my existing programming knowledge to ASTRA.

6. The syntax of ASTRA made it easier to understand.

7. There is a steep learning curve for ASTRA.

8. The lack of a debugger made ASTRA more difficult to learn.

9. ASTRA offers a good level of abstraction for programming distributed systems.

<table>
<thead>
<tr>
<th></th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neither Agree nor Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
C.1.4 WAIST

Please indicate your level of agreement with the following statements.

1. The biggest problem I faced while completing the assignment was understanding

<table>
<thead>
<tr>
<th>SIXTH</th>
<th>OSGi</th>
<th>ASTRA</th>
<th>The Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

2. The supporting information for the ASTRA problem was adequate to solve the problem.
3. The supporting information for the Java problem was adequate to solve the problem.
4. Using ASTRA made the assignment easier to complete.
5. Using ASTRA code was simpler than Java code.
6. Using ASTRA code was easier to modify than Java code.
7. My ASTRA solution was more readable than my Java solution.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neither Agree nor Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Q3</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Q4</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Q5</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Q6</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>Q7</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

C.2 Results

C.2.1 Personal Information

1. Please describe your experience with programming. Ideally the number of years within the industry or since graduation.

   Min  | 1  
   Max  | 20 
   Mean | 7.65 
   SD   | 5.12 

2. Please list the programming languages you have experience using.
3. What is your level of experience with the Java programming Language?

4. What is your level of experience with OSGi?

5. What is your level of experience with event-driven programming?

6. What is your level of experience with distributed systems?

<table>
<thead>
<tr>
<th></th>
<th>No Experience</th>
<th>Little Experience</th>
<th>Moderate Experience</th>
<th>Very Experienced</th>
<th>Extremely Experienced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3</td>
<td>5%</td>
<td>15%</td>
<td>35%</td>
<td>30%</td>
<td>15%</td>
</tr>
<tr>
<td>Q4</td>
<td>85%</td>
<td>10%</td>
<td>5%</td>
<td>5%</td>
<td>0%</td>
</tr>
<tr>
<td>Q5</td>
<td>5%</td>
<td>40%</td>
<td>40%</td>
<td>40%</td>
<td>5%</td>
</tr>
<tr>
<td>Q6</td>
<td>15%</td>
<td>25%</td>
<td>40%</td>
<td>40%</td>
<td>0%</td>
</tr>
</tbody>
</table>

C.2.2 Agents in General

1. Agents are a useful level of abstraction.
2. I would consider using an AOP language in my (future) work.

3. AOP languages make distributed programming easier.

4. AOP languages make concurrent programming easier.

5. Studying AOP languages enhanced my understanding of distributed computing

<table>
<thead>
<tr>
<th></th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neither Agree nor Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0%</td>
<td>0%</td>
<td>20%</td>
<td>80%</td>
<td>0%</td>
</tr>
<tr>
<td>Q2</td>
<td>0%</td>
<td>15%</td>
<td>45%</td>
<td>40%</td>
<td>0%</td>
</tr>
<tr>
<td>Q3</td>
<td>0%</td>
<td>10%</td>
<td>55%</td>
<td>30%</td>
<td>5%</td>
</tr>
<tr>
<td>Q4</td>
<td>0%</td>
<td>15%</td>
<td>65%</td>
<td>20%</td>
<td>0%</td>
</tr>
<tr>
<td>Q5</td>
<td>5%</td>
<td>5%</td>
<td>30%</td>
<td>55%</td>
<td>5%</td>
</tr>
</tbody>
</table>

C.2.3 ASTRA

1. Static typing, and the verification this enables, are important.

2. Static typing is a necessary feature of AOP languages.

3. Static typing makes ASTRA code easy to read.

4. ASTRA was easy to learn.

5. I found it easy to apply my existing programming knowledge to ASTRA.

6. The syntax of ASTRA made it easier to understand.

7. There is a steep learning curve for ASTRA.
8. The lack of a debugger made ASTRA more difficult to learn.

9. ASTRA offers a good level of abstraction for programming distributed systems.

<table>
<thead>
<tr>
<th>Question</th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neither Agree nor Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0%</td>
<td>5%</td>
<td>20%</td>
<td>60%</td>
<td>15%</td>
</tr>
<tr>
<td>Q2</td>
<td>0%</td>
<td>20%</td>
<td>40%</td>
<td>40%</td>
<td>0%</td>
</tr>
<tr>
<td>Q3</td>
<td>0%</td>
<td>0%</td>
<td>35%</td>
<td>55%</td>
<td>10%</td>
</tr>
<tr>
<td>Q4</td>
<td>0%</td>
<td>10%</td>
<td>30%</td>
<td>50%</td>
<td>10%</td>
</tr>
<tr>
<td>Q5</td>
<td>0%</td>
<td>20%</td>
<td>30%</td>
<td>45%</td>
<td>5%</td>
</tr>
<tr>
<td>Q6</td>
<td>0%</td>
<td>20%</td>
<td>15%</td>
<td>65%</td>
<td>0%</td>
</tr>
<tr>
<td>Q7</td>
<td>0%</td>
<td>65%</td>
<td>10%</td>
<td>20%</td>
<td>5%</td>
</tr>
<tr>
<td>Q8</td>
<td>5%</td>
<td>15%</td>
<td>5%</td>
<td>50%</td>
<td>25%</td>
</tr>
<tr>
<td>Q9</td>
<td>5%</td>
<td>5%</td>
<td>20%</td>
<td>55%</td>
<td>15%</td>
</tr>
</tbody>
</table>

### C.2.4 WAIST

1. The biggest problem I faced while completing the assignment was understanding SIXTH

<table>
<thead>
<tr>
<th></th>
<th>SIXTH</th>
<th>OSGi</th>
<th>ASTRA</th>
<th>The Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>25%</td>
<td>20%</td>
<td>30%</td>
<td>25%</td>
</tr>
</tbody>
</table>

2. The supporting information for the ASTRA problem was adequate to solve the problem.

3. The supporting information for the Java problem was adequate to solve the problem.

4. Using ASTRA made the assignment easier to complete.

5. Using ASTRA code was simpler than Java code.

6. Using ASTRA code was easier to modify than Java code.
7. My ASTRA solution was more readable than my Java solution.

<table>
<thead>
<tr>
<th></th>
<th>Strongly Disagree</th>
<th>Disagree</th>
<th>Neither Agree nor Disagree</th>
<th>Agree</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>0%</td>
<td>35%</td>
<td>20%</td>
<td>45%</td>
<td>0%</td>
</tr>
<tr>
<td>Q3</td>
<td>10%</td>
<td>30%</td>
<td>20%</td>
<td>40%</td>
<td>0%</td>
</tr>
<tr>
<td>Q4</td>
<td>0%</td>
<td>10%</td>
<td>30%</td>
<td>45%</td>
<td>15%</td>
</tr>
<tr>
<td>Q5</td>
<td>0%</td>
<td>30%</td>
<td>30%</td>
<td>30%</td>
<td>10%</td>
</tr>
<tr>
<td>Q6</td>
<td>0%</td>
<td>35%</td>
<td>45%</td>
<td>20%</td>
<td>0%</td>
</tr>
<tr>
<td>Q7</td>
<td>0%</td>
<td>20%</td>
<td>30%</td>
<td>35%</td>
<td>15%</td>
</tr>
</tbody>
</table>