Framework for High Level Programming of Wireless Sensor and Actuator Networks

Alia Ibrahim

Submitted for the Degree of Doctor of Philosophy from the University of Surrey

Institute for Communication Systems
School of Electronics and Physical Sciences University of Surrey Guildford, Surrey GU2 7XH, UK

August 2015

© Alia Ibrahim 2015
Summary

The vision of ubiquitous computing (UbiComp), as introduced by Mark Weiser, goes beyond simple embedding of computational capabilities in the environment. The vision foresees the way humans interact with technology in a very different way; the future of man-machine interaction will not be through clicking and typing on computers but through interacting with objects around us through speech, touch, gestures or movement, etc.; any required computation will be performed in the background, hidden from the user.

Since Weiser first introduced his idea, technology has been progressing making essential bricks of such an environment available and successful research projects have been developed; however, the UbiComp vision is still far from current reality. This is partly due to the fact that technology is still too visible, and partly due to the complexity of the systems themselves but even more so due to the complexity of setting up and handling both systems and services.

Most users of UbiComp applications can be expected to be technology agnostic, they don’t have technical background and their knowledge of the programming languages that are typically used to set up UbiComp applications is either very limited or does not exist. This has so far limited building smart environment applications to professional developers. At the same time, the actual users of UbiComp have typically rather individual requirements depending on their intended uses and environment; they tend to define tasks in their own way to take control over their environment. Thus there is a need to move the customisation functionality of applications to the users’ side of the process. This entails simplifying the way service are defined and customised by making discovery, operational modification and provisioning transparent to the user.

This thesis presents a top down approach providing a high level UbiComp programing language that aims at empowering end-users with the ability to control their environment without needing understanding about the underlying technologies. The work defined and investigated a set of algorithms for mapping users’ service requirements to available assets in the network and proved that they outperform SOTA in resource allocation. The here defined programming language builds on this mapping and hides away the complexity of the system from its users. The usability of the language was verified using “cognitive dimensions”, formal semantics was used for verifying the language and connecting it to the mapping algorithms.

Keywords: Ubiquitous Computing, Wireless Sensor Networks, Mission Assignment, Spatial Reasoning.

Email: alia.ibrahim@surrey.ac.uk, WWW: http://www.surrey.ac.uk/
Acknowledgments

First of all I would like to deeply thank my supervisors, Prof. Klaus Moessner and Prof. Rahim Tafazolli for providing me with clear directions and valuable feedback towards improving my research. I would also like to thank Dr. Francois Carrez for his valuable comments during the course of my PhD.

Spatial thanks to all my friends in ICS, Safaa Sway, Argyris Kyrgiazos, Riccardo Pozza, Hayeoung Lee, Basil Elmasri, who made these years fun and enjoyable.

I would like to deeply thank my family: my mom, dad, sister and brother, without their support and encouragement along the way none of this would have been possible.

Finally I would like to thank my loving husband, Rabie, for his care, encouragement, and for standing by my side throughout my studies.
Contents

Summary ........................................................................................................................................... ii
Acknowledgments ............................................................................................................................... iii
Contents ................................................................................................................................................ iv
List of Figures ...................................................................................................................................... viii
List of Tables ....................................................................................................................................... x
Glossary of Terms ............................................................................................................................... xi

1 Introduction....................................................................................................................................... 1
  1.1 Characteristics of UbiComp ......................................................................................................... 2
  1.2 Existing UbiComp Initiatives ........................................................................................................ 4
  1.3 Motivation .................................................................................................................................... 5
  1.4 Research Problem ....................................................................................................................... 6
  1.5 Research Challenges ................................................................................................................... 6
  1.6 Objectives .................................................................................................................................... 7
  1.7 Why Wireless Sensor & Actuator Networks? ............................................................................. 8
  1.8 Overview of Contributions .......................................................................................................... 8
  1.9 Research Methodology .............................................................................................................. 10
  1.10 Publications ............................................................................................................................... 10
  1.11 Structure of the Thesis .............................................................................................................. 11

2 Literature Review on High-level Programming tools and Modelling Techniques for UbiComp................................................................................................................................. 14
  2.1 High-level Programming for UbiComp and Wireless Sensor Networks .............................. 15
    2.1.1 The Need for End-user Development Tools ......................................................................... 15
    2.1.2 Features of End-User Development Tools ......................................................................... 16
    2.1.3 User-centric Programming for Wireless Sensor Networks ............................................. 18
    2.1.4 Tools for End-User Development in Ubiquitous Computing ......................................... 19
      2.1.4.1 Target Domains .................................................................................................................. 19
      2.1.4.2 Design Approach .............................................................................................................. 20
      2.1.4.3 Development Interface Paradigm ..................................................................................... 21
    2.1.5 Discussion ............................................................................................................................... 25
## Contents

2.2 Task Allocation for Wireless Sensor Networks .......................................................... 26  
  2.2.1 Coverage-oriented Sensor Selection ................................................................. 27  
  2.2.2 Object Tracking and Localisation ................................................................. 27  
  2.2.3 Sensor Mission Assignment as an “Application Layer” Task ....................... 28  
    2.2.3.1 Weapon Target Assignment (WTA) ...................................................... 28  
  2.2.4 Summary .......................................................................................................... 33  
2.3 Formal Modelling of UbiComp and Its Enabling Technology ............................................. 34  
  2.3.1 The Need for Modelling Ubiquitous Computing and Its Technology ........ 34  
  2.3.2 Ontology for Modelling UbiComp and Its Enabling Technology ..................... 35  
    2.3.2.1 Ontology for Wireless Sensor Networks ................................................. 35  
    2.3.2.2 Ontologies for Ubiquitous Computing .................................................... 37  
  2.3.3 Discussion ........................................................................................................ 38  
  2.3.4 Spatial Reasoning ............................................................................................. 38  
    2.3.4.1 GeoSPARQL ............................................................................................. 39  
2.4 Discussion .................................................................................................................. 39  
3 Proposed Architecture for End-User Development Tools ...................................................... 42  
  3.1 Proposed Architecture ............................................................................................. 43  
    3.1.1 Design Goals ................................................................................................... 43  
  3.2 System Components and Architecture ...................................................................... 45  
  3.3 System Operation Steps .......................................................................................... 48  
  3.4 Conclusion ............................................................................................................. 51  
4 Spatial Reasoning for Wireless Sensor and Actuator Network Mission Assignment .... 52  
  4.1 Requirements for Sensor Mission Assignment .......................................................... 53  
    4.1.1 Spatial Reasoning with GeoSPARQL ............................................................. 54  
    4.1.2 Semantic Sensor Network Ontology (SSN) ..................................................... 56  
  4.2 Ontologies for Semantic Spatial Reasoning with WSN .............................................. 57  
  4.3 Spatial Reasoning Approach .................................................................................. 59  
    4.3.1 Spatial Matching of Tasks and Assets .............................................................. 59  
    4.3.2 Geospatial Sensor Mission Assignment Algorithm (GeoSMA) .................... 61  
      4.3.2.1 Algorithm Evaluation ............................................................................. 64  
    4.3.3 Selfish Geospatial Sensor Mission Assignment algorithm (SGeoSMA) ....... 72  
      4.3.3.1 The Selfish Gene .................................................................................... 73  
      4.3.3.2 Algorithm Implementation ................................................................. 75  
      4.3.3.3 Algorithm Evaluation ........................................................................... 76  
  4.4 Conclusion ............................................................................................................. 82
5 Spatial Weapon Target Assignment for Mission Assignment in Wireless Sensor &
Actuator Networks ...................................................... ................................................... 84

5.1 The Weapon Target Assignment Problem .................................................. 85
  5.1.1 Weapon Target Assignment Problem Formulation ........................................... 85

5.2 Weapon Target Assignment Approach to Mission Assignment in Wireless Sensor
Networks .................................................................. ................................................... 86
  5.2.1 The Selfish Gene Algorithm for Multiple Task Allocation ................................. 88
  5.2.1.2 Implementation ........................................................ ....................... 89
  5.2.1.3 Simulation ........................................................ ......................... 90

5.3 Multi-Objective Optimization for Sensor Mission Assignment ....................... 95
  5.3.1 Bi-level Optimisation Problems .................................................................... 97
    5.3.1.1 Bi-level Sensor Task Assignment ........................................................ 97
    5.3.1.2 Genetic Algorithms with Local Search ............................................... 99
    5.3.1.3 Sensor Detection Probability ........................................................... 100

5.3.2 Geospatial Bi-level Selfish Algorithm for Sensor Mission Assignment Algorithm
(BiSMA)........................................................ ................................................... ........ 102
  5.3.2.1 Validation ................................................................................................ 105

5.4 Conclusion ........................................................................................................... 107

6 High Level Programming Language for Wireless Sensor/Actuator Networks .......... 109

  6.1 Environment Programming Language ............................................................ 111
    6.1.1 Overview .................................................................................................. 111
    6.1.2 Problem Domain ...................................................................................... 112
      6.1.2.1 User Interface ...................................................................................... 112
      6.1.2.2 Ontology Models ................................................................................ 113
      6.1.2.3 Allocation Layer ................................................................................ 114
      6.1.2.4 Formal Abstract Syntax ...................................................................... 115
      6.1.2.5 EPL Dynamic Semantics .................................................................. 118

  6.2 EPL Concrete Syntax ...................................................................................... 120
    6.2.1 EPL Use-Scenarios .................................................................................... 123

6.3 Evaluation ............................................................................................................. 125

  6.3.1 EPL Formal Semantics Evaluation .............................................................. 125
  6.3.2 EPL Usability Evaluation ............................................................................. 125
    6.3.3 Cognitive Dimensions .............................................................................. 127
      6.3.3.1 Abstraction ........................................................................................ 127
      6.3.3.2 Closeness of Mapping .................................................................... 128
Contents

6.3.3.3 Consistency ......................................................................................... 129
6.3.3.4 Viscosity ............................................................................................ 129
6.3.3.5 Visibility & Juxtaposability ................................................................. 131
6.3.3.6 Hidden Dependencies ........................................................................ 132
6.3.3.7 Progressive Evaluation ...................................................................... 133
6.3.3.8 Premature Commitment and Enforced Look-ahead ......................... 133
6.3.3.9 Secondary Notation ........................................................................... 133
6.3.3.10 Discussion ......................................................................................... 134

6.4 Conclusion ............................................................................................... 134

7 Conclusion and Future Directions ............................................................... 136

7.1 Conclusion ............................................................................................... 136
7.2 Future work ............................................................................................... 139
  7.2.1 Testing ................................................................................................. 139
  7.2.2 Resource Allocation ............................................................................. 139
    7.2.2.1 Spatial Task Allocation .................................................................. 139
    7.2.2.2 Additional Factors ....................................................................... 140
  7.2.3 Time Integration .................................................................................. 141

Bibliography ................................................................................................. 142
List of Figures

Figure 2-1 Trade-off between cost of learning and expressiveness in programming languages [59] ................................................................. 17
Figure 2-2 Temperature Monitoring Application written with the SEAL language ......................... 19
Figure 2-3 Classification of EUD Tools .................................................................................. 20
Figure 2-4 Genetic Algorithm Operation ................................................................................ 31
Figure 3-1 System Architecture ........................................................................................... 47
Figure 3-2 System Processes and Interactions ....................................................................... 48
Figure 4-1 GeoSPARQL Ontology Main Classes [43] ............................................................. 54
Figure 4-2 GeoSPARQL Geometry Hierarchy [43] .................................................................. 55
Figure 4-3 Stimulus-Sensor-Observation SSN [139] ............................................................... 56
Figure 4-4 Ontologies for SMA ............................................................................................ 57
Figure 4-5 Sensing Range Specification ............................................................................... 59
Figure 4-6 Basic Spatial Matching Relations found in SF ontology ........................................ 60
Figure 4-7 Geospatial Sensor Mission Assignment Algorithm ................................................ 64
Figure 4-8 Effect of node’s density ....................................................................................... 66
Figure 4-9 Task Allocation Success Rate with Changing Node Density ................................... 67
Figure 4-10 Effect of Sensing Range ..................................................................................... 68
Figure 4-11 Task Allocation Success Rate with Changing Node Sensing Range ......................... 68
Figure 4-12 Effect of Task Radius ......................................................................................... 70
Figure 4-13 A Mission and its Solutions ............................................................................... 71
Figure 4-14 Sensor Description in Ontology ......................................................................... 71
Figure 4-15 Mission and its Solutions .................................................................................... 72
Figure 4-16 Selfish Gene Individual Selection ....................................................................... 74
Figure 4-17 Selfish Gene ....................................................................................................... 75
Figure 4-18 SGeoSMA Protocol ........................................................................................... 76
Figure 4-19 Effect of Number of Sensors on Computation Time ............................................ 79
Figure 4-20 Effect of Number of Sensors on Computation Time (zooming at the small numbers) ......................................................................................... 79
Figure 4-21 Allocation Algorithms Comparison with different Solution Space ......................... 80
Figure 5-1 Sensor Mission Assignment Problem ..................................................................... 85
Figure 5-2 Multi-Star Selfish Gene Operation ......................................................................... 89
Figure 5-3 Fitness Curve for S = 50, T = 50. Results are averaged over 10 Trials ..................... 94
Figure 5-4 Fitness Curve for S = 20, T = 20. Results are averaged over 10 Trials ..................... 94
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-5</td>
<td>SSN Energy Module [138]</td>
</tr>
<tr>
<td>5-6</td>
<td>BiSMA Flowchart</td>
</tr>
<tr>
<td>5-7</td>
<td>BiSMA procedure</td>
</tr>
<tr>
<td>5-8</td>
<td>Tasks and Network Fitness</td>
</tr>
<tr>
<td>5-9</td>
<td>Number of Assigned sensors</td>
</tr>
<tr>
<td>6-1</td>
<td>System Architecture</td>
</tr>
<tr>
<td>6-2</td>
<td>Temperature Monitoring Application</td>
</tr>
<tr>
<td>6-3</td>
<td>Defining new Property with EPL</td>
</tr>
<tr>
<td>6-4</td>
<td>CAMP Application. To change time changing the magnate is simple</td>
</tr>
<tr>
<td>6-5</td>
<td>ESPranto Application development</td>
</tr>
<tr>
<td>6-6</td>
<td>Application Development using EPL</td>
</tr>
<tr>
<td>6-7</td>
<td>CAMP Visibility</td>
</tr>
</tbody>
</table>
List of Tables

Table 1-1 Mapping the Challenges to Objectives and Technical Contributions ........................................ 10
Table 2-1 SOTA gaps, Challenges and Proposed technology for UbiComp Systems .................................. 41
Table 4-1 Simulation Parameters for Density Scenario ................................................................................... 66
Table 4-2 Simulation Parameter for Sensing Range Scenario ........................................................................ 67
Table 4-3 Simulation Parameter for Task Region Scenario ............................................................................. 69
Table 4-4 Simulation Results - Node Density Effect ......................................................................................... 77
Table 4-5 Simulation Results - Sensing Range Effect ....................................................................................... 78
Table 4-6 Simulation Results - Task Region Effect ......................................................................................... 81
Table 5-1 Simulation Results for Randomized Data Scenario. Results are averaged over 10 trials .......... 92
Table 5-2 Average Fitness Value after One Hour Run Results are Averaged over 10 Trials ...................... 93
Table 5-3 Simulation Results - Network Fitness ............................................................................................... 107
Table 6-1 Formal Abstract Syntax .................................................................................................................. 118
Table 6-2 Dynamic Semantics ....................................................................................................................... 120
Table 6-3 Selected Commands of The Language ............................................................................................ 124
# Glossary of Terms

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAL</td>
<td>Ambient Assisted Living</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>ANTLR</td>
<td>ANother Tool for Language Recognition</td>
</tr>
<tr>
<td>ASP</td>
<td>Active Sensor Processes</td>
</tr>
<tr>
<td>BB</td>
<td>Branch and Bound</td>
</tr>
<tr>
<td>BiSMA</td>
<td>Bi-level Selfish Mission Assignment</td>
</tr>
<tr>
<td>CAC</td>
<td>Context-Aware Calculus</td>
</tr>
<tr>
<td>CAMP</td>
<td>Capture and Access Magnetic Poetry</td>
</tr>
<tr>
<td>CCA</td>
<td>CCA: a Calculus of Context-aware Ambients</td>
</tr>
<tr>
<td>CDs</td>
<td>Cognitive Dimensions of notations</td>
</tr>
<tr>
<td>CMAN</td>
<td>Calculus for Mobile Ad hoc Networks</td>
</tr>
<tr>
<td>COBRA-ONT</td>
<td>Context Broker Architecture Ontology</td>
</tr>
<tr>
<td>CSN</td>
<td>Calculus for Sensor Networks</td>
</tr>
<tr>
<td>CWS</td>
<td>Calculus for Wireless Systems</td>
</tr>
<tr>
<td>DODE</td>
<td>Domain-Oriented Design Environments</td>
</tr>
<tr>
<td>DOLCE</td>
<td>Descriptive Ontology for Linguistic and Cognitive Engineering</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language</td>
</tr>
<tr>
<td>DWTA</td>
<td>Dynamic Weapon Target Assignment</td>
</tr>
<tr>
<td>EE</td>
<td>Energy Efficiency</td>
</tr>
<tr>
<td>EMO</td>
<td>Evolutionary Multi-objective Optimization</td>
</tr>
<tr>
<td>EO</td>
<td>Evolutionary Optimization</td>
</tr>
<tr>
<td>EPO</td>
<td>Extension Plugin Ontology</td>
</tr>
<tr>
<td>ESP</td>
<td>Edutainment Sensor Platform</td>
</tr>
<tr>
<td>EUD</td>
<td>End User Development</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>EUP</td>
<td>End User Programming</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>GML</td>
<td>Geographic Mark-up Language</td>
</tr>
<tr>
<td>HBI</td>
<td>Human Building Interaction</td>
</tr>
<tr>
<td>HCI</td>
<td>Human Computer Interaction</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communications Technology</td>
</tr>
<tr>
<td>IFC</td>
<td>Industry Foundation Framework</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>ISR</td>
<td>Intelligence, Surveillance and Reconnaissance</td>
</tr>
<tr>
<td>MANET</td>
<td>Mobile Ad hoc Networks</td>
</tr>
<tr>
<td>MBR</td>
<td>Minimum Bounding Rectangle</td>
</tr>
<tr>
<td>MOOP</td>
<td>Multiple Objective Optimization Problem</td>
</tr>
<tr>
<td>OGC</td>
<td>Open Geospatial Consortium</td>
</tr>
<tr>
<td>OPC</td>
<td>One Point Crossover</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>PBD</td>
<td>Programming By Demonstration</td>
</tr>
<tr>
<td>PBE</td>
<td>Programming By Example</td>
</tr>
<tr>
<td>RCC</td>
<td>Region Connection Calculus</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RFID</td>
<td>Radio Frequency Identification</td>
</tr>
<tr>
<td>SDK</td>
<td>Software development kit</td>
</tr>
<tr>
<td>SDO</td>
<td>Sensor Data Ontology</td>
</tr>
<tr>
<td>SensorML</td>
<td>Sensor Model Language</td>
</tr>
<tr>
<td>SF</td>
<td>Simple Feature</td>
</tr>
<tr>
<td>SG</td>
<td>Selfish Gene</td>
</tr>
<tr>
<td>SGeoSMA</td>
<td>Selfish Geospatial Sensor Mission Assignment</td>
</tr>
<tr>
<td>SHO</td>
<td>Sensor Hierarchy Ontology</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>SMA</td>
<td>Sensor Mission Assignment</td>
</tr>
<tr>
<td>SOUPA</td>
<td>Standard Ontology for Ubiquitous and Pervasive Applications</td>
</tr>
<tr>
<td>SSN</td>
<td>Semantic Sensor Network</td>
</tr>
<tr>
<td>SUMO</td>
<td>Suggested Upper Merged Ontology</td>
</tr>
<tr>
<td>SWE</td>
<td>Sensor Web Enablement</td>
</tr>
<tr>
<td>SWTA</td>
<td>Static Weapon Target Assignment</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
</tr>
<tr>
<td>UbiComp</td>
<td>Ubiquitous Computing</td>
</tr>
<tr>
<td>UE</td>
<td>User Experience evaluation</td>
</tr>
<tr>
<td>UEA</td>
<td>User Experience Assessment</td>
</tr>
<tr>
<td>UGV</td>
<td>Unmanned Ground Vehicle</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>VP</td>
<td>Virtual Population</td>
</tr>
<tr>
<td>VPL</td>
<td>Visual Programming Language</td>
</tr>
<tr>
<td>WKT</td>
<td>Well Known Text</td>
</tr>
<tr>
<td>WSAN</td>
<td>Wireless Sensor and Actuator Networks</td>
</tr>
<tr>
<td>WSN</td>
<td>Wireless Sensor Networks</td>
</tr>
<tr>
<td>WTA</td>
<td>Weapon Target Assignment</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Mark-up Language</td>
</tr>
</tbody>
</table>
Chapter 1

1 Introduction

With the advances in Information and Communications Technology (ICT), the popularity of mobile devices (laptops, mobile phones, or other smart personal devices) has increased, resulting in shifting the computing paradigm from personal computers to ubiquitous computing (shortly UbiComp) [1]. First introduced by Mark Weiser [2, 3] with the idea of integrating different types of computing devices in everyday activities, allowing users to access information anytime, anywhere. In this vision, the core principle is to shift the computing paradigm from desktop computers with traditional input/output devices into a smarter environment where input/output and computing devices are pervaded in everyday objects. In such an environment, according to Weiser’s vision, computation and communication will become indiscernible from the environment where they operate [3, 4]. Computing devices in future UbiComp include sensors, actuators, communication devices, RFID, etc. and they are referred to as resources or assets.

Implementing UbiComp makes use of networked heterogeneous devices like mobile phones, sensors, actuators, as well as computers embedded in every gadget or electronic goods a person owns in order to “understand” the environment (through capturing contextual information) and interact with it [1]. Enabling UbiComp span over a wide range of technologies and research areas including -but not limited to- the Internet, mobile and wireless communication, sensor technology [5], Artificial Intelligence (AI) [6], Human Computer Interaction (HCI) [7], Internet of Things (IoT) [8], etc. These areas have been under considerable research in recent years; as a result, technology has been progressing and providing essential “bricks” for building a UbiComp environment (e.g., sensor networks, wearable computers, Radio Frequency IDentification (RFID), etc. [9]). Regardless of advances in UbiComp technology; before being able to implement a full-fledged UbiComp, other essential steps need to be reached: as devices and elements of UbiComp become more integrated with everyday objects, they should also become invisible to people. Weiser argues in his paper “The computer for the 21st century” [3] that:

“The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.”
Having computing devices “invisible” to people requires a shift in the computing paradigm and HCI. Components of UbiComp should be able to sense their surroundings and react to events that happen in the environment according to user’s need.

1.1 Characteristics of UbiComp

UbiComp is characterized by the use of networked heterogeneous devices; they can be hardware, software, fixed, mobile, etc. [10]. UbiComp devices are increasingly integrated in our everyday lives, e.g., in smart objects. They work and can communicate with each other and eventually also with the environment surrounding them (through sensing and actuation), creating smart spaces that can respond properly to specific situations as well as end-user requests and needs.

Characteristics of UbiComp systems along with some of their enabling technologies can be summarised as follows:

- **Integration**: the first step towards integrating computing devices into our everyday objects is to minimize their sizes [11]. Advances in the nanotechnology field [12] resulted in making ICT devices much smaller. This dramatic size reduction facilitates integrating them into everyday objects and transforming them into smart objects [13];

- **Heterogeneity**: the vision of UbiComp can only be realized with the deployment of a wide range of heterogeneous -but still interoperable- hardware, software, and networking technologies with different capabilities, e.g., smart phones, PDAs, computers, sensor networks, etc. [1];

- **Ad-hoc Networking and Mobility**: smart objects in UbiComp are expected to be able to connect to available resources and services in their environment on-the-fly and without predefined infrastructures. Ad-hoc networking has been widely accepted as an enabling technology for communication and connectivity to the network in UbiComp [14]. Dealing with mobility is also an important feature of ubiquitous computing; as an essential aspect of everyday activity. If mobility was not supported in UbiComp applications, users would lose connectivity with services and resources when on the move which would contradict the “anytime / anywhere” concept of UbiComp [15]. Mobile Ad-hoc NETworks (MANETs) are one enabling technology which provides connectivity and mobility in UbiComp [16, 17];

- **Context Awareness**: Dey, et al. [18, 19] define Context as:

  “…any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to
Context awareness refers to the ability of computing devices and other elements of UbiComp systems to understand the current situation so they can offer services that are truly tailored to end-user needs, also referred to as smart services. The fact that context is “any information” allows it to span over a large combination of attributes like location, time, activities, as well as objects which constitute parts of user environment that define a specific situation [11]. Indeed, the ability to sense is a central requirement for enabling context awareness in UbiComp. Sensing helps capturing different characteristics of the environment, which may contribute to producing (part of) the contextual information needed to describe a situation [20]. Thus Wireless Sensor Networks (WSNs) are increasingly of interest as a key technology for providing context awareness aspects [21, 22]. WSNs are wireless networks composed of a potentially large number of small, low energy nodes (possibly mobile) equipped with a microprocessor for basic data processing, and a sensing board for monitoring aspects of the environment. In addition and in order to support connectivity, they are equipped with wireless interfaces like ZigBee\(^1\), or WIFI [23], or IEEE 1451 [24], or Bluetooth low energy (Bluetooth smart)\(^2\). WSNs are powered with battery, which may be non-chargeable [25, 26]. Xin, et al, [27] illustrate integrating WSNs in different UbiComp applications;

- **Energy constrains:** UbiComp devices will be everywhere, and not necessarily always connected to an unlimited source of power; it is therefore essential, in order to extend the network lifetime that battery-operated nodes make an optimal use of their available power based on dedicated power management mechanisms and strategies [28]. Despite being mostly battery operated; these networks are expected to survive for long periods of time – in remote locations sometimes– without the need to keep changing their energy resources. Research into new, energy aware routing protocols for WSNs and MANETs [29-31], as well as energy harvesting techniques [32] has been conducted;

- **Invisibility:** As we move away from standalone computers with specific input/output devices into the age of “calm technology”, computation also becomes smoothly integrated into users’ environment and activities; they become therefore less and less noticeable to people. Quoting Weiser [33]:

> “*If computers are everywhere they better stay out of the way*”.

\(^1\) [http://www.zigbee.org/](http://www.zigbee.org/)

\(^2\) [https://www.bluetooth.com/](https://www.bluetooth.com/)
In order to reach this level of invisibility, there is a need to shift the nature of interactions in HCI from explicit to implicit [7, 34]. Dix [35] calls the new approach for HCI: “incidental interaction”. This means that different actions or gestures performed by users will be detected by sensors and interpreted by the system. The process should be invisible to the users. One example where gestures are captured by sensors to perform some tasks can be found in Microsoft Kinect controller\(^3\), Wii remote\(^4\), and Play Station move\(^5\).

The aforementioned characteristics of UbiComp span over different aspects of UbiComp systems. While research has mainly focused on the technical challenges imposed by the new computing paradigm like interoperability between heterogeneous devices, energy consumption, networking, situation awareness, etc., challenges of implementing these technologies and integrating them into our environment are gaining more interest recently. Computing devices and sensing devices are already everywhere around, in smart phones, tablets, cars, fridges, TVs, etc. Small microcontrollers and computers which allow developing interactive objects applications also exist like Raspberry Pi\(^6\) and Arduino\(^7\); yet, the UbiComp vision still falls short of having computing devices widely integrated in our lives. One of the main reasons that have caused this failure is the lack of simple and efficient tools that allow users to control their environment and define their very own services and applications in a simple accessible manner [36, 37].

1.2 Existing UbiComp Initiatives

As previously mentioned, the shift of computing paradigm from personal computing to UbiComp has been driving research in enabling technologies for nearly 20 years; since Weiser first introduced the vision of future computing. A solid technology foundation to facilitate various features of UbiComp has already been achieved, such as planning, machine learning, autonomous systems, etc. Along with advances in enabling technologies we started to see various projects implementing smart environments, e.g., The Aware Home Research Initiative at Georgia Institute of Technology\(^8\), and MavHome\(^9\) (Managing an Adaptive Versatile Home) at Washington State University and the University of Texas at Arlington. The Georgia Tech initiative uses a three floor smart building equipped with different technologies (sensors, actuators, etc.) as a living test-bed for researching and prototyping new technologies with the aim of using off-the-shelf technologies in order to

\(^3\) http://www.microsoft.com/en-us/kinectforwindows/
\(^6\) http://www.raspberrypi.org/
\(^7\) http://arduino.cc/
\(^8\) http://www.awarehome.gatech.edu/drupal/
\(^9\) http://ailab.wsu.edu/mavhome/index.html
improve people’s lives. The main focus of the initiative covers: health and well-being, digital media and entertainment, and sustainability. MavHome, also aims at creating smart home environments by using sensors and actuators. Users can use MavHome to decrease the cost of maintaining the house while maximizing its comfort for example. iHomeLab\textsuperscript{10} is another centre for deriving research on smart building development based in Lucerne University of Applied Sciences (HSLU). The aim is to enhance energy efficiency, security, and comfort in homes, with research activities spanning over the following three research areas: Energy Efficiency (EE), Ambient Assisted Living (ASL) and Human-Building Interaction (HBI). Other smart environment projects include classroom2000 [38], Labscape [39, 40], and Philips HomeLab \textsuperscript{11}. Future UbiComp applications would follow a similar process of using various assets –including sensors and actuators, etc. – to gather information, infer context and trigger appropriate actions.

### 1.3 Motivation

So far, most of those projects have mainly been implemented for research purposes; a major challenge to implementing smart environment projects in our daily life is the massive amount of required customization and the complexity pertaining to this task. Some systems (essentially commercial products) already provide end-users with easy development tools in order to allow them take control of their environment. This approach, however, has some flaws; first, the vendors have control over the products which limits the users’ ability to extend their systems without getting back to the developing company, which in turns limits their application scope. Second, it will be very unlikely that one company will be able to provide all assets necessary for the various domains (environments or areas of interest) of UbiComp [37]. On the other hand, other proposed systems for programming smart environment are scalable and dynamic, they can work in any domain [36], however, they are very complex and therefore developing application is limited to developers, who predefine the possible behaviours of the assets in the system. As a result, the end-users get excluded from customizing or defining their own applications. With the increasing numbers of cheap, off-the-shelf products using open standards, the task of developing a smart environment is even more challenging and complex for both developers and end-users.

Research has also been taking another direction to hide technology and its details from end users by implementing self-learning techniques where devices learn from user’s behaviour and adapt themselves accordingly [36, 41]. While self-learning systems do hide technology from end-users, smart systems still need to be controllable when appropriate –especially due to the fact that people

\textsuperscript{10} http://www.ihomelab.ch/
\textsuperscript{11} http://www.research.philips.com/technologies/projects/ami/background.html
Chapter 1. Introduction

sometimes change their behaviour, making self-learning a repetition and time consuming process. We note that self-learning systems are out of scope in this thesis and the main interest in controllable environments.

In order to boost the uptake of UbiComp, there is a need to move customization functionalities in the system towards the user’s side, while at the same time there is a need for hiding the complexity and computation from the user. We can see here a very challenging task: empowering users so that they can get more control, while at the same time hiding at the same time complexity and pushing the underlying technologies to the background so that they becomes invisible to the user.

1.4 Research Problem

Taking all the above mentioned factors into account, the following research problem has been identified: how can prospective users of smart environments be empowered to manipulate and control their environments and make decisions reliably and efficiently about everyday objects and their properties without the need for expert knowledge of the underlying computing elements and architecture of the system?

The lack of any effective system providing such level of invisibility to computing devices, where users can reason about and control objects within their smart spaces while forgetting about the enabling technology has inspired this thesis and provided the rationale for investigating such a system. Users of UbiComp would only be thinking of their programs in terms of their environments and the behaviour they require, whereas the underlying system would take control of the technology enabling such manipulation.

In the following sections, the research challenges, objectives and technical contributions of this thesis are presented, thus and to make it easier to follow and map them, the following abbreviations will be used: Challenges are referred with C, Objectives with O and technical contributions are TC.

1.5 Research Challenges

To resolve this research problem, a number of challenges (C) have to be overcome:

C1) Moving technology and its computation to the background: how to offer smart environment concepts to the users of the system and allow them to reliably manipulate objects of this environment without the need for explicit understanding of the underlying technology that builds up the system;

C2) Overcoming the dynamic nature of the environment: The resources (devices, services) in the environment can change and so does their availability. Different environments have
different resources as well. This makes it very hard for users to define how a system should operate under different contextual information and varying resources;

**C3) Distribution of processing tasks:** With large numbers of different types of devices and services and with tasks that can exceed the capabilities of single sensors it might be possible to perform some tasks in more than one way and the system should be able to find the best solution without involving end-users.

Enabling easy implementation and control of UbiComp faces more challenges like, data aggregation and routing, etc. In this thesis we are mainly interested in the three challenges mentioned above as they provide an initial top-down approach that connects various aspects, necessary to enable the wider implementation of UbiComp technology.

### 1.6 Objectives

More specifically, this thesis focuses on introducing a top-down model to allow users of ubiquitous computing systems to program their environment and manipulate its objects and properties while hiding away the underlying technologies –in this case, we consider *Wireless Sensor and Actuator Networks* (WS&AN). The objectives are (O):

**O1** Providing end-users with a high-level and technology-agnostic object-centric programming language, so that they do not need to be aware of the available resources within their system. Using this language, users need only to focus on the behaviours they require from the objects in the environment, ignoring the technology that enables those behaviours. This essential step satisfies the invisibility criterion discussed earlier;

**O2** Separating the description of resources or assets (resources and assets are used through this thesis to refer to the various sensors and actuators available in the network.) in the network from those of the real world objects that users can reason about would contribute to adapting to the dynamic nature of the network. This separation eliminates the need for hard-coding the tasks to the assets in the network, enabling users to reason about the tasks that were not predefined in the setup of the system. Empowering this separation with semantic and spatial task allocation algorithms enables the system to make use of any asset available at the network for any imposed task at the required time and space. Tasks-asset allocation is an important aspect to reliably move computation to the background of the system and make technology “invisible”;

**O3** Optimizing the use of the network by deriving a novel allocation algorithm that takes into consideration improving the utility of the network as a whole when assigning more than one task that compete for the access to the assets in the network. Algorithms that can
Chapter 1. Introduction

optimize resource allocation would offer a solution to the problem of distributing tasks arriving at the network to available suitable resources.

1.7 Why Wireless Sensor & Actuator Networks?

One of the main features of UbiComp is context awareness (as mentioned in characteristics of UbiComp, section 1.1). The ability to sense the environment is an important enabler for context awareness, even if practically context could contain more information than just information about the physical world around. Small sensing devices that are able to detect different phenomena is a key technology to pervasive computing. McCullough in his book Digital Ground [42] motivates the need of sensors for pervasive technology:

“If technologies are to keep out of the way, they need to see us coming. If computationally embedded environments are to be useful yet unobtrusive, they have to recognize what is happening in them”.

The context in this case can be seen as the perception that an object has about its environment. Sensors need to report the activity they recognize and their perception of their surroundings to other components of the environment to take proper actions. Thus they need some sort of communication. With sensors everywhere integrated in all aspects of the environment, unplanned communication is the best way forward. WSNs are equipped with a communication component which enables this ad hoc means of communication making them suitable for future UbiComp.

Actuators complete the loop; quoting McCullough [42]:

“We want environmental systems to keep out of the way, but we also want them to do something”.

Actuators enable domain systems to regulate themselves according to different readings from sensors and feedbacks. Actuators that respond to changes in conditions have been around for sometimes now even if they and their “primitive intelligence” are hidden from us like thermostats that stabilize indoor temperature. Cars, as another example are loaded with actuators like airbags, anti-lock braking systems and many other small actuators that improve the whole system’s response. Wireless Sensor & Actuator Networks (WSANs) are thus receiving much attention in UbiComp, and for the aforementioned reason WS&ANs are the main focus of this thesis.

1.8 Overview of Contributions

The Technical Contributions (TC) of this thesis can be summarised as follows:
Chapter 1. Introduction

TC1) Extension of domain (area of interest) specific ontologies (sensor and object ontologies) with concepts of space by integrating the ontologies with the GeoSPARQL ontology [43] in order to represent and reason about objects’ and sensors’ spatial properties. This contributes to meeting O2;

TC2) Investigated and proved two algorithms (GeoSMA, SGeoSMA) for solving the problem of sensor mission or task assignment (mapping sensors/actuators to missions or tasks) based on their spatial properties. Spatial relations between user’s missions and sensors are derived based on the use of geospatial relations and functions –qualitative and quantitative – which are provided by TC1 and are the basis of the proposed algorithm. This contributes also to O2 by finding the most appropriate set of nodes for the task. The proposed algorithm proved to improve the allocation of tasks to missions especially in dense networks;

TC3) Provided and proved a multi objective optimized spatial mission allocation algorithm with the aim of utilizing the performance of the network. The new algorithm –called BiSMA– makes use of the optimization problem presented in the area of defence for weapon target assignment [44, 45]. This step contributes to O3.

TC4) Investigated and developed syntax and semantics for a high-level programming language for real world objects that can be used to manipulate real world objects. This contributes to O1 by:

a. Introducing a concrete syntax based on the actions that users might perform to manipulate real world objects. This provides the required level of abstraction for end-users, and allows the language to be dynamic and scalable to work in any domain where knowledge is presented in the right ontology;

b. Translating the user’s program into a mission target assignment problem which would make use of the first three contributions to find the set of assets in the network that can perform the users’ task and utilize the use of the network.

c. Extending the spatial representation of objects to include alleviation to be able to reason with 3D spatial relations, such as, above, and below.

Table 1-1, maps the challenges of the proposed research problem to the objectives of the research as well as to the technical contributions of this thesis.
Table 1-1 Mapping the Challenges to Objectives and Technical Contributions

<table>
<thead>
<tr>
<th>Challenges</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives</td>
<td>O1</td>
<td>O2</td>
<td>O3</td>
</tr>
<tr>
<td>Technical Contribution</td>
<td>TC4 (a, b, c)</td>
<td>TC1, TC2</td>
<td>TC3</td>
</tr>
</tbody>
</table>

1.9 Research Methodology

The research methodology that has been used throughout the thesis and can be seen in particular in Chapters 4 and 5, can be summarized in the following steps:

*Step 1:* The first step is to highlight the research challenges that each chapter targets, and highlight the limitations of current state of the art that have been found in the literature review in Chapter 2.

*Step 2:* After defining the research challenges and the gaps in the literature. The challenge is formulated, mainly based on similar research problems found in the literature.

*Step 3:* The proposed algorithms are then introduced, along with all the algorithms or technologies that are used in the proposed algorithm and any amendments to the algorithms.

*Step 4:* The algorithms are then evaluated by implementing them along with the algorithms of the literature review. Simulation is used to compare the proposed algorithm with the literature approaches based on use cases that are introduced in literature when applicable.

1.10 Publications


1.11 Structure of the Thesis

The remainder of this thesis is structured as follows:

Chapter 2: “Literature Review on End-User Development tools and Modelling for UbiComp”

This chapter starts by reviewing the state of the art in WSNs high level programming languages and end-user development tools that were introduced to allow users to create their own applications and customize their environment. Task allocation approaches for wireless sensor networks is then presented as one of the gaps found in the literature for end-user development tools. The chapter also reviews higher-level modelling for WSNs and UbiComp which target the semantics of the network components and its operations with a focus on proposed ontologies. Spatial reasoning and its formal modelling ontology (GeoSPARQL) is presented later in the chapter. Finally a discussion of the limitations and shortcomings of the approaches covered in literature, which hinders the wider application of UbiComp, along with suggested solutions to overcome these limitations are presented.

Chapter 3: “Proposed Architecture”

This chapter describes the proposed architecture of the system. First, it presents the design goals of the system based on the literature review’s conclusions. After that, the different components of the system are described and mapped to how they can contribute to the design goals. A top-down architecture of the system is presented describing the interactions between the different components of the system. The chapter ends by presenting a top-down walkthrough scenario to explain how the system would handle users’ mission and the interactions between its different components.

Chapter 4 “Spatial Reasoning for Wireless Sensor Network Mission Assignment”

The focus of this chapter is on the spatial allocation of missions to sensors in the network. It investigates an integration of two main ontologies –SSN and GeoSPARQL– to allow extending the SSN ontology with spatial representation. Two algorithms called Geospatial Sensor Mission Assignment (GeoSMA) and Selfish Geospatial Sensor Mission Assignment (SGeoSMA) are introduced and explained. The aim of the algorithms is to find the optimum set of assets that can contribute to a task based on the spatial relationships between the region of the mission and the sensing range of assets. Both algorithms proved to outperform the current approaches in spatial allocation which are based on a Minimum Bounding Rectangle and the availability of assets in the exact location of the task –currently used in spatial crowdsourcing;
Chapter 5 “Spatial Weapon-Target Assignment for Multiple Mission Assignment in Wireless Sensor Networks”

This chapter provides an extension to the spatial mission assignment by considering additional factors to the mission assignment problem like the available energy of the nodes and the priority or value of the task. With the aim of utilizing the use of the network, in this chapter, the mission assignment problem is formulated as a bi-level optimization weapon-target assignment (WTA) problem with spatial properties and restrictions. The aim is to utilize the use of the network by increasing its gain from the tasks at the upper level optimisation while reducing the number of allocated sensors at the lower level. First, a multi-star selfish gene algorithm is presented to solve the WTA optimisation problem. The algorithm proves that it outperforms regular genetic algorithms with different cross-over operators in terms of the network gain and the processing time, especially with the increase in the number of sensors. A nested by-level genetic algorithm based on the selfish gene and Hamilton rule for altruism to solve the bi-level weapon target assignment problem is introduced. The operation of the new algorithm Bi-level Selfish Mission Assignment (BiSMA) was compared with a greedy algorithm that aims to solve the task with highest value first. The simulation results prove that BiSMA outperforms the greedy approach and results in a better utilization of the assets in the network to improve the overall fitness of the network.

Chapter 6 “High Level Programming Language for WS&AN”

This chapter introduces the “Environment Programming Language (EPL)”, which is a textual programming language for manipulating real world objects and their properties by end-users without the need for any knowledge about the underlying technology that enables such manipulation. The chapter starts by introducing the syntax and semantics of the language and examples of how users would be programming and controlling their smart environment. The chapter proceeds to evaluate the design and syntax of the language and the system that lies underneath on different dimensions based on the cognitive dimensions of notation framework (a discussion set of concepts used to validate user development tools). We compare the design with other pervasive computing designs and wireless sensor & actuator networks based on CDs and prove by discussion that our language provides a better abstraction and simpler for novice programmers as well as non-programmers.

Chapter 7 “Conclusion and Future Directions”

Chapter 7 concludes the thesis; it discusses the advantages of building a system suitable for UbiComp applications, highlights and discusses the contributions of this thesis, and
outlines those research questions that have now opened up as a result of the completion of this work.
Chapter 2

2 Literature Review on High-level Programming tools and Modelling Techniques for UbiComp

As previously mentioned, one challenge for a wider implementation of UbiComp applications in real-world scenario is the lack of simple and scalable end-user programming models and tools; tools that can allow prospective users without technical background to take control over their environment and customise its behaviour.

Various programming models have been proposed for pervasive computing and wireless sensor networks. However, most of these approaches consider a specific environment; as a result, their scope remains too limited and they are unable to adapt to a new environment –even with similar resources. In these approaches the resources and capabilities of the network are usually hardcoded in the application, and when considering using a tool for a new environment, developers need to re-customize their applications and resources. Another limitation of these approaches is that they consider end-users as developers; they leave it up to the user to specify the behaviour of resources to establish the required task rather than entirely concentrating on the task. This approach does not consider neither the dynamic nature of the network nor the nature of potential users of the system.

As the cost of sensing devices reduces, they are likely to be deployed with very high density in some applications, and thus the network might be capable of performing tasks in various ways making programming these networks even more challenging to users, especially with the added burden of choosing the best assets to optimise the services provided.

In the previous chapter the challenges for a high level programming language were verified: (i) providing the appropriate level of invisibility and moving computation to the background, (ii) mapping user’s tasks to the available resources in the environment and optimizing the task/asset mapping. In this chapter, a review of the approaches that have been introduced for programming ubiquitous computing environments and wireless sensor networks is presented. A review of the approaches that have investigated task allocation in wireless sensor network as a step to reliably move computation to the background of a high-level programming model is also introduced.
Ontologies that provide knowledge representation for wireless sensor networks and UbiComp are then reviewed.

2.1 High-level Programming for UbiComp and Wireless Sensor Networks

2.1.1 The Need for End-user Development Tools

Before describing the state of the art of End-User Programming (EUP) tools –also called End User Development (EUD)– there is a need to understand how users expect their smart environment to act. The involvement of end-users and taking their needs into account during the designing process of building EUD tools has been a starting objective for many researchers aiming at providing EUD tools that can turn UbiComp into reality.

Different studies were performed with the aim of answering the question “what do users want from their environment” [37, 46-49]. In their survey, Holloway, et al [37, 46] conducted an online study and asked people how they imagine and expect a smart home application to behave based on both sensing and actuating capabilities. They found out that different people expect different behaviours from their smart home, they customize their environment very differently, and they even perform tasks differently. The varying personalised scenarios that people require from their smart home confirm the need for enormous customisation when implementing such systems. With UbiComp applications everywhere, there will not be enough developers to perform the task of designing personal applications to all different users. In addition, it is very hard for developers to speculate on the required information and sensors for defining a specific context or whether the information as defined makes sense to users [50]. Their observation raises the need for moving the customization step into the user’s side of the design process. They also found that users need more abstraction in terms of objects in their environment; meaning that they need a high-level user-centric interface to control and customize objects in their environment rather than a low-level technology-centric approach.

The study in [49] also identified two recurrent patterns in users’ defined applications of smart home: the first one is that users are not concerned with technology that would perform the required tasks; participants in the study did not reference cameras, sensors, controllers, etc. They were rather more interested in the task they aim at achieving in terms of people, objects, location, etc. The second pattern they found is that user’s description of data types is “implicit”; users would like to record a conversation and not to capture audio data. Their findings suggest again the need for a high-level
of abstraction that does not just hide a technology’s functionality and implementation, but also its existence.

Similar to the findings of the two previous studies, authors in [48] found that most users would like to be able to get information about their smart objects and control them. Once more, the need for a high-level abstraction of the system was raised. Forsyth and Martin [51] have looked at the wider picture of design tools for UbiComp applications and argued for the need to involve designers and business requirement into the development of UbiComp. In what follows, a review of the different tools that were proposed for UbiComp is presented after describing the main characteristics of such tools.

### 2.1.2 Features of End-User Development Tools

End-User Development was defined at the EUD-Net workshop [52] as:

> “A set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create or modify a software artefact”.

This definition highlights the emphasis on non-professional end-users as software developers. The main motivation behind developing EUD tools comes from the inherent communication difficulties existing between domain experts and programmers, and especially programmer difficulties in acquiring the domain knowledge. Empowering end-users, giving them the ability to develop their own customized application is perceived as a way to optimize the development cycle of domain applications [53]. However, this motivation comes with a huge challenge: EUD tools have to be designed in such a way that they overcome the lack of programming expertise of end-users.

EUD already succeeded in domains like games, mobile devices, data-bases, spreadsheets and browsers [37]. An early statistic in 2005 [54] estimated the numbers of end-users in 2012 to be 90 million in America, with over 55 million using EUD tools like spreadsheets and data bases. However, EUD has not reached its potential in many domains including UbiComp.

Characteristics of end-user development tools play a significant role in the success of the tool. In order to be successful, a EUD in UbiComp must meet the following requirements:

- **Usability (simplicity or user-friendliness):** First of all, an EUD tool should be simple so that novice users can easily develop applications. Repenning *et al.* [55] argue that syntactic

---

12 Domain refers to the area of interest, e.g. in a smart home environment a domain expert is the owner/user of the house where the technology needs to be implemented, whereas for environmental monitoring the environmental scientists are the domain experts because they are the people who know what information they need access to and when.
errors leads to anxiety amongst beginner programmers and reduces the willingness to learn a language. An important approach to provide usability of programming language that already exist in software engineering is abstraction. Abstraction aims at hiding the complexity of programming from end-users by providing them with easy to manipulate components where all the technical details are already implemented by developers, e.g. visual programming like AgentSheets [56] and programming-by-demonstration [57].

- **Expressiveness:** In addition to simplicity, the tool should be expressive enough to allow professional programmers build complex applications. Myers, *et al.* [58] have called these two themes for evaluating EUD tool as threshold and ceiling. Researchers have argued that an effective EUD tools must provide a “gentle” learning curve that allow end-users to develop more complex applications while their experience in using the tool increases [53, 59]. Fischer *et al.* [59] reviewed the trade-off between the cost of learning and expressiveness in programming languages (see Figure 2-1 below). The Figure shows that the ideal programming languages are those with high scope and low learning cost.

![Figure 2-1 Trade-off between cost of learning and expressiveness in programming languages][1]

- **Extensibility and Reusability:** An important property of EUD for UbiComp is to permit reusing, extending, and further developing new functionalities to the system in order to

---

[1]: figure21.png
meet changing and increasing users requirements [52]. This has to be provided with minimum changes or without any changes to the underlying structure of the tool.

- **Scope independence**: This refers to the ability to develop a wide range of programs and areas of interests using the same language. High-level programming languages like Java and C++ for example are general, whereas domain-specific languages (DSL) are tailored to a specific domain like Microsoft Office Excel. With UbiComp applications spanning over wearable computers, smart buildings, large scale sensing, etc., a EUD tool should support UbiComp across various domains, so that users will not have to learn a different development tool for each domain.

In what follows, a review of the different existing EUD tools for UbiComp is presented.

### 2.1.3 User-centric Programming for Wireless Sensor Networks

Providing a programming language to ease the use and deployment of wireless sensor networks has recently received an intensive and increasing interest in research, and many programming approaches (like middleware, microprogramming, etc.) were subsequently introduced. A survey of programming approaches for WSNs can be found in [60, 61]. However, very few languages have been proposed for programming WSNs at the user-level. These languages aim at abstracting the complexity of programming the sensors in the network away to make it easier for novice programmers and domain specialists to program the network.

SEAL [62] is an abstract programming language that was proposed for enabling domain experts and novice programmers to program a wireless sensor network without requiring help from computer scientists. SEAL is also a Domain Specific Language (DSL) that targets at WSANs (Wireless Sensor Actuators Networks) by providing a simple syntax that is easy to learn by novice programmers. The primary elements of the language are the components of the system (in this case, the sensor and actuator nodes in the network) to allow programming WSANs according to the logic of the application without worrying about complex aspects of the network such as routing. The code is then translated to the C language for deployment. Only two types of executable commands (read and use) are permitted by the language: “read” a component usage which is performed periodically and “set” command which is executed only once. Three types of components (sensors, actuators, and output) available in the language and their operations differ. The use of a sensor is to “read” its observation and return a value, as for an actuator the use is to “set” an activation for the actuator (like being ON or OFF), and finally, the output component which prints the data on the network with the “print” command or saves it when it becomes available. An example of a simple program in SEAL can be found in Figure 2-2;
WASP [63] is another high level programming language for WSNs used by domain experts. The language allows programming the behaviour of sensors in the network, for instance data processing and sampling rate, using the keyword “LOCAL”. In addition, the behaviour of the network as a whole can also be programmed, for instance data aggregation, using the keyword “NETWORK”. The compiler translates the language into the nesC program for TinyOS. Unlike SEAL, which can specify a specific sensor like temperature, WASP programs all sensor nodes in the network and defines their functionality to perform the same task, which further restricts the use of the language. Other programming languages include makeSense [64] and Cubix [65]. Even though these languages have provided a high-level programming model for WSNs and ease WSN application development, they target domain experts who know how the operation of the system should be carried out in order to perform a task. Users of ubiquitous computing applications would only consider the task they require without being aware of the task to perform and without caring about how the system performs it. Thus, UbiComp applications still require an even higher level of abstraction where users program the tasks in terms of real world objects rather than system components. In the next section, tools that have been introduced to provide higher level of abstraction for UbiComp are discussed.

2.1.4 Tools for End-User Development in Ubiquitous Computing

This section provides a review of the tools that were proposed for end-users of UbiComp applications. All these tools are motivated by the need for introducing an abstraction level which hides away technical details from the end-users. We categorize the tools based on (i) target domain, (ii) their design approach and (iii) their proposed method of interaction with users of the system, i.e. the development interface paradigm. Figure 2-3 provides an overview of the different aspects used in our classification.

2.1.4.1 Target Domains

The target domain reflects the generality feature of EUD tools; two categories can be distinguished in the literature:

- **Domain-independent**: these tools are general; their vocabulary is not specified for one particular domain and they can be extended and implemented in various applications with different devices and properties;
• **Domain-dependent:** on the contrary, these tools have an approaches that targets a specific domain, such as a smart home, or elderly home, etc. These approaches are restricted by their vocabulary or architecture to their specific domain making them very hard to adapt to other domains.

![EUD Tools Diagram](image-url)

**Figure 2-3 Classification of EUD Tools**

### 2.1.4.2 Design Approach

The design approach on the other hand is based on the abstraction level considered by the development tool. It refers to the primary elements of the programming language which the user will interact with. Two levels of abstraction can be spotted in the literature:

• **Device-centric:** the main components in these systems are devices that build up the smart environment, e.g. cameras, sensors, heaters, etc. At this level, the end-user is aware of the networked components that s/he needs to use in order to build an application that answers the required tasks. The user is concerned with: “what devices shall be used and how to make them work together”:
• **User-centric:** user-centric approaches, which sometimes are called task-centric or goal-centric revolve around users and their tasks to the system, and they hide the existence of any technology underneath. Elements of these categories are everyday objects and their properties. At this level, users are only concerned with the behaviour of the system, meaning: “how to respond to various events in the system and what the outcome of tasks is”.

The target domain and design perspective were briefly reviewed as they reflect the generality and abstraction properties on EUD tools and will be used for comparison. In what follows the development interface paradigm is described in more details, and finally all these tools are categorised based on the design criteria.

**2.1.4.3 Development Interface Paradigm**

Human Computer Interaction is one of the most important aspects of designing EUD tools for UbiComp. “Development Interface Paradigm” in the taxonomy found in Figure 2-3 represents the different HCI approaches that aim at making it easy for end-users to develop their own applications. In the literature, there are four different HCI approaches: visual, programming-by-demonstration, tangible, and multiple representation.

**2.1.4.3.1 Visual Programming**

Because many novice or non-programmers have difficulty in using textual languages and because dealing with syntax errors can be very challenging for those users, an alternative was suggested for EUD: using visual/graphical languages [66]. A Visual Programming Language (VPL) is a language that models concepts and components of the system as graphical objects—like icons and pictures—and that facilitates designing an application by manipulating these components graphically rather than textually. Some of the tools that follow this approach include CAMP, Playing With The Bits, and iCAP.

*Capture and Access Magnetic Poetry* (CAMP) [49] introduced an interface in order to allow end-users design their smart home applications based on the *magnetic poetry* metaphor. The interface provides a set of pieces (magnets) with a word written on each of them and users build their applications by arranging words into a poem or a statement. New words can be added if required using new magnet features. CAMP is built on top of INCA (INfrastructure for Capture and Access) [67] that allows constructing the application.

Playing With The Bits makes use of jigsaw pieces in order to represent the assets in the environment [68, 69]. Users program the behaviour of their system by dragging and dropping the jigsaw representation of heterogeneous devices in their environment into the work-space, creating more
complex devices and applications. All devices are defined as components using JavaBeans\(^ {13} \), which share their properties across the dataspace. Each component has a jigsaw representation.

On the other hand iCAP [70, 71] uses sketching. It deals with three types of context-aware applications: if-then rules, relationship-based activities (where a relationship can be spatial, temporal or personal) and finally environment customization. iCAP provides the user with an interface which is used to define necessary elements and properties for the application (people, location, time, objects, etc.), as well as sketch icons that are used to represent each element. iCAP uses those interface and sketch icons to describe situations and associated actions. The system can be set to either simulate the context aware application or be used in a real context-aware environment via the context toolkit [72]. Example of a scenario defined using iCAP: “if John is in the office and the temperature is less than 50°F, or if john is in the bedroom and the temperature is between 30°F and 60°F, turn on the heater”.

Other Visual Programming tools include, Visualino\(^ {14} \), a visual programming tool for Arduino, [73], HYP [74], iPlumber [75], Oscar [76] and Zhang, et al. [77].

While visual programming approaches present a simple-to-use interface to end-users using the concept of “jigsaw” pieces or “drag & drop”, a limitation of these approaches still remains. The tools are not expressive enough to model complicated applications and they do not provide the learning curve suggested for EUD tools [53, 59]. Another limitation is that most visual programming approaches are domain specific and they require the system developer to adapt the application to other domains.

### 2.1.4.3.2 Programming-By-Demonstration

*Programming-By-Demonstration* (PBD) – sometimes called Programming-By-Example (PBE) – aims at making software and application development as easy as possible by allowing users develop applications without writing code. Users build their application by teaching the computing devices the required behaviour by introducing examples that simulate the application and show how the system needs to interact with it.

The CAPpella [78] PBD paradigm integrates supervised machine learning with user input in order to establish the context-awareness in the system. Through a GUI, the user can create a new situation (like a meeting) and get CAPpella to record all the relevant information that happened throughout the situation. The user can then train the system several times on different meetings to learn the

\(^{13}\) http://www.oracle.com/technetwork/java/javase/documentation/spec-136004.html

\(^{14}\) http://visualino.cc/
behaviour and attributes related to that situation. CAPpella uses four main components. First, a Recording System composed of a collection of heterogeneous sensors (camera, microphone, RFID, etc.) to capture raw data related to a specific situation. Second, an Event Detection component which uses the data recorded by the sensors to derive higher order information about the situation e.g. number of people in a meeting and their locations, or whether someone is speaking. The third component is the User Interface, which is used to check all recorded information and to amend necessary changes, e.g. using or omitting the data from one sensor. The last step is then to send the data to the machine-learning component either for training purposes or for actually detecting any behaviour.

Other PBD approaches include PiP [79], GALLAG Strip [80] and Alfred [81].

PBD approaches have a clear advantage: generality; the user can train the system on how to operate under different contextual information and environment by using sensor’s readings to detect different contextual information and on how to respond to specific events and information. However, disadvantages of PBD approach is their reliance on machine learning. A system will not be able to respond reliably to any situation without large training sets to make sure that it can capture all relevant information. This makes it also not suitable for applications where user’s required behaviour can change to a new situations because it needs to be trained for every new situation making it not suitable for real-time UbiComp applications.

### 2.1.4.3.3 Tangible Interaction or Representation

EUD tools in this category have considered the required shift in HCI for UbiComp. They are tools introduced for designing applications based on physical representations and interactions. iStuff [82] introduced a physical toolkit for controlling the environment –in this case an interactive room, with the main goal of allowing multiple users to control different devices in the environment simultaneously. The approach aims at integrating different devices (called iStuff components) either to make new devices or to introduce a new behaviour to the interactive room. An iStuff component can be any device, like a button, light, pen, speaker, etc, and the components can be combined in order to define a specific user-customized behaviour in the system, such as recording a program on TV. It was built on top of iROS (interactive Room Operating System) [83]; which is a middleware for information exchange between user applications and available devices in an interactive meeting room (shortly iRoom). Each device in the environment is represented as an iStuff component. It is a pairing of the physical device mounted with a Radio Frequency (RF) transmitter/receiver and a proxy connected to the iROS server for encapsulating or extracting data. The application communicates with the devices using event messages supported by iROS. One implementation of
iStuff was called iWall: an application to allow multiple users with different cursors to manipulate different images projected on a wall.

Media Cubes is another tangible language for programming smart spaces [84]. Elements of the language are cubes –wooden so far– each face of which can be associated a specific behaviour by the user (the cube has 4 active faces). Each cube is mounted with a collection of sensors, transducers, a microprocessor, and batteries. The interface of the cube is composed of a press-button, a LED, and a transducer that informs the user about status change which means a nearby cube. Thus, these cubes are thought of as “one-button remote controls”. The idea is to have one controller for more than one device, then the user can use the same cube to control similar functionalities in radio, VCR, TV, CD player for example. Different cubes are used to program a specific behaviour in the environment. These include: “Do-When” cube, “Event” cube, “Clone” cube, “Connector” cube, “Generalization” cube and “AllOf” cube.

StoryRoom [85] and SiteView [86] are other tangible programming tools for UbiComp applications.

Tangible interaction moves beyond the usual input/output methods of using regular screens, keyboards, etc. However, all approaches that have been proposed under tangible programming category aims at creating smart objects and are still far from providing a model to program an UbiComp applications. The reason for that is because: (i) most approaches are still hard coded for a specific applications, (ii) all languages are aimed still at smart objects and the interaction between all these devices to create a smart environment is still very limited.

2.1.4.3.4 Multiple Representations

Tools with Multiple Representations are those that include any combination of previously mentioned representation styles –in addition to the general textual programming interface. The aim is usually to make the tool expressive and simple enough to support both end-users and professional developers. This is achieved by providing different representations for different levels of abstraction.

Gracia, et al. [87], presented a simple and flexible programming tool for programming smart environments. At the core of their approach, there is a kernel rule-based programming language that enables representing all interactions within the system. Because the kernel language is textual and therefore not really user-friendly, three UI on top of the kernel language were proposed. These UIs are: a drag & drop interface, a word tokens-based GUI (similar to the magnet poetry interface), and a menu-based web interface. The system runs on top of a middleware [88], which abstracts objects into high-level entities with properties and relationships. A rule-based programming
language is introduced for writing actions based on context of the system [89]. The users can use either the textual programming or visual interfaces to develop their applications.

Another multiple representation tool is ESPPranto [90]. It is a part of the Edutainment Sensor Platform (ESP), which aims at allowing users with different programming skills to combine a collection of sensors and actuators and to produce tangible applications, mainly for creating games and educational toys. ESP is composed of two components: the ESPPranto SDK, which runs on a PC, is used to develop the software of the application and the ESP runtime environment which runs on the embedded devices and executes the applications. In order to accommodate different types of user skills, ESPPranto SDK is separated into four different UI layers, each of them allowing different flexibility and complexity as far as developing applications is concerned. For novice programmers, a graphical editing layer was presented based on drag & drop approaches. The second layer is called the macro layer, and consists of a textual representation of the graphical layer mainly targeting domain experts. For professional programmers, the next layer is the kernel layer at the core of ESPPranto language, which programmers use in order to define the basic behaviour of the hardware of the system. Finally the most complex level is the legacy programming language itself.

Different applications were built using ESPPranto SDK such as TagTiles [91] and StoryToy [92].

Microsoft Touch Develop is another multiple representation application development environment for developing applications for mobile devices, tablets, and computers. It allows access to sensors available with the device, data, media, and cloud storage. From drag and drop to textual script writing, the editor provides a learning curve suitable to the user’s programming skills. It also allows users to share their developed applications over the web.

As we have just seen, the expressiveness limitation—which other design approaches also have—was dealt with in the multiple representation tools by providing different interfaces addressing both beginners/non-professionals and expert/professional users. Because of the different levels of complexity, multiple representation systems can support users with their different levels of abstraction, they can further satisfy the generality extensibility to the system feature of an end-user programming model for ubiquitous computing.

2.1.5 Discussion

Regardless of advantages and drawbacks of the different tools presented, most of them are still far from being able to provide a coherent tool for EUD of UbiComp. Their major limitation is their level of abstraction: all approaches are still far from providing the proper level of abstraction required by UbiComp users, because they all still take the developer point of view and leave it up

15 https://www.touchdevelop.com/
to the user to implement the system and define how its different components need to interact to accomplish a certain behaviour, instead of just defining the behaviour.

In order to provide a high-level of abstraction which can ensure invisibility as envisioned for UbiComp, task-based or user-based abstraction are required. In their guidelines for developing Domain-Oriented Design Environments (DODEs), Fischer, et al. [93] emphasised the importance of invisibility when designing development tools for domain experts: “Domain experts like to interact with problems not computer systems”. One of the challenges with providing such a level of abstraction is the trade-off between simplicity and expressiveness. Loke [94] also recognizes the role of task-based abstractions for easing application development for users: “Task-based abstraction can express user tasks via underlying services with minimal effort”. He identifies one of the gaps in task-based abstraction to be mapping the tasks that users want to the services that the environment can provide.

As one gap in task-based abstraction is the need for task-sensor mapping, in the next section task allocation approaches that were introduced in WSNs are reviewed to identify what is missing to enable allow task-sensor allocation for UbiComp applications.

### 2.2 Task Allocation for Wireless Sensor Networks

Increasing interest in WSNs and WSNs applications in various domains including UbiComp brought a lot of attention towards the problem of mission assignment in WSNs. Sensor nodes are becoming cheaper; consequently, WSNs can be deployed with a high density of nodes. Which—in case all sensor nodes were active all the time—can result in large amount of redundant data to be sent through the network. As a direct consequence, the available energy at the nodes would drain fast and the life span of the network, which is expected to operate for a long period of time, will be drastically reduced. In order to avoid this situation, sensor nodes usually alternate between sleep mode and active mode with the aim of preserving their energy and therefore extending the overall network life span. Deciding which sensor nodes has to be active and which ones has to be idle is a problem that has received a lot of attention recently and is called sensor role or mission assignment or sensor role assignment. Sensor mission assignment is the problem of finding the most appropriate node, or selection of the nodes that are able to perform a specific task according to their capabilities in order to convey the information required by users.

Different approaches have been proposed for the role assignment problem in WSNs; they differ according to the different goals and factors they consider. So far, role assignment algorithms have mainly considered network coverage and object tracking and localisation with the aim of reducing energy consumption. Sensor role assignment as a mission assignment problem have also been
gaining a lot of interest in research recently. Other role assignment algorithms have considered quality of service [95], however, quality of service is not in the scope of this work thus it will not be discussed.

2.2.1 Coverage-oriented Sensor Selection

This schema targets densely deployed sensor networks. The aim is to activate only the minimal subset of nodes which can fully cover the area of interest and to put remaining nodes in sleep mode in order to save their energy [96-98].

One first approach presented by Perllio, et al [98], aims at finding the best schedule to determine the mode in which each node or set of nodes should operate and for how long to maximize the lifetime of the network, using linear programming. Finding the appropriate set of nodes for the application takes into account requirements from the application e.g. covering 90% of the field.

Another algorithm MC-MIP [96] tries to extend the lifespan of the network by finding the maximal number of disjoint sets of sensor nodes that can either cover a specific field or monitor a set of predefined targets. One disjoint set is active at a time and the role of the active sets is iterated in a round-robin fashion. By maximizing the number of disjoint sets, the amount of time a sensor node is active will be reduced, extending therefore the life time of the network.

DASS (Distributed Active Sensor Selection scheme) [97] is another algorithm for sensor selection with the aim of covering the sensing field with minimum sub sets of sensor nodes based on Voronoi diagram [99]. The algorithm can find a minimum number of sensors to perform the sensing task and cover the field of the task while preserving energy to extend the life span of the network.

2.2.2 Object Tracking and Localisation

These approaches try to find the set of nodes whose activation can improve the estimation of object location and tracking [100-102]. ANS (Autonomous Node Selection) algorithm [101] is a distributed strategy for selecting nodes which can contribute to a tracking and localization task for a specific target. A node decides if it will contribute to the localization task based on its additional utility relative to the active set of nodes. Each node uses knowledge of the relative location of the objects to itself as well as the previous active node, to decide if it should participate in object tracking and how far to send its information through the network.

Another algorithm that aims at maximizing information utility for object tracking was introduced by Pahalawatta, et al [102]. This algorithm uses an Unscented Kalman filter [103] for solving the tracking problem and finding the set of nodes which will perform the task.
Zhao, et al. [100] presented an information-driven collaborative sensing tracking algorithm. The algorithm ignores the localization phase and concentrates on the tracking phase. At the first tracking phase, a node estimates the best next sensor node which is believed to provide the best measurement for tracking the object and passes the status of the object to the best next node which again searches for the best next node and so on. The result is increasing the utility of measurement just like the previous approaches.

### 2.2.3 Sensor Mission Assignment as an “Application Layer” Task

Sensor mission assignment as an “application layer” task is different from the two previous categories. In this case, the goal of sensor selection is to find the nodes that are most suitable for a specific task –tasks might be different through the life time of the network–. This means that the selection schema try to enhance the performance of the network while meeting the energy constraints and requirements of the tasks. This problem is challenging because current sensor networks are application centric and pre-programmed for performing a specific task, or a set of tasks defined by their respective applications. This limits the re-usability of the network by other applications. Algorithms for this kind of sensor selection mainly use a utility function in order to (i) figure out the relevance of a node to the task and (ii) assign nodes which provide are more relevant to the task while considering energy the nodes’ energy limitation [104, 105].

Very few approaches consider the application layer mission assignment problem. The use of mission context for node assignment rather than the location of the node, its address or energy level has not received much attention yet.

In [106, 107]; the authors attempt to solve the problem of mission assignment in Intelligence, Surveillance and Reconnaissance (ISR) systems from a semantic web perspective. They propose using a combination of task and asset ontologies and a Mission and Means Framework (MMF) [108] in order to assign different assets to tasks. Their aim is to assign a collection of ISR assets (including sensors and sensor platforms) to a mission that can be composed of different tasks in order to satisfy ISR requirements.

A similar problem to the application layer mission assignment in WSNs can be found in the field of defence; the problem is then called Weapon Target Assignment (WTA) which will be introduced and described in more detail in Chapter 5;
Chapter 2. Literature Review

problem consists of optimally assigning various numbers of weapons (they can be of the same type) to a set of targets with the aim of maximizing the expected damage to the targets (minimize their expected survival chances) or to maximize the survival value of the defence assets. Unlike classical optimization problems, WTA allows allocating more than one asset to the same task. In the past, WTA problems used to be solved by human operators making all decisions. However, with the increased complexity of assets, WTA aims at supporting operators in decision making.

There are two types of WTA: Static WTA (SWTA) and Dynamic WTA (DWTA). The SWTA problem considers a single phase approach: it first assumes that all information about the situation is known and then aims at solving the problem as a resource constrained assignment problem. The DWTA, on the other hand deals with the problem at different phases, where the results of every phase (in this case the effect of the attack of first set of weapons) is evaluated, and this affects the allocation of the assets at the next phase to maximise the damage [112].

WTA is a problem that has received a lot of interest recently, and different approaches have been proposed in order to solve the problem, many of them being based on biologically inspired algorithms, like for instance genetic algorithms. However, there is not yet an optimum algorithm to solve the problem of WTA yet especially with large numbers of assets and targets. We formulate the problem of mission assignment in wireless sensor network based on the SWTA problem.

Since genetic algorithms have been successfully implemented for solving WTA problems, we present two genetic algorithms that we will be using to compare with our introduced algorithm.

2.2.3.1.1 Genetic Algorithms (GA)

Genetic algorithms are a subclass of the evolutionary algorithms that aim at finding solutions for optimisation problems inspired by natural evolution. Developed by Holland [113]; a GA presents a heuristic search approach inspired by natural selection. In nature, fit species have higher chances of surviving and transferring their genes to their next generations through reproduction. In the long run, generations with strong genes become dominant while the weak ones are more likely to be extinct. During evolution, sometimes changes might happen to the genes, if the changes have advantages, they survive and new species evolve. However, if the changes had no advantages they would be discarded by natural selection.

In GA, a solution for a problem is called a chromosome (or a genotype). Each chromosome is made up of a set of units called genes, and each gene is responsible for one feature of the chromosome. As such, a chromosome is a mapping of the problem to a possible solution. A solution space of a problem is a set of chromosomes called a population. The algorithm starts by finding a population —which is usually randomly initialized. While the search for an optimum solution continues through
an iterative procedure, the chromosomes in the population improve to provide better fitness for a network task until they eventually converge to an optimum or near optimum solution.

GAs use two genetic operations in order to iterate the chromosomes in the population and to generate new generation of chromosomes with the aim of eventually finding the optimum one; these are: crossover and mutation. In the crossover procedure, two chromosomes from the population are chosen –these are called parents– based on their fitness for the task, which is usually the objective function in the optimization problem. The parents’ chromosomes are combined in order to produce a new generation of chromosomes called the offspring. Because choosing the parents is based on their fitness for the task, their offspring are more likely to inherit good genes making them fitter parents for the next generation. By iteratively performing the crossover procedure, it is more likely to populate chromosomes with better genes until eventually they converge to a good solution.

The other operator used by GAs for populating fitter generations is mutation. Mutation introduces random alterations to the genes in the chromosomes. In Gas, with a very low mutation rate, new generations are not very different from their parents. However, this low mutation rate helps bring back diversity to the solution space. Reproduction for producing next generation is also based on the fitness of the chromosome to the task –just like the first step--; the higher fitness the chromosome has, the more likely it will survive for the next generation. The procedure of genetic algorithms can be seen in Figure 2-4.
2.2.3.1.1 Crossover Procedures and Mutation

Crossover:

As already described, the crossover operation combines genes from two parents and produces an offspring that constitute the next generation. There are different approaches introduced to perform
the crossover operation, in this research we consider two operations to compare with the proposed algorithm these are:

- **One Point Crossover (OPC):** sometimes also called One-Cut-Point (OCP). In OPC a gene from the chromosome is randomly selected and genes from parents are swapped either before or after the selected point and the resulting chromosome is the offspring. The following example explains the OPC operation:

  Assume having 7 weapons (W=7) and 5 targets (T=5), possible parents are:

  \[
  \begin{align*}
  A &= 1 \ 5 \ 2 \ 1 \ 1 \ 5 \ 3 \\
  B &= 3 \ 5 \ 1 \ 2 \ 3 \ 4 \ 2 
  \end{align*}
  \]

  The A parent mean that weapon 1 was assigned to task 1, weapon 2 was assigned to task 5, weapon 3 was assigned to task 2, and so on. Assume that the randomly selected gene is the 4\textsuperscript{th} gene, and the crossover operator swaps the genes after the 4\textsuperscript{th} gene of the father’s chromosome with those of the mothers –also after the 4\textsuperscript{th} gene. In this case the resulting offspring are:

  \[
  \begin{align*}
  A &= 1 \ 5 \ 2 \ 1 \ 3 \ 4 \ 2 \\
  B &= 3 \ 5 \ 1 \ 2 \ 1 \ 5 \ 3 
  \end{align*}
  \]

- **The EX operator:** this process for crossover aims at producing offspring with possibly good genes by preserving the good genes from both parents and passing them to the offspring then randomly selecting two genes which were not preserved and exchange them. The operation of the EX operator is described in the following example:

  Assume having the same example of the OPC, and the same parents:

  \[
  \begin{align*}
  A &= 1 \ 5 \ 2 \ 1 \ 3 \ 4 \ 3 \\
  B &= 3 \ 5 \ 1 \ 2 \ 3 \ 4 \ 2 
  \end{align*}
  \]

  The algorithm first finds the genes which are shared between the parents, in this case the 2\textsuperscript{nd} and 5\textsuperscript{th}, these genes are checked if they have the highest fitness amongst all targets j assigned to the weapon i, \((V_j \times P_{ij}) – V_i\) is the importance of the task and \(P_{ij}\) is the probability that weapon i is assigned to task j. If the genes have the highest value they are passed to the offspring, then two other genes are randomly selected (let’s say 3\textsuperscript{rd} and 6\textsuperscript{th}) and swapped in the offspring, and in this case the resulting chromosomes are:

  \[
  \begin{align*}
  A &= 1 \ 5 \ 4 \ 1 \ 3 \ 1 \ 3 \\
  B &= 3 \ 5 \ 4 \ 2 \ 3 \ 2 \ 2 
  \end{align*}
  \]

  Mutation is a random procedure which aims at changing genes in the chromosome to bring back diversity to the solution and avoid a local optimum solution. Every new chromosome which results
from crossover can undergo a mutation process. For every gene in the chromosome, a random number is generated and if the number is smaller than or equal to the mutation rate, then the specific gene will be randomly altered to another gene.

2.2.3.1.2 Reproduction Procedures

After performing the crossover and mutation operation on the population \( p(t) \) –where \( t \) refer to the generation number– and finding the resulting children \( p(t+1) \), it is important to define the criteria for choosing members of the next generation to evolve. Fittest chromosomes are usually chosen to evolve and improve the population; this happens following an evolution strategy where first, all redundant solutions from both the parent and the offspring populations are eliminated to avoid a local optimum solution, and then the fittest genes from the resulting pool are chosen based on their fitness for the objective function. After the new population is selected, the whole evolutionary process is repeated until it reaches a stopping criteria. This can be a predefined number of generations or after a specific time is satisfied or even when reaching a solution with an acceptable fitness level.

2.2.4 Summary

The different approaches that have been proposed for mission assignment in WSNs have mainly considered optimizing the behaviour of the network according to node restrictions (like energy and coverage). They all consider the network to be homogeneous in terms of the property they observe, which is not realistic in future UbiComp applications. None of them –apart the Intelligence Surveillance and Reconnaissance (ISR) approach– have looked into the problem of task allocation from an application layer point of view where (i) tasks can differ through the life time of the network according to their context, and (ii) there is a risk that the requirement for a specific task cannot be satisfied by one sensor or a collection of sensors. However, unlike sensor mission assignment in ISR, and unlike the WTA problem, the mission assignment problem is more challenging in WSNs; in the WTA the assets are sent to the area of interest after finding the best collection that can answer the task’s requirement, while for WSNs the nodes have already been deployed to their specified locations and have capability restrictions caused by sensing range and obstacles in the environment around.

In order to be able to perform “application layer” tasks in WSNs, these tasks, as well as the network of assets that performs them, need to be specified and described. Having the tasks integrated into the application and then performing a task allocation algorithm limits the potential wide development and adaptation of UbiComp technologies;

Different approaches have been introduced to model and describe WSNs and UbiComp, such knowledge representation is important to provide the required level of description and semantics.
for the different assets and tasks to enable “application layer” mission assignment. In the following section he approaches that have been proposed for modelling UbiComp characteristics as well as its technologies are presented.

2.3 Formal Modelling of UbiComp and Its Enabling Technology

2.3.1 The Need for Modelling Ubiquitous Computing and Its Technology

Regardless of all the advances in UbiComp and its technologies, a precise understanding of ubiquitous computing, how it might develop and how its different technologies and concepts would integrate is still missing [10]. Building a ubiquitous computing system involves different activities like analysing, designing, formalizing, evaluating, verifying, etc; and modelling expands over all these activities. There already exist effective theories for modelling different aspects of computation and communication. These include –but not limited to– sequential computation, mobility, concurrency, knowledge representation, databases and distributed computing etc. However, there is not yet a rigorous model for describing different aspects of ubiquitous computing systems and their integration, as none of the existing theories is sufficient enough for dealing with UbiComp because none of them can capture and model all the different characteristics and features of UbiComp. A general model for ubiquitous computing would help building, analysing and verifying the system, its algorithms and protocols at different levels of abstraction [10].

More theories are being introduced to describe the behaviour and characteristics of ubiquitous computing technologies –as already mentioned– like WSNs, MANETs, context awareness, etc. Process algebra approaches have been proposed to model the lower level abstraction of MANETs; they mainly focus on introducing a model to verify communication aspects and protocols. Some of these calculi include; a Calculus for Mobile Ad hoc Networks (CMAN) [114], Calculus for Wireless Systems (CWS) [115], and \( \omega \)-calculus [116].

WSN is a domain that uses ad hoc networking for communication and for establishing a topology. Different process algebra calculi have been proposed for modelling WSNs based on those for MANETs. However, because WSNs have additional features in addition to those of MANETs, this imposes additional restrictions on the network, such as computation, memory, and energy limitation [117, 118]. Furthermore, these features are mainly application-specific [119]. New process algebra approaches have been proposed especially for WSNs to model their additional features like: a Calculus for Sensor Networks (CSN) [120], CALLAS [121], and Active Sensor Processes (ASP) [122].
In UbiComp applications, computation happens on multiple, small, integrated, and invisible devices that use wireless communication in order to exchange data. However, UbiComp has additional characteristics, mainly context-awareness [123], which is considered to be a key feature of adaptive systems. Heterogeneity of UbiComp devices makes context awareness essential in order to be able to reason about a user’s tasks and available services. Modelling context awareness has been receiving increasing interest [124]. Process calculus approaches for modelling systems with context awareness have been proposed. Most of these calculi are based on Ambient Calculus [125]. Examples of such approaches include: Context-Aware Calculus (CAC) [126], Calculus of Context-aware Ambients CCA [127, 128], and CONAWA [129, 130], etc.

On a higher level of abstraction, formal modelling describes the different concepts and devices found in the environment. Such a modelling approach focuses on the objects, their relations, and inherited properties. At this level of abstraction, knowledge is represented in different forms like rules and ontologies. Because the main aim of this thesis is on making UbiComp applications easier to develop by end-users, we are more interested in this high-level modelling of the system which is capable of providing invisibility to the system users by introducing real world objects rather than technology, e.g. sensors and actuators for users to manipulate. Different ontologies have been presented to describe the domain of UbiComp and technologies it relies on, like Wireless Sensor Actuator Networks (WS&AN) WS&ANs. Some of these ontologies are described in the following section.

2.3.2 Ontology for Modelling UbiComp and Its Enabling Technology

Ontology as defined by Gruber [131] is “explicit specification of a conceptualization”. It is a formal knowledge representation which describes entities in a domain of interest, their properties, and relations. There is a number of specifications that represent ontologies, such as Resource Description Framework (RDF)\textsuperscript{16}, and Web Ontology Language (OWL)\textsuperscript{17}, etc.

2.3.2.1 Ontology for Wireless Sensor Networks

Ontologies that have so far been proposed for WSNs aim at providing a description of sensor nodes in a WSN, their platforms, observation, properties, and data. There already exist a large number of ontologies for WSNs; some of them are described as follows:

2.3.2.1.1 Sensor Hierarchy Ontology (SHO)

\textsuperscript{16} http://www.w3.org/RDF/
\textsuperscript{17} http://www.w3.org/TR/owl2-overview/
SHO [132] was proposed to represent the WSN data semantics, which in turn can be used to improve search and processing of sensor data. The derived ontology was then used by the same authors for building a two-layer ontology that comprises of: on the upper layer Suggested Upper Merged Ontology (SUMO), and on the lower level three distinct ontologies: Sensor Hierarchy Ontology (SHO), Sensor Data Ontology (SDO), and Extension Plug-in Ontology (EPO) to ease data fusion and reasoning in a heterogeneous WSN [133]. Since the ontology is mainly concerned with the semantics of the sensor data, it mainly describes features and parameters such accuracy, calibration, and format, etc.

2.3.2.1.2 SensorML

Sensor Model Language (SensorML) [134] presents a model for describing sensors, their observation, geometry, and characteristics. It uses Extensible Mark-up Language (XML) for encoding, and it is one of the OGC’s Sensor Web Enablement (SWE) component that allows services and applications to connect to all types of sensors over the web. The main purpose of SensorML is to provide information about sensor data and measurement processes. Geometry is considered as a physical property that describes a sensor node. Geographic Mark-up Language (GML) is used for specifying the geometry property of the sensors in a network.

2.3.2.1.3 OntoSensor

OntoSensor [135] is a prototype for knowledge representation of a sensor network, it uses SensorML as its building block for describing all specifications of sensors, and it extends SUMO upper ontology to define knowledge based ontology for WSN applications. As the ontology uses SensorML as a backbone ontology, the location model of the ontology also depends on GML for describing coordinate reference system.

2.3.2.1.4 The semantic sensor network (SSN) ontology

SSN [136] was developed to describe different concepts for representing sensor nodes, e.g. their sensing devices, systems, observation, measurement restriction, and operations under different circumstances. The SSN ontology does not propose any representation about sensor locations and operational regions. It is left to the user to specify and model this information to provide more flexibility and reusability of the ontology. The ontology was aligned with DOLCE-UltraLite upper ontology.

Other sensor ontologies include [137-139]. A Survey of ontologies for describing WSNs can be found in [140].
2.3.2.2 Ontologies for Ubiquitous Computing

While ontologies for WSN are more concerned with specification of sensors and their data, ontologies for UbiComp specify all features and properties of UbiComp applications. Some of these ontologies are described in the following.

2.3.2.2.1 Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA)

SOUPA [141] is an ontology proposed to model applications for pervasive computing. The ontology describes the main concepts that correspond to an application; these include: person, time, space, action, policy, and event. Computing entities in the system are described using the concept of an agent, with properties of knowledge, belief, plan, intention, goals, etc. Space is used to model entities with spatial extensions. Spatial vocabulary is borrowed from OpenGIS and OpenCyc, as well as Region Connection Calculus (RCC) for describing spatial relations.

2.3.2.2.2 COBRA-ONT

Another ontology for pervasive context-aware applications is COBRA-ONT [142]. Mainly intended to describe an intelligent room scenario, this ontology describes places, events, agents, and their properties. The spatial information provided by the ontology is the location of an agent (longitude, latitude) and the containment relation of spaces between each other.

2.3.2.2.3 CoDAMoS project Ontology

The proposed ontology in [143] is aimed at creating context-aware infrastructure from small embedded devices to high-end platforms. The ontology distinguishes four different concepts: user, platform, service, and environment. It models context information at the lower level and the whole system at a higher level. The ontology is proposed as a base for solving challenges of UbiComp, like generating devices’ user interfaces, code generation, etc.

A survey of various ontologies for modelling ubiquitous computing and smart environments can be found in [144].

2.3.2.2.4 IoT-lite

An ontology that is used to describe the Internet of Things systems or resources, services and objects18. It distinguishes three types of resources: sensors, actuators and tag devices. Each resource has a coverage range to describe the area that is covered by the IoT device. The ontology makes use

18 http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lite
of the SSN ontology to describe sensing devices and connect it with other attribute and objects that can be found in an IoT environment.

2.3.3 Discussion

At different levels of abstractions, different approaches have been proposed for modelling the important characteristics of UbiComp system at the specified level. While at the lowest levels of details we find theories of communications, at a higher level of abstraction, ontologies that specify the domain knowledge are used.

Looking at the invisibility property of UbiComp applications, ontologies for UbiComp have considered the future vision of ubiquitous computing by looking into services, agents, users, and real world objects. This model of the world permits users to interact with smart objects rather than devices in the network. However, EUD tools are still too much device-centric and assume the users have to deal with assigning tasks to computing devices.

Many of the proposed ontologies that specify WSNs or applications of pervasive computing presented the use of spatial properties to represent the location of objects. Some approaches extend that to model with some qualitative spatial relationships like RCC. However, in all ontologies there is not yet a complete semantic description of spatial properties of the domain and none of the development tools have considered the spatial properties and relations of objects and assets for developing the applications. This is because all development tools are pre-programmed and the relations between different tasks and assets in the network is hardwired prior to the use of the system.

Because of the lack of a semantic description of the spatial properties of various objects in a domain ontology, the next section is dedicated to Spatial Reasoning, with the main focus on GeoSPARQL.

2.3.4 Spatial Reasoning

Spatial reasoning is crucial in many domains like transportation and emergency applications. Its importance for UbiCom has long been recognised by researchers in the area, Brumitt, et al [145] emphasised the importance of geometric representation for UbiComp:

“For computing to move off of the desktop and be accepted, it must have a comprehension of physical space which is related to that of the user, else the proliferation of smart devices will only increase the complexity of the user’s experience, instead of simplifying it”.

Spatial representation and reasoning have long been embedded with databases in these domains [146]. Extending databases with spatial representation allows efficient storage and reasoning with spatially indexed data so that questions like “find the set of temperature sensors that cover a school
Chapter 2. Literature Review

playground”, “find the nearest sensor node to an object” or “check if our tanks are in the firing range of enemy tanks”, etc. can be answered. The Open Geospatial Consortium (OGC) standards for geometry [147, 148] are the basis of extending both Oracle\(^{19}\) and Microsoft SQL servers\(^{20}\) with spatial representation and reasoning functionalities.

### 2.3.4.1 GeoSPARQL

GeoSPARQL is another standard from OGC that introduces a standardized representation of geospatial RDF to facilitate representing and querying spatial data on the semantic web [43]. It extends SPARQL\(^{21}\) –a Resource Description Framework (RDF) query language similar to SQL in databases– with functions and relations that allow qualitative and quantitative reasoning about spatial data. GeoSPARQL defines a small ontology –based on the OGC standards– composed of two main concepts for spatial information representation: Features and Geometry. The former concept represents any entity occupying a spatial location like sensor, room, person; etc. While the latter defines the geometry shapes used to represent the location of the object, and its other spatial properties like the location and area occupied by the object. GeoSPARQL ontology also provides a set of functions and relations to allow qualitative and quantitative reasoning over spatial entities.

Extending current ontologies for WSNs and domain information with spatial representation and reasoning capabilities can better allocate tasks to various sensors in the network.

### 2.4 Discussion

Having investigated the state of the art in various research challenges imposed to high-level programming for future UbiComp applications, we identify the gaps in literature which needs to be investigated to make UbiComp a reality. Table 2-1 briefly describes the approaches reviewed in the state-of-the-art (SotA) for each of the research challenges and their limitations; it also proposes approaches as well as technology (if available) to overcome these challenges.

Examining the challenges we have set earlier and considering the SotA presented in this section, it can be noted that the link between UbiComp future applications and their enabling technologies as well as space modelling and reasoning are still missing. While ontologies proposed for UbiComp do provide the required level of abstraction and describe objects of the domain of interest and their properties, these advantages and approaches in the literature have not yet found their way into the tools proposed for developing UbiComp applications. Those tools still allow users to deal mainly with the technology of UbiComp applications, and thus they do not provide the level of invisibility

\(^{19}\) [http://docs.oracle.com/cd/B28359_01/appdev.111/b28400/sdo_intro.htm](http://docs.oracle.com/cd/B28359_01/appdev.111/b28400/sdo_intro.htm)


\(^{21}\) [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)
expected from future UbiComp applications. Integrating end-user tools with domain ontology to allow dealing with real world objects rather than technology is one approach to reach the required level of abstraction in UbiComp.

Another challenge that arises when dealing with the changing tasks of the users is how to find the appropriate set of assets available in the system to perform the task? While task allocation in WSNs have mainly looked into constrain of the node in the network and aimed at performing one specific task for as long as possible, or with the maximum coverage, future applications will request tasks which are different according to the users’ preferences, and thus having all tasks hardcoded in the system –as is the case in current applications– will limit their usability. To overcome this problem, a mapping between tasks and assets needs to be performed at some level in the system. In order to establish invisibility and hide the technology from the users, allocating tasks to assets needs to be performed by the system in the background, i.e. at a lower abstraction level without any involvement from the end-users. This requires a reliable dynamic task allocation approach. Semantic matching of tasks to assets is one approach to facilitate user’s task allocation.

Presently a crucial element that is still missing in UbiComp application and tools is the formal modelling (representation and reasoning) of spatial properties of tasks and objects. Indeed, since UbiComp systems are related to spatial environments, formal representation about spatial properties in addition to semantic reasoning can be very useful to ensure that the assets in the system can fulfil user requirements.

One limitation on spatial task allocation in WSNs is the lack of formal representation of the spatial properties of sensor nodes. GeoSPARQL is an ontology that can fill in that gap and allow representation and reasoning about spatial relationships in the system.

With the vision of having sensing devices integrated with every asset associated with a person, smart environments will be highly populated with sensors everywhere. This means that it might be possible to accomplish some tasks using different combinations of assets. Therefore, deciding on which sets of assets can be used to perform required tasks becomes another challenge UbiComp applications will face; this has unfortunately not yet received much attention research-wise. However, the problem is well-formulated in the field of defence as the WTA problem. The problem of mission assignment in WSNs can be formulated as that of WTA with additional restrictions that can reflect the characteristics of WSNs, with the aim of utilizing the use of resources in the system.
### Table 2-1 SOTA gaps, Challenges and Proposed technology for UbiComp Systems

<table>
<thead>
<tr>
<th>Research Challenge</th>
<th>State of The Art Approaches</th>
<th>Gaps in SOTA</th>
<th>Proposed Technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Level Programming Languages, (invisibility) (C1, O1) <em>Chapter 1: section 1.5 (p.6), section 1.6 (p.7)</em></td>
<td>• End User Development Tools with; PBD, visual, tangible and multiple representation interfaces.</td>
<td>• Still asset or technology based approaches • They do not provide the required level of invisibility</td>
<td>The use of domain ontology and its concepts of real world objects as primarily components of the language (TC4) <em>Chapter 1: section 1.8 (p.9)</em></td>
</tr>
<tr>
<td>Mapping users’ tasks to available assets in the system (C2, O2) <em>Chapter 1: section 1.5 (p.6), section 1.6 (p.7)</em></td>
<td>• Task allocation considering the coverage of the network, object tracking and Utility base approach • Ontology based task allocation in ISR system</td>
<td>• No application layer task allocation in WSN • No consideration of dynamic tasks and networks • No spatial task allocation</td>
<td>Provide spatial properties representation and reasoning using GeoSPARQL ontology (TC1, TC2) <em>Chapter 1: section 1.8 (p.8-9)</em></td>
</tr>
<tr>
<td>Optimizing task allocation using spatial allocation (C3, O3) <em>Chapter 1: section 1.5 (p.7), section 1.6 (p.7)</em></td>
<td></td>
<td>• No optimization for high-level task allocation, so far all tasks are hard coded to the assets available in the network</td>
<td>WTA formulation of the optimization problem to better utilize the performance of the system (TC3). <em>Chapter 1: section 1.8 (p.9)</em></td>
</tr>
</tbody>
</table>
Chapter 3

3 Proposed Architecture for End-User Development Tools

While considering a completely automated approach to control the environment using machine learning and other intelligent techniques will ensure keeping the technology out of the way, smart systems need to be operable and controllable where appropriate [42]. Some systems might need to be configured only once, while others might need to be updated as part of a daily routine. Quoting McCullough [42]:

“Know when to eliminate an obsolete “legacy” operation, when to automate and when to assist an action. Know how to empower not overwhelm”.

The vision of this thesis is to propose a top-down model for programming and controlling real world objects by non-programmers without the need for any knowledge about the underlying enabling technology. A big challenge of providing such level of abstraction –as has already been defined in the problem challenges– is: how to move computation/communication in such systems to the background, and how to reason reliably and efficiently about everyday objects and their properties without prior knowledge of the underlying computing elements and architecture of the system performing the reasoning?

A multi-level system with different levels of abstraction and various components that are compatible with the research objectives, and whose integrations can answer the research problem is proposed. At the highest level of abstraction, a programming language based on real world objects is introduced to enable manipulating one’s environment (O1, Chapter 1: section 1.6 (p.7)). At the lower level of abstraction, the specification of the system’s technology as well as the real world is described using a collection of ontologies (O2, Chapter 1: section 1.6 (p.7)). The information provided by these ontologies provide the basis for resource allocation algorithms that can find the most appropriate set of assets capable of performing the tasks required by users of the system while keeping the technology invisible from them (O3, Chapter 1: section 1.6 (p.7)).

In this section the proposed architecture for such a system is presented. First the design goals that influence the architecture design are highlighted. Based on those design goals the proposed components of the system and its architecture are introduced. Procedural steps are finally described
to help understand how the different components of the system can work together and contribute to providing the required features of the system.

### 3.1 Proposed Architecture

#### 3.1.1 Design Goals

The design goals (DG) of the system have been identified based on both the State of the Art (Chapter 2) and motivations of the need for such a system. They are split into different components based on the gaps found during literature review.

At the highest level of abstraction there is a need for an abstract language based on real world objects that the end user will interact with, we define the following design goals from literature review.

**Programming Language Invisibility**

DG1: *An end-user programming language for UbiComp should be task-centric.*

**Rationale:** Prospective users of UbiComp applications are not interested in technologies that enable performing tasks but in the tasks themselves. In order to reach the vision of “calm” computing with computing devices everywhere, making technology invisible to the users is essential. As such, a successful tool should model and present the users with objects of real world, which users are interested in manipulating rather than the technology that allows manipulating them.

DG2: *An end-user programming language should be simple enough to allow users to take control of their systems.*

**Rationale:** Prospective users of future UbiComp applications are people with different backgrounds many of them being non-technical; high level programming language should abstract away from the complex low level details of computation and communication and be simple enough for allowing users taking control and developing their own applications.

DG3: *An end-user programming language should be expressive and flexible.*

**Rationale:** With the changing demands and requirement from users on their smart environment, and with the varying interest of different users, a successful language should also be expressive enough for allowing users to build complex applications and change their environment corresponding to their needs.
At a lower level of abstraction, computation and communication should be performed by the system while being kept invisible from users. To allow invisible computation in the system we define the following design goals:

**Semantic Task Allocation**

DG4: *An end-user “friendly” programming tool should be able to perform task allocation without involving users.*

**Rationale:** solving various missions and programming the behaviour that users require from their environment either happens explicitly at the user’s side or implicitly at the system’s side. For a successful end-user programming tool to be able to perform and customize the WS&AN’s behaviour following user’s requirement, the detailed tasks (allocating sensors and actuators to specific task) which build the environment’s behaviour have to be solved by the system not the users. In order to guarantee the required level of abstraction which can hide technology, task allocation should happen at a lower technical level that is invisible to the users.

**Spatial Task Allocation**

DG5: *An end-user programming should provide an efficient spatial modelling of the domain of interest and WS&AN.*

**Rationale:** since UbiComp applications are related to spatial environment, establishing formal representation and reasoning about the spatial properties of objects in addition to the ontological ones (relation between objects e.g. member-of, is-a,...) is a good addition to ensure the provision of an accurate model of the objects within the system and the space they occupy. Subsequently they provide the users with the ability to reason on special properties of objects (see DG6 below). Partitioning space (which has been the main trend in modelling UbiComp systems) imposes unnecessary restriction on location model which limits its representation capabilities. Continues space is the natural modelling for objects of real world in UbiComp to better reflect reality and allow a more reliable reasoning about spatial properties and relationships of real world objects.

DG6: *An end-user programming tool should also provide efficient spatial reasoning and task allocation.*

**Rationale:** in order to allow different undefined tasks in the environment, being able to reason about the assets’ locations, functional range as well as their spatial relations to objects in the system is essential. Providing an efficient spatial model of the system is a powerful first step (DG5), as it enables reasoning about the spatial properties of objects. However, as explained
in DG4 task allocation is essential to hide the technology, computation and complexity of the network from the user. With the increasing interest in crowdsourcing (especially in UbiComp environment) where a task can use assets that do not belong de facto to the system but bound to the system at the task location time, spatial reasoning becomes even more essential.

3.2 System Components and Architecture

Visual and Textual User Interfaces
A visual interface for users to manipulate objects of real world and program their environment would satisfy the goal for DG1 and DG2. Main components of such an interface needs to be imported from the domain ontology to present objects of real world which users can manipulate. Satisfying DG3 requires another interface—in this case textual—. Providing such an interface for users to control their environment corresponds to TC4. In this thesis, a visual interface has not been implemented yet, only the textual interface have been implemented.

Ontologies
As already mentioned, ontologies are important for presenting real world objects and their properties to the users to manipulate as the main components of the programming language. However, they are also important for other design goals of the system (DG4, DG5, and DG6). A domain ontology would describe those real world objects together with their properties and relationships, while an asset ontology (in this case Semantic Sensor Network Ontology SSN) for describing assets and their properties is important for the semantic task matching occurring in the background. This latter ontology would provide a basis for satisfying DG4 by describing capabilities and properties of the system assets. An ontology to describe the spatial properties of assets and objects is also essential to satisfy DG5, as well as being a base for DG6. Ontologies take part in our TC1, TC2 and TC3 (Chapter 1: section 1.8 (p.8-9)).

Sensor Task Fitting Reasoner
To satisfy DG4, a reasoner would first try to find the different types of assets that can help answering the object-level tasks required by the end-user (users are only interested in objects and not in assets). Finding the potential solutions happens at two levels: at the first level, semantic matching between the assets and the tasks is performed in order to find the available types of assets that can perform such a task, and at the second level a spatial matching is established to find the set of all the assets that can provide the required information in term of their physical location and spatial relationship to the task. Spatial reasoning would satisfy DG5 and contribute to TC2, TC3 (Chapter 1: section 1.6 (p.8))
Allocation Algorithms

- **GeoSMA, SGeoSMA**
  
  In order to optimize the task allocation of assets, two allocation algorithms aim at finding the optimum allocation of sets of assets to the various user tasks. After finding the set of all assets that can contribute to a mission, the algorithms will look for a subset of the solution space that can best optimize the use of the WS&AN, by minimizing the number of nodes that contribute to the mission. GeoSMA, SGeoSMA is our TC2 (*Chapter 1: section 1.6 (p.8)*).

- **BiSMA**
  
  Another allocation optimization algorithm can be used after finding the set of all nodes that can take parts of the various missions arriving at the network. This algorithm aims at allocating assets to the various tasks in the network with the aim of optimizing the performance of the network. The algorithm tries to increase the utility of the network by choosing the assets with higher detection probability while at the same time it reduces the energy consumption in the network by choosing the minimum set of nodes (TC3, *Chapter 1: section 1.6 (p.8)*).

**Code Generation**

After finding the optimum task–assets allocation, and based on the different mission properties specified by the users –like the frequency of performing the task or its length– a corresponding code will be generated to each of the assets in the task. As mentioned before this step and the next one has not been implemented.

**Code Deployment**

The code is then sent to the appropriate nodes in the network

**Top-down System Architecture**

Figure 3-1 shows an overview of the architecture. Ontologies lie in the heart of our proposed model and they act as the building block for all other actions and processes that are performed by the system.
Fig. 3-1 System Architecture

Figure 3-2 provides an end-to-end picture of the reasoning process from the users interface down to code deployment.

The figure shows the different components of the system and how they interact with each other to implement and accomplish a task specified by the end-user.

In the following section we present a walkthrough scenario in order to show what can be a task defined by an end-user and how the different components of the system interact with each other to satisfy the requirements of the end-user’s task.
3.3 System Operation Steps

In this section a top-down operational steps of the proposed architecture for end-user development tool for UbiComp is presented. The scenario covers the following steps:

1. Import a domain ontology to facilitate reasoning on information,
2. Define the end-user’s mission to the system –or the Behaviour Requirement (BR),
3. Extract properties of real world objects as well as objects themselves from the BR,
4. Map the mission requirements to the assets that can contribute to the mission based on task/asset mapping,
5. Find the optimum set of assets that can be assigned to the defined mission,
6. Based on the BR and the result of the allocation algorithm, generate mission-based code for each asset,
7. Deploy the code to the assigned assets,

*Step 1*

The aim of our approach is to empower users with the ability to reason about real world objects and their properties at a high level rather than defining explicitly the behaviour of the system at the low level. In order to allow such a behaviour, the user tool or programming language needs to deal with real world objects of the specified domain only, keeping low level issues at the background. The programming environment imports a domain ontology and presents its concepts and their properties as the main entities for defining the required behaviour of the environment. In a smart home scenario for example, an ontology that describes all the objects in the home like kitchen, living room, bed, alarm, etc. along with their properties like temperature, presence, would be presented to the user of the system as objects that he/she can manipulate.

*Step 2*

The end-user defines the task that he wants the system to perform in terms of real world objects that have been already imported or filtered from the ontology using the programming language provided. One example of such a mission or BR can be: “detect presence in an office”. The user can define more complicated missions about properties that are not already defined in the ontology but based on pre-existing ones like: “detect all suspicious vehicles on the road where a suspicious vehicle has a speed above the threshold”.

*Step 3*

The system need to extract properties and objects that the user reasons about from the mission, in the first case the property is “presence” and the object is “office”. In the second example the property is “speed” while the object is the “vehicle”.

*Step 4*

A semantic mapping from the behaviour requirement to the available assets in the network is performed. The aim of this step is to find all types of assets that can take part in performing the mission. This mapping happens at the class level of the ontology before allocating the mission to
the set of appropriate asset instances. At this step the system would find the assets that can detect presence in an office in the first example based on the assets available in the network. The outcome of this step might be “CCTV camera or motion detector”. However, the second example will need data about the speed of vehicle and will require a different set of assets. This step would help narrow down the search in the network for the most suitable sets of assets.

**Step 5**

Once we have identified all types of assets that can be a part of the task solution, finding the instances that can be allocated to the task is the next step. At this step task allocation happens based on a spatial reasoning about the region of a mission and the capabilities of the assets in the network. In the first mission example, the range of the mission or requirement of the program is the area occupied by the office. In the second example the range of the mission is the space of the garage. The system tries to find the set of all instances of assets that can be a part of the mission because their sensing range can monitor the required area. Spatial relation between the region of the assets and the region of the task is important because it might be the case that a single sensor is not sufficient to detect the property, in which case, a collection of sensors might be required to perform the mission, and readings from more than one sensor is required. For example, for detecting office presence, an available CCTV camera might not be able to perform the task alone because its sensing range cannot cover the whole area of the office. In addition, for detecting an intrusion, an alarm needs to be raised requiring an additional actuator to perform that task.

In the case of the first example, after finding a list of all assets that can be a part of the task, we can end up with two possibilities: “office1 CCTV” and “office1 IR”. Then the system tries to optimize the solution by finding the minimum set of assets that can answer the task. Many benefits can be obtained by minimizing the solution sets, these include: energy saving and reducing the number of packages sent through the network. Finding the minimum set of assets also have the advantage of keeping other assets free to be used by other missions. After optimizing the solution space of the first example, a possible solution can be to only use the IR in the office rather than using both sensors, as a single CCTV camera might not be able to cover the whole area of the office or be only directed towards the door. However, there might be a case where the solution space cannot be optimized because sensors cannot cover the whole area of a task and in this case the algorithm sticks to all sensors that can contribute better rather than getting no answer at all.

**Step 6**

After finding the optimum set of assets in the network, a code that corresponds to the task the user wants, needs to be generated for the assets in the solution set. It is worth noting that the user’s mission can have additional parameters – not considered in the previous step- that correspond to
the behaviour of the task like for instance how often the task needs to be performed if not only once. An example of such a task can be “detect the presence in the office every 10 seconds during 2 hours”, the temporal parameters do not play any role in the allocation algorithm but can affect the program to be generated for the different assets in the network.

Step 7

After generating the code for the nodes that will take part in the mission, the code needs to be disseminated through the network in order to be deployed at the appropriate nodes in the system. However, code deployment is not a part of this research study, as the aim is to be able to translate an object centric task to a task allocation for WS&AN.

3.4 Conclusion

Ubiquitous computing entails a wide range of research areas and technologies, from static, mobile, and wearable technologies, mobile agents, Internet, resource description, discovery, integration, location model, etc. A successful UbiComp system needs to be able to combine these various technologies while at the same time be easy to operate by regular technology-agnostic users. Thus, while it is very important to represent the various types of components of a system (data, processes, location models, devices) at some level of abstraction, it is as important to hide them at another (at behaviour description level for example). This illustrates the need for multi-level systems of UbiComp to enable its wide deployment and user by end-user.

In this section we investigated and reviewed the design goals that lead to the proposed architecture of such a multi-level system. A system that combines device model, location and spatial model, with resource allocation techniques and a domain specific language for end users. We described the different components which build up a system for real world reasoning and manipulation. We presented a walkthrough scenario in order to illustrate the behaviour of the system and the interaction between its different components. The purpose of the scenario is to facilitate the understanding of the proposed architecture and system design and to show how an end-user application would look like and be implemented by the proposed architecture.

In the following sections different components of the system are described in detail, and the proposed algorithms are reviewed. The gaps found in literature (which where emphasised in chapter 1&2) will be thoroughly filled at the different levels of the system architecture.

In the next section, the problem of task allocation in wireless sensor network from an application layer perspective is targeted using spatial properties of sensors and tasks.
Chapter 4

4 Spatial Reasoning for Wireless Sensor and Actuator Network Mission Assignment

One of the gaps to establishing easy programming of ubiquitous computing applications that we found in literature is the lack of mapping between the tasks required from the network and the services and assets available in the network, referred as the problem of sensor task assignment or allocation.

As already mentioned (Chapter 2), the problem of sensor mission assignment from an application layer perspective has not received much attention. What we mean by mission assignment from an application layer perspective is the problem of identifying a collection of sensor and actuator nodes that are capable of satisfying the requirement of a task specified by the end-user. In this case the user’s tasks can have different contextual information which might not be answered by a single sensor and require the collaboration of more than one sensor. One form of this contextual information that we consider in this section is the spatial information.

Location models and geometry representation is essential for future UbiComp; Microsoft researchers Brumitt, et al [145] have presented the case for incorporating physical space and geometry in UbiComp:

“Fundamentally, a geometric model can greatly improve the user’s experience in a system where devices and interactions are dynamic. This occurs because appropriate devices can be automatically selected, and because the system’s and the user’s knowledge of the physical world has been moved one step closer together”.

Sensors’ capabilities are restricted by their locations and sensing ranges, and they can only contribute to tasks that take place within their sensing ranges. Allocating tasks to sensors based on their sensing ranges has not been considered in literature due to the lack of spatial representation of sensors’ and actuators’ spatial properties in WS&AN.

Unlike mission assignment in Intelligence, Surveillance and Reconnaissance (ISR) systems where the assets are sent to perform a specific task after finding the appropriate assets that can answer the ISR requirement –like Unmanned Aerial Vehicles (UAV) or Unmanned Ground Vehicle (UGV)–,
the problem is more challenging in WSNs. The reason for that is because the network is already deployed and sensors have locations and spatial properties, such as sensing and transmission range, which limits their ability to contribute to various missions in the network. On the other hand, and unlike the previously introduced sensor role assignment approaches—which consider a homogenous sensor network where all sensors have similar sensing ranges and functionalities—sensors in ubiquitous computing applications would be heterogeneous with variable capabilities and restrictions—like a wall that can limit their sensing range—. Tasks will also have various contextual information including their location and the spatial area they occupy. So far, no approach has considered the relationship between the sensing capabilities of sensor nodes and task areas in order to perform mission assignment.

In this chapter, a novel spatial-based mission assignment algorithm is introduced. The algorithm finds the most appropriate sensors—in terms of their sensing range—capable of answering an end-user’s mission. This goal is accomplished by extending ontologies for sensor networks and tasks with spatial properties representations and introducing an allocation algorithm which takes into consideration the spatial relationships between both the sensor and the task ranges in order to find the most suitable set of assets able to perform the task.

### 4.1 Requirements for Sensor Mission Assignment

The use of semantic matchmaking to find the various types of sensors that can accomplish users’ tasks is proposed; this would then be used in order to find a spatial matching of sensor’s functional range to the region of the task. This approach relies on the use of ontologies as an expressive way to represent knowledge and reason about it. We use ontologies in our approach for the following activities:

- Specify the capabilities provided by the assets—sensors/actuators—in the network; including spatial properties;
- Specify properties of real world objects in a domain of interest—including their spatial properties;
- Compare the specifications of real world objects against those of assets in the network to identify the set of assets that are fit to a task;
- Compare the spatial relations of real world object against those of assets in the network to find assets that can participate to a specific task.

The use of integrated ontologies which includes all objects and assets in a domain of interest, allows building on top of existing ontologies in various domains rather than reinventing the wheel for every application. This would result in a flexible approach that can work in various domains rather than
an approach tailored to a single application or application area. The two ontologies at the core of the proposed approach are GeoSPARQL and SSN; in what follows a description of the essential features of these two ontologies and how they can contribute to a geospatial approach to sensor mission assignment.

### 4.1.1 Spatial Reasoning with GeoSPARQL

As already been presented, GeoSPARQL is composed of two main concepts: Features and Geometry. Figure 4-1 shows the relation between feature and geometry classes in GeoSPARQL ontology.

![Figure 4-1 GeoSPARQL Ontology Main Classes](http://schemas.opengis.net/sf/1.0/simple_features_geometries.rdf)

The geometry properties of an object can be of shapes like point, line or polygon or multi point, etc. These shapes are defined in the simple feature ontology\(^\text{22}\) (SF) –used by GeoSPARQL. The different shapes of geometry objects defined in GeoSPARQL and their hierarchy can be seen in Figure 4-2.

It also allows defining collection of spatial Geometries. GeoSPARQL uses two serializations for representing the geometry objects on the map, these are: Geography Markup Language (GML) and Well-Known Text (WKT) literals.

---

\(^{22}\) [http://schemas.opengis.net/sf/1.0/simple_features_geometries.rdf](http://schemas.opengis.net/sf/1.0/simple_features_geometries.rdf)
GeoSPARQL introduces a set of relations and functions that extend SPARQL in order to enable topological and geometry-based (non-topological) reasoning about entities of space based on their spatial specifications and relations. Topological relations –also referred to as qualitative spatial relations– describe spatial entities based on their inherent relations to each other. GeoSPARQL presents vocabularies that describe three types of topological relations. These vocabularies are: Simple Features (sf), Egenhofer, and Region Connection Calculus (RCC-8). Examples of functions and topological relations between two geometries that are supported by GeoSPARQL include: equals, disjoints, overlaps, touches, within, contains, covers, tangential proper part, etc. Querying about these relations in GeoSPARQL would return a Boolean value which reflects their spatial relation.

Non-topological relations supported by GeoSPARQL allow quantitative reasoning in addition to the qualitative reasoning provided by topological relations based on geometry information. Some functions provided in this category include: distance between two nodes, the convex hull of a region and the Minimum Bounding Rectangle (MBR) -also called envelope- of a region, etc. Querying about quantitative relations between spatial entities would return either a new geometry (like the union of two geometries of the MBR) or in the case of the distance between two geometries will return the shortest distance in units between both geometries.
4.1.2 Semantic Sensor Network Ontology (SSN)

SSN [136] has already been reviewed in Chapter 2. What we are especially interested in is the interconnection that SSN can provide in order to relate sensors to tasks and objects. Such a connection can bridge the gap between real world objects and the assets used to manipulate them; indeed, it can be used to assess the semantic fitness of sensors to the various tasks required by the end-user of the system. SSN provides Stimulus-Sensor-Observation paradigm, which aims at describing the link between sensors and the property of real world objects they can detect. The connection can be seen in Figure 4-3.

![Figure 4-3 Stimulus-Sensor-Observation SSN [136]](image)

A property of a feature of interest –like the temperature or presence property of a room or the wind speed in a particular town (where feature of interest can be any real world object or task) is detected by a specific sensor as described by the SSN ontology. This interconnection provides the basis for reasoning about the semantic matching of sensors to different missions, which would facilitate manipulating real world objects in our model at a high level of abstraction.

As already mentioned, SSN does not provide any spatial information about sensors or their properties and leaves that information to the developers of the application to deal with allowing more flexibility.
4.2 Ontologies for Semantic Spatial Reasoning with WSN

We propose the integration of three different ontologies in order to allow geospatial reasoning about tasks in the network. At the core of our ontologies are GeoSPARQL and SSN, the connection between both ontologies and domain ontologies is the basis for reasoning about the fitness of assets to tasks in the network. The interconnection can be seen in Figure 4-4.

A domain ontology is dependent on the application of the system. Here, objects are described based on their properties, attributes, characteristics, etc. In a smart building scenario, a domain ontology should be able to describe all objects of the building like rooms, doors, windows, pieces of furniture etc. One example of such an ontology is the *Industry Foundation Classes* (IFC) for describing elements of a smart building\(^{23}\).

\(^{23}\) http://www.buildingsmart-tech.org/specifications/ifc-overview
Integrating ontologies –domain ontology and SSN– with GeoSPARQL enables geospatial representation and reasoning about sensors, objects, the physical space they occupy as well as their spatial relationships using the functions and relations provided by GeoSPARQL ontology. In order to ease spatial reasoning, a characterization of the primitive spatial entities and their use to represent the spatial properties in the network is introduced. The following main geometries for describing objects and sensor spatial properties are considered –the representation of spatial properties follows the GeoSPARQL and SF ontologies for describing space:

- **Object/Assets Placement:** the location of objects as well as assets in the network is assumed to be a point in a 2D geometry defining the longitude and latitude of the objects, like the location of a sensor or actuator node. The representation of a point geometry in the simple feature (sf) ontology is: \textit{Point} (long, lat), using GML or WKT serialization;

- **Object Region:** the region of an object is a spatial representation of the area covered by the object itself, like the space of a room or a building floor. Representing spatial objects is done using the polygon geometry object of the simple feature ontology, and it is represented in (sf) ontology as: \textit{Polygon} ((long$_1$ lat$_1$, long$_2$ lat$_2$, long$_3$ lat$_3$ ... long$_n$ lat$_n$, long$_1$ lat$_1$)).

- **Sensor Sensing/Transmission Range:** similar to the object space, the sensing and transmission range of a sensor refers to the space covered by the sensors respectively for sensing or for communication. The sensing range represents the area that lies within the scope of the sensing device –like temperature, humidity, etc. – of the node. While the transmission range is the area covered by the transmitter. Both ranges are represented similar to the region of the object using polygon object following the (sf) ontology. Figure 4-5 shows the specification of a sensor range using Protégé. Other spatial properties describing the location, transmission range of sensors and missions are specified similarly.

- **Actuator Functional Space:** the functional space of an actuator is the area in space surrounding the actuator in which it can perform a specific task. It is described similarly to the sensing/ transmission range of the sensor node.
4.3 Spatial Reasoning Approach

This work introduces two novel spatial task allocation algorithms: Geospatial Sensor Mission Assignment Algorithm (GeoSMA), a Branch and Bound (BB) [149, 150] algorithm for spatial task allocation in WS&AN, and Selfish Geospatial Sensor Mission Assignment (SGeoSMA) an evolutionary algorithm for task allocation in WS&AN. Both algorithms use spatial functions and relations between sensors and tasks in the network as basis for task allocation. The aim of the algorithms is to find the minimum number of assets as well as their area of coverage that can contribute to a specific task in the network.

4.3.1 Spatial Matching of Tasks and Assets

The use of a rich spatial specification facilitates the use of spatial functions to reason about the fitting of various assets to the tasks required by the end users of the system such as if the sensing range of the sensor covers or overlaps with the region of the task. We assume that end-users reason about properties of objects, and that each object occupies some space –even if that space was just the point which the object is located in. This means that every task required by the user of the system has a spatial property –which is not necessary visible to the end-user– like area of a room or a building floor; thus the programming environment can assign available assets to the task allowing...
the end user to reason about properties of real world objects. Figure 4-6 shows the basic spatial matching relationships that we use in our algorithm based on simple feature (SF) topological relations. The relations and their description according to GeoSPARQL is presented after the figure. Assuming that \( T_r \) represents the region occupied by the user’s task and \( S_r \) represents the sensing range of a sensor node. It is worth mentioning that the shapes and areas of spatial regions differ between various sensors and tasks in the network.

Figure 4-6 Basic Spatial Matching Relations found in SF ontology

- **Equals**: \( sfEquals(T_r, Geometry, S_r, Geometry) \) returns true if both geometries are equal and false otherwise;
- **Disjoint**: \( sfDisjoin(T_r, Geometry, S_r, Geometry) \) returns true if the two geometries have no point in common;
- **Overlaps**: \( sfOverlaps(T_r, Geometry, S_r, Geometry) \) returns true if \( T_r \) and \( S_r \) have some but not all points in common;
- **Touches**: \( sfTouchs(T_r, Geometry, S_r, Geometry) \) returns true when \( T_r \) and \( S_r \) have at least one boundary point in common;
- **Contains**: \( sfContains(T_r, Geometry, S_r, Geometry) \) returns true is geometry \( S_r \) contains the geometry \( T_r \);
- **Within**: \( sfWithin(T_r, Geometry, S_r, Geometry) \) returns true if geometry \( S_r \) is contained by geometry \( T_r \);
• Intersects: \( sfIntersects (T, \ Geometry, \ S, \ Geometry) \) returns true if geometry \( T \) and geometry \( S \) have at least one point in common – whether the point was in the boundary or interior of the geometry. This makes the intersect relation is span over, touches, overlaps, contains and within;
• Union: \( union (S_1 \ Geometry, \ S_2 \ Geometry): \ S, \ Geometry, \) returns a new geometry object which represents the union of both geometries \( S_1 \) and \( S_2 \);
• Envelope: \( envelope (T, \ Geometry): \ MBR, \ Geometry, \) the function returns a new geometry which represents the minimum bounding rectangle, or minimum boxing rectangle (MBR) of the geometry \( T \).

4.3.2 Geospatial Sensor Mission Assignment Algorithm (GeoSMA)

We formulate a spatial Branch and Bound (BB) algorithm to facilitate finding the optimum solution for allocating sensors to tasks based on their abilities to contribute to the required task, by taking their sensing ranges into account [151]. BB is an algorithmic paradigm for solving optimization problems by searching the entire solution space in order to find an optimum solution. The algorithm dynamically populates a rooted tree with all possible solutions for the problem and evaluates the possible solutions at every state of the solution search process. At the root of the tree we find the full solution set. The algorithm then populates the branches of the tree with subsets of the full solution and evaluates the populated solutions. Before populating each branch, it is tested against a lower bound which is calculated at each step. The branch is eliminated if it is unable to produce a better solution to the current optimum one. Evaluating every branch prior to continuing the solution search process is important to decrease the overall search complexity, because it allows searching only a part of the solution space; doing so eliminates the need for populating the exhaustive set of possible solutions which can take too long due to the exponential increase of possible solutions [150].

To implement the BB algorithm for spatial task allocation, we consider that the optimisation problem that the algorithm tries to solve is to minimize the number of sensors. The rationale for minimizing the number of nodes is to keep as many free sensors as possible in order to allow other tasks to be allocated in the network and –in case there are no more tasks– to preserve the energy of nodes in order to extend the life span of the network by using as minimum resources as possible.

Most existing approaches for mission assignment in WSNs assume that all sensor nodes have similar capabilities and ignore the spatial properties and restrictions of the nodes when assigning missions to them [95]. They are also pre-configured in order to perform one single specific task. On the other hand, the approaches that consider the spatial properties of a task either consider the task to occupy one 2D point in space –like tracking and localization tasks– and they do not look at spatial
regions of the task or consider the possibility that task may be covering the entire field of interest – maximizing the coverage. They also perform one specific task that was predefined and cannot adapt to other different tasks assigned to the network. In other research areas spatial task allocation has been emerging as a new focus of recent studies –like spatial crowdsourcing [152-157] –where the system tries to allocate tasks to different workers available at the location where the task needs to be performed. However, these approaches assume that the worker has to be exactly at the same location of the task or has to move to the location of the task. They represent the worker’s and task locations as a point in a 2D space. Here, the worker location and the task location need to be either exactly the same Point(long, lat) or within a certain distance –which the worker needs to travel. This is not a realistic assumption as some workers can contribute to a task without being at exactly the same point, provided that they are close enough.

The challenge of task allocation when considering spatial properties for tasks and sensors is that not all sensors in the network can contribute to a specific task. What makes task allocation more challenging when considering spatial regions (i.e. geometric shapes and not only single 2D points) of a task is that it might be impossible to find one sensor that can perform a specific task in the network due to availability issues or capability restrictions. In this case more than one sensor node may be needed in order to perform a specific task. An example of such a case would be to detect a suspicious person inside a building, in this case using one sensor –a camera for example. To perform such a mission might have low success rate but a collection of sensors located at different floors and rooms in the environment can carry out the task with high success.

In our approach, we use semantic and spatial matching techniques and try to match tasks to the available sensors in the network before allocating a set of sensors to a specific task. This helps filtering all the sensors that cannot contribute to the mission –either because they cannot detect the required property or because the task does not fall in their sensing ranges. As a result we can reduce the solution space of the algorithm in order to find the optimum allocation in minimal amount of time.

As said earlier, every object occupies a specific region, and since a task is reasoning about and manipulating objects then we assume that every task has a region which is the spatial geometry occupied by the object. Thus every task allocated within the network is considered to be a spatial task. The space or range of a task $T_j$ is given by $T_j(r)$. Every sensor in the network $S_i$ has a sensing range as well given by $S_i(r)$. A sensor can contribute to a specific spatial task required by the end user if the range of the task intersects with the sensing range of the sensor. In this case, the solution space (SP) for a specific task is defined as the set of all sensors whose sensing ranges can intersect with the region of the task;
Where $S$ is the set of all sensors with a type that can be assigned to the mission.

In dense networks, it can be the case that there are multiple sensors and sets of sensors that can perform a specific task. We consider a valid solution to be a set of sensors that are capable of covering the region of the task. An optimum solution set $\text{OptS}$ is then defined as:

$$\text{OptS} = \min \left\{ \sum S_i \mid S_i \in S \bigwedge \left( S_i(r) \text{ intersects with } T_j(r) \right) \right\}$$  \hspace{1cm} (4.2)

The lower bound of the algorithm which every solution will be compared to is considered to be the region of the task $T_j(r)$. This means that any branch in the algorithm is tested against this bound and if it cannot satisfy it (the union of the sensors region cannot cover the space of the task) then none of the branch’s children can, and hence the branch is eliminated. The algorithm starts by finding all the sensor nodes that can contribute to a specific task because their sensing ranges intersect with the region of the task. The set of sensors is considered to be the current optimum solution—the steps of the algorithm can be tracked in Figure 4-7 with the numbers on the right side, this step is (1). The algorithm then tries to find a better solution by dividing the solution set to subsets and compare them with the current solution (2). Before comparing the current solution to the optimum one, every subset is compared to the lower bound of the algorithm trying to find the minimum set of sensors— or a single sensor— whose aggregation can cover the range of the task. If the algorithm finds two solutions with the same number of nodes it will choose the set of nodes whose aggregation of space is smaller. The optimum solution would then be the one with a sensing range that is equal to the region of the task (3)—the optimum solution can be one sensor node that can cover the region of the task. The algorithm repeats the second step and divides the current optimum solution to a smaller ones in the search for a new optimum one (4). If the branch does not satisfy the lower bound which is the region of the task, the branch is eliminated (5) and the algorithm continues to explore other branches (6), when all branches have been explored or eliminated the algorithms returns the current optimum one (7).

There might be cases where there is no sensor or any collection of sensors that can satisfy the lower bound, which means that no sensor can cover the spatial region of the task (this case usually happens when the network is not dense or when the region of the task is very large). In this case the optimum solution is the set of sensor nodes whose sensing ranges intersect with the spatial region of the task.
4.3.2.1 Algorithm Evaluation

To the best of our knowledge, no other approach has taken the sensing region of sensors into account for task allocation. Thus, in order to evaluate our algorithm we decided to compare it to approaches for spatial allocation in other research areas. We considered spatial crowdsourcing as well as spatial task analysis for event detection in wireless sensor network (we found one approach that considers the region of an event to allocate sensors that can perform a task [158]).

Spatial crowdsourcing has looked into allocating tasks to users with the assumption that users are located at the same location with the task –basically the same longitude and latitude. For our comparison, we are taking a similar but more realistic algorithm: instead of assuming that sensors and tasks have exactly the same location (the same point in space), we assume that sensors are located inside the region of a task. The reason for modifying the assumption is because, with randomly generated tasks and assets, it is nearly impossible for tasks and sensors to have the same longitude and latitude, and thus we consider the distance from each sensor to the task. As a result, a spatial allocation algorithm would never find assets to contribute to the task.
The other approach, that we compare our algorithm with, was proposed for sensor network event detection. The approach considers regions to be defined by sensor locations, which is also not really realistic especially for task allocation because each task has its own unique characteristics including the space it occupies which is not related to other objects in the environment like the sensors. In order to detect an event, the algorithm tries to find all sensor nodes whose locations can define a minimum bounding rectangle for the region of the task –MBRs are usually used in spatial information systems as efficient approach for finding objects. Similar to other non-spatial sensor task allocation approaches, this approach considers sensors to be homogeneous in terms of their sensing and communication ranges and the tasks they can perform.

In order to evaluate the algorithm, an ontology is populated with randomly generated sensor nodes in a field with random locations and sensing ranges. Tasks are also generated randomly in different locations inside the field and with different regions. We run the algorithm and try to allocate the optimum set of sensors to the different tasks based on their ability to contribute to the task depending on their sensing capabilities –in this case their sensing ranges.

The algorithm is compared with both the MBR approach and the spatial crowdsourcing approach. In this algorithm the aim is to minimize the number of sensors assigned to a spatial task, and an optimum solution is assumed to find a minimum set –can be a single sensor– that can perform the task whenever that is possible. The result is compared to both considered algorithms (MBR and crowd sourcing).

Different scenarios are compared. They show the effect of changing node density in the network as well as changing the sensing ranges and the region of the task on the algorithm and on the new optimum solution.

The work focuses on networks with high node density –they are considered to be higher than that of the techniques found in literature. With a higher node density it is also possible to use a lower sensing range and preserve more energy. We vary the number of sensors, their regions and the regions occupied by the tasks, in order to study the effect of spatial properties on the allocation algorithm.

For all scenarios an SSN ontology is populated with sensor nodes randomly distributed within an area of 100 x 100 m² [159, 160] with various densities, the number of nodes usually starts with 200 following similar assumptions in literature [104], however, with the aim of studying the effect of nodes density on the allocation algorithm this number is increased in other scenarios. Sensor locations are defined following the OGC GeoSPARQL spatial data representation, and sensor ranges are generated with random shapes and radius. Tasks are also generated at random locations as well and random shapes within the same area.
We simulate our algorithm using Java, and Parliament. Parliament\textsuperscript{24} is a triple store –built on top of Java and Jena\textsuperscript{25}– that enables reasoning with GeoSPARQL about spatial relationships and functions.

### 4.3.2.1.1 Effect of Node Density

The first scenario evaluates the effect of density of the nodes on the selected algorithms, Table 4-1 describes the scenario’s parameters that we consider;

<table>
<thead>
<tr>
<th>Network area size (m(^2))</th>
<th>Number of nodes</th>
<th>Radius of Sensor (m)</th>
<th>Radius of task (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 x 100</td>
<td>200, 250, 300, 350,</td>
<td>5</td>
<td>1, 2</td>
</tr>
<tr>
<td></td>
<td>400, 450</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4-1 Simulation Parameters for Density Scenario

Figure 4-8 shows the effect of density of nodes on the task allocation algorithms. The figure illustrates the behaviour of each algorithm for the two radius of the task in Table 4-1. With the increase in the number of nodes in the field, it is more likely that more sensors would be able to contribute to the task. This is more obvious with the minimum bounding rectangle approach, as the algorithm tries to find all sensors that can cover a specific task, the higher the density of the network, the more sensors will be allocated for a task. Such an approach would consume a lot of energy by allocating the maximum number of nodes to a specific task, it also results in a lot of redundant data to be sent through the network.

![Node Density Effect](image)

Figure 4-8 Effect of node's density

\textsuperscript{24} http://parliament.semwebcentral.org/
\textsuperscript{25} https://jena.apache.org/
On the other hand, looking at the result of the crowdsourcing approach which targets sensors available at the same location of the task, we can see that the chances of this assumption to happen is very slim, and for most the tasks that were sent through the network, no sensor was found. Again with the increase of the density of nodes it becomes more likely that we find sensors that can perform a task. However, this cannot guarantee that the sensors located in the region of the task can fully cover the task. When comparing the two previous approaches (i.e. crowdsourcing and MBR) to our proposed algorithm, we find out that our algorithm GeoSMA can find a smaller set of sensors to be assigned than those found using the MBR algorithm. In many cases, the algorithm could find a single sensor that can cover the region of the task without the need for additional sensors. Even when one sensor could not be found, it could still outperform both the MBR algorithm and the crowdsourcing approach by finding a set with a smaller number of assets than the original set.

![Figure 4-9 Task Allocation Success Rate with Changing Node Density](image)

Figure 4-9 compares the success rate in finding a set of sensors that can cover a specific region of a task. GeoSMA and MBR have the same success rate because the starting point of our algorithm is to find the maximum number of sensors that intersect with the region of the task. Both algorithms outperform the crowdsourcing algorithm –as most of the time it cannot successfully find sensors that cover the region of the task, which proves that the assumption taken for spatial allocation is not realistic.

### 4.3.2.1.2 Effect of Radius of Nodes

Table 4-2 shows the simulation parameters for the second scenario, which evaluates the effect of node sensing radius on the allocation algorithms

<table>
<thead>
<tr>
<th>Network area size (m²)</th>
<th>Number of nodes</th>
<th>Radius of Sensor (m)</th>
<th>Radius of task (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 x 100</td>
<td>200</td>
<td>5, 6, 7, 8, 9, 10</td>
<td>1, 2</td>
</tr>
</tbody>
</table>
Figure 4-10 shows the effect of changing the sensing range on the algorithms. Similarly to changing the density, increasing the sensing range of the nodes means that more sensors can contribute to different missions. This effect can also be seen mainly on the MBR approach, as the algorithm tries to find the sensors whose aggregation of space can cover the MBR of the task. Increasing the sensing range has no effect at all on the crowdsourcing algorithm – unlike the increase in density – because it is based on the location of the sensor nodes only with no consideration of the sensing region. Still GeoSMA outperforms both algorithms, with the increase of sensing range it becomes more likely to find a single sensor that can perform the required task. Figure 4-11 also shows that the success rate of task allocation for crowdsourcing is worse the success rate achieved where the node density increases, which is realistic since the algorithm only uses the location of the nodes which does not change, unlike the GeoSMA and MBR.

**Sensor Range Effect**

![Sensor Range Effect](image)

**Figure 4-10 Effect of Sensing Range**

![Crowdsourcing vs GeoSMA/MBR](image)

**Figure 4-11 Task Allocation Success Rate with Changing Node Sensing Range**
4.3.2.1.3 Effect of Radius of Task

The last scenario that we consider is the effect of changing the size of the task region on task allocation. We consider changing the radius of the tasks from a very small radius where it is likely for a single sensor node with a large sensing area to be able to detect the task, to a bigger radius which are kept to a maximum equals the radius of the nodes; and in this case it becomes very unlikely to have one sensor that is capable of covering the whole area of the task. Table 4-3 defines the simulation parameters.

<table>
<thead>
<tr>
<th>Network area size (m²)</th>
<th>Number of nodes</th>
<th>Radius of Sensor (m)</th>
<th>Radius of task (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 x 100</td>
<td>200, 300</td>
<td>6</td>
<td>1, 2, 3, 4, 5, 6</td>
</tr>
</tbody>
</table>

Figure 4-12 shows the effect of the increase of task radius on the allocation algorithm. Unlike the previous two scenarios, increasing the radius of the task without increasing the density of the network or its sensing range results in poor performance of the algorithm. When a task has a large radius—a large area of space which the task spans over—the chance of finding a set of sensors that can cover the whole area becomes slim. This can be easily observed by analysing the Figure 4-12 below. We can notice that with the increase in task radius—and as a result the task region—both GeoSMA and MBR algorithm use the maximum number of nodes that can contribute to the task. When the task radius is 5 and 6, GeoSMA could not optimize the allocation of sensor nodes to the task and had to go with the maximum number of nodes that can cover the area. Crowdsourcing on the other hand, still finds a smaller number of nodes inside the region of the task compared to those found by GeoSMA and MBR. Still, the number of nodes in the solution provided by crowdsourcing also increases with the increase in task region because with a larger task region it is more likely to get a larger numbers of sensor nodes inside the region.
The other observation that can be made based on the simulation scenario is that with the increase of the task region, it becomes also less likely to find a set of sensors that can fully cover the region of the task. This effect can be found in Figure 4-13. With the increase in the task region the result gets even worse.
4.3.2.1.4 Process Evaluation

In addition to the simulation results that have been introduced in the previous section, this section evaluates one run of the algorithm by showing the areas occupied by the task, the assets that could contribute to the task and the optimal solution.

The simulation parameters are similar to some of those in the previous section. An ontology is populated with 350 temperature sensors with a sensing radius of 6 m, a task with a radius of 1 m is also randomly generated. The reason for choosing a small radius for the task is to allow less number of assets be able to contribute to the task to make the figure clearer, and the reason for a large radius of the operational range of assets is to increase the chances of having assets that can cover the region of the task.

The generated task and sensors that are chosen by both GeoSMA and MBR are compared, can be seen in Figure 4-13.

The radius of the mission is 1
The task is Polygon {(94.76 67.19, 95.63 66.69, 95.63 65.69, 94.76 65.19, 93.89 65.69, 93.89 66.69, 94.76 67.19)}

Nodes in the MBR are

The optimal solution of the specified task is
http://www.semanticweb.org/ontologies/2014/9/SpatialReasoning.owl#TemperatureSensor303

Figure 4-13 A Mission and its Solutions

Figure 4-14 shows a snapshot of the description of one of the sensors (TemperatureSensor303) from the generated ontology;

```
  <SpatialReasoning:hasRange>
      <geo:Polygon rdf:resource="http://www.opengis.net/ont/sf#MultiLiteral">
        <Polygon{(94.95 72.06, 100.15 69.06, 100.15 63.06, 94.95 68.06, 89.75 63.06, 89.75 69.06, 94.95 72.06)}"></Polygon>
      </geo:Polygon>
    </sf:Polygon>
  </SpatialReasoning:hasRange>
  <SpatialReasoning:hasPlacement>
    <sf:Point rdf:about="http://www.semanticweb.org/ontologies/2014/9/SpatialReasoning.owl#TemperatureSensorLocation303">
      <geosparql:asWKT rdf:datatype="http://www.opengis.net/ont/sf#MultiLiteral"/>
    </sf:Point>
  </SpatialReasoning:hasPlacement>
</sn:Sensor>
```

Figure 4-14 Sensor Description in Ontology

To evaluate of the algorithm, the result of running both GeoSMA and MBR are compared with what the outcome should be when finding the results by hand. Figure 4-15, shows the area of the task that was generated and all the assets that could contribute to it because their operational range intersect with the region of the task.
Chapter 4. Spatial Reasoning for WS&ANs Mission Assignment

4.3.3 Selfish Geospatial Sensor Mission Assignment algorithm (SGeoSMA)

While branch and bound algorithms are efficient in finding the optimum solution for optimisation problems, their time can grow exponentially with problem size, because they search the solution space for every possible combination of sensors. Our Branch and Bound algorithm (GeoSMA) have proven to find the best allocation of tasks to assets in the network. However, the tree size of the algorithm grows with the number of assets and also with size of the sensing and task range –because the initial solution space will increase–, resulting in an exponential increase in the algorithm time. In this section, a novel evolutionary algorithm that can find a near optimal solution to the problem based on the Selfish Gene theory of evolution is proposed and proved.

Figure 4-15 Mission and its Solutions

The Figure shows that the assets that could contribute to the mission (or task –can be seen in red) are those that were found by the MBR algorithm, and they all either intersect or cover the task. The Figure shows three possible assets that can cover the range of the task individually, these are: Sensor 303, Sensor 217 and Sensor 81. GeoSMA has chosen Sensor 303 (can be seen in yellow) as an optimal solution because when the algorithm finds a tie between two solutions regarding the number of assets, the best is chosen to be the one which occupies a smaller space.

The Figure shows that GeoSMA algorithm operates as expected and can produce a better outcome than MBR which does not try to optimise the solution.
4.3.3.1 The Selfish Gene

Genetic algorithms (GA) are search and optimisation algorithms that are proposed based on the Darwinian theory of evolution. They assume having a population of individual answers or solutions to problem (also called genotype or chromosome) and they try to optimise these answers/solutions by evolutionary processes, mainly: crossover and mutation. Under this assumption, the basic units of these algorithms are the individuals, and each individual is composed of a collection of genes (values that make up the genotype). Hence, the result of this optimisation problem is influenced largely by the initial population which is generated randomly most of the time. The dependency on the initial population for GA is one limitation of the algorithm, because, if a resource is not assigned to any task in the initial population, it is very likely that it will not be assigned to any task in the optimised solution. This can be eliminated by increasing the size of initial population which however has the side effect of increasing the computation time.

An alternative view into evolution comes from the work of biologist Richard Dawkins in his book The Selfish Gene [161], which describes a gene-centred view of evolution. Under the selfish gene theory, the basic unit of evolution is the gene rather than the chromosome or individual, which is unlike Darwin’s natural selection. The point of view of the Darwinian Theory—survival of the fittest—consider the fitness of an individual for its survival. However, following Dawkins theory, individuals do not survive, they only do so for some generations (e.g. children in most cases are only half of each parent’s chromosomes) but the genome of individuals survive evolution by replicating themselves into subsequent generations.

In Dawkins theory of evolution, genes compete to appear in the genotype of individuals; individuals are not important for the evolution process, they are rather just the carriers of the genes and since they are mortal, it is not their fitness that is important to the evolution process but the fitness of their genes. The Selfish Gene theory does not have any crossover or mutation processes because it is not based on a population of individuals and their fitness level, evolution happens by an unspecified kind of sexual reproduction. Individuals are only generated when needed from a Virtual Population (VP) which represents the “gene pool”. Every potential solution is encoded as a genotype where every gene have a location in the genotype called “locus” and a value called “allele”. The frequency of an allele for a given gene in the VP models its goodness and success in the genotype. Fitness, however, is still calculated on the individual level.

The Selfish Gene (SG) algorithm was first introduced by Conrno, et al. [162]. They introduced an abstract model for virtual population where the number of individuals and their identities are not important. Since individuals are not listed explicitly, Conrno, et al. introduce reproduction through its effects on the VP, where a tournament—based on their fitness for the objective function—between
two chromosomes in every generation affects the VP for the next generation. Since SG models the gene pool and generate genomes from the pool, each genome tends to be unique because of the large number of possible combinations, however, some alleles might be more frequent than others in the VP.

The algorithm is specified as follows: let there be $g$ number of genes in every genotype, occupying the locus $L_i$ ($i = 1, 2 \ldots g$). Each locus can be occupied by $n_i$ different alleles, $a_{ij} = (a_{i1}, a_{i2}, \ldots a_{in_i})$ is the list of alleles that can be assigned to locus $i$ where different loci can be allocated with different numbers of alleles, and finally let $p_{ij}$ be the probability for $a_{ij}$. The steps for SG algorithm as introduced by Conron, et al. can be seen in Figure 4-17.

Initializing the genotype is done by choosing one allele to be placed in each locus using its probability, which is updated following the outcome of the tournament (competing between the two chosen genotypes based on their fitness), by rewarding the winner genome and penalizing the loser. The functions used for rewarding and penalizing genes can be seen in the formulas (4-3, 4-4):

$$P_{ij} = P_{ij} + \epsilon_i \quad \text{for each Locus } i \text{ in genome } H \quad (4.3)$$

$$P_{ij} = P_{ij} - \epsilon_i \quad \text{for each Locus } i \text{ in genome } H \quad (4.4)$$

Where $\epsilon_i$ is a constant, which is usually the same for all loci.

The use of the reward and penalize function affects the probability of the genes and as a result their chances to be picked in the next generation, and so the alleles of the winner genome increases their chances of survival and derives a better algorithm solution. Individual selection is detailed in Figure 4-16. To introduce further variability to the chromosomes, mutation can happen in which case the alleles are chosen randomly for some locus.

![Figure 4-16 Selfish Gene Individual Selection](image)
4.3.3.2 Algorithm Implementation

When implementing the SGeoSMA—similar to GeoSMA—the aim is to minimize the number of sensor/actuators assigned to a specific task. Furthermore, like GeoSMA we use the spatial relations between sensors and tasks in order to allocate sensors to missions. We consider the same goal of GeoSMA, to find the minimum number of nodes that can perform the specified task by the end user, for that we define the fitness function to be a set of sensors that can cover the area of interest and the aim is to minimise the fitness function. We define the gene to be a sensor and the different values it can have, in this case 0 if the sensor is not a part of the mission and 1 otherwise. We initialise the virtual population to be the set of all sensors that can answer the mission and their
different values. In every generation, we choose just one genotype, when choosing a genome, each gene is chosen using roulette selection; we first compare it to the area of the task, if it can cover it we reward the genes of the genotype, else we penalise them. The genotype is then compared to the current best genotype and the best one is chosen to compare with next generation, the comparison between two genomes is a comparison of the space they occupy, a better solution is a genome whose area is contained in the other genome as it results in a smaller task allocation region. The steps of SGeoSMA algorithm can be seen in Figure 4-18.

<table>
<thead>
<tr>
<th>Selfish Geospatial Sensor Mission Assignment (SGeoSMA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>initiate all $p_{ij} \leftarrow \frac{1}{n_i}$</td>
</tr>
<tr>
<td>Select genotype Best</td>
</tr>
<tr>
<td>while (Satisfy criteria = false) do</td>
</tr>
<tr>
<td>Select genotype G1</td>
</tr>
<tr>
<td>if (Fitness(G1) Covers Region(Task)) then</td>
</tr>
<tr>
<td>reward G1</td>
</tr>
<tr>
<td>if (Fitness(Best) Covers Fitness (G1)) then</td>
</tr>
<tr>
<td>Best = G1</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>penalise G1</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>return Best</td>
</tr>
</tbody>
</table>

Figure 4-18 SGeoSMA Protocol

### 4.3.3.3 Algorithm Evaluation

SGeoSMA is compared with the previously introduced GeoSMA algorithm. Similar to the previous evaluation’s parameters, we populate the ontology with sensors in a field of 100m x 100m. The locations of the sensors and their sensing ranges as well as the regions occupied by the tasks are randomly generated in the field and are modelled following the OGC GeoSPARQL spatial data representation. Both the regions of the tasks and the ranges of the sensors have random shapes modelled according to GeoSPARQL. We compare the convergence percentage of the algorithm as well as the time it takes for the algorithm to reach an optimal or near optimal solution. Convergence percentage reflects the percentage of times the algorithm can reach an optimal solution based on the total number of runs—in this case 10 [163]. The simulation parameters and the simulation results
of the different scenarios considered can be found in Tables 4-4, 4-5, and 4-6, respectively. $S_r$ refers to the radius of the sensor’s sensing range and $T_r$ refers to the radius of the task region—the radius is only used when generating spatial region property to influence the area they occupy or can detect for sensors. For SGeoSMA, we define the following parameters as well: mutation rate $p_m = 0.4$, reward/penalise factor $\varepsilon_i = 0.01$, and the number of generations is 2000, following similar assumptions in the literature [44, 162, 163]. We need to point out that even though we set the number of sensors in the range 200-500, we are not considering the case of all sensors being able to perform the task. As it is already described in our approach, the algorithm first tries to find an appropriate set of sensors that can contribute to a task—we call this set the solution space—then tries to optimise it.

Table 4-4 Simulation Results - Node Density Effect

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Number of sensors in solution space</th>
<th>Algorithm</th>
<th>Percentage of convergence</th>
<th>Time – sec (Std deviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of Sensors = 200</td>
<td>3</td>
<td>GeoSMA</td>
<td>100%</td>
<td>2.732 (1.915)</td>
</tr>
<tr>
<td>$S_r = 5$</td>
<td></td>
<td>SGeoSMA</td>
<td>90%</td>
<td>43.945 (9.563)</td>
</tr>
<tr>
<td>$T_r = 2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num of Sensors = 300</td>
<td>6</td>
<td>GeoSMA</td>
<td>100%</td>
<td>12.743 (6.324)</td>
</tr>
<tr>
<td>$S_r = 5$</td>
<td></td>
<td>SGeoSMA</td>
<td>90%</td>
<td>51.523 (11.36)</td>
</tr>
<tr>
<td>$T_r = 2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num of Sensors = 400</td>
<td>7</td>
<td>GeoSMA</td>
<td>100%</td>
<td>112.777 (81.508)</td>
</tr>
<tr>
<td>$S_r = 5$</td>
<td></td>
<td>SGeoSMA</td>
<td>80%</td>
<td>46.457 (9.353)</td>
</tr>
<tr>
<td>$T_r = 2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num of Sensors = 500</td>
<td>9</td>
<td>GeoSMA</td>
<td>100%</td>
<td>4847.545 (6224.5)</td>
</tr>
<tr>
<td>$S_r = 5$</td>
<td></td>
<td>SGeoSMA</td>
<td>80%</td>
<td>52.467 (2.96)</td>
</tr>
<tr>
<td>$T_r = 2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The percentage of convergence is always compared to Branch and Bound algorithms because they can always find the optimal solution, in this case GeoSMA. A percentage of convergence of 100% means that all 10 tests could find the optimal solution. Results from Tables 4-4, 4-5 reflects the effect of problem size on the performance of both algorithms. It should be noted that in tables 4-4, 4-5, we omit the cases where there is no sensor or a single sensor in the field to contribute to the task, because then there is no problem to optimise. For the same reason, we also omit the cases where the solution space cannot cover the task region; however this will be discussed in more detail.
in the results of Table 4-6. Results from tables 4-4 and 4-5 show that the time for GeoSMA to converge to the optimal solution increases with the increase in the initial number of sensors that can answer the task. This increase is proportional to the number of nodes in the network and their sensing range, because it means that more sensors are capable of contributing to the mission, which as a result, increases the search space and the generated search tree for finding the optimal solution.

Finally, it can be noted that when the problem size is relatively small GeoSMA can find the optimum solution in a shorter time than SGeoSMA since the latter is related only to the number of generations. However, as the problem size increases, the time for GeoSMA increases because it searches all the solution space for a possible solution. Table 4-4 shows that while SGeoSMA have a very similar computation time for any problem size, the average processing time of GeoSMA increases exponentially with the solution space. While the maximum solution space that was considered in the algorithm is 9, which is considered as a relatively small number of sensors in the solution space, if that space increases to 10 the processing time for GeoSMA can increase to an average of 15513.825 second, as can be seen in Figure 4-19.

### Table 4-5 Simulation Results - Sensing Range Effect

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Number of sensors in solution space</th>
<th>Algorithm</th>
<th>Percentage of convergence</th>
<th>Time - sec (Std deviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num of Sensors = 200</td>
<td></td>
<td>GeoSMA</td>
<td>100%</td>
<td>1.476 (0.505)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SGeoSMA</td>
<td>80%</td>
<td>39.279 (9.314)</td>
</tr>
<tr>
<td>S&lt;sub&gt;r&lt;/sub&gt; = 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T&lt;sub&gt;r&lt;/sub&gt; = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num of Sensors = 200</td>
<td></td>
<td>GeoSMA</td>
<td>100%</td>
<td>12.537 (16.845)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SGeoSMA</td>
<td>80%</td>
<td>42.432 (16.529)</td>
</tr>
<tr>
<td>S&lt;sub&gt;r&lt;/sub&gt; = 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T&lt;sub&gt;r&lt;/sub&gt; = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num of Sensors = 200</td>
<td></td>
<td>GeoSMA</td>
<td>100%</td>
<td>190.933 (197.672)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SGeoSMA</td>
<td>80%</td>
<td>49.964 (10.328)</td>
</tr>
<tr>
<td>S&lt;sub&gt;r&lt;/sub&gt; = 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T&lt;sub&gt;r&lt;/sub&gt; = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Num of Sensors = 200</td>
<td></td>
<td>GeoSMA</td>
<td>100%</td>
<td>378.6464 (267.442)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SGeoSMA</td>
<td>70%</td>
<td>44.128 (6.222)</td>
</tr>
<tr>
<td>S&lt;sub&gt;r&lt;/sub&gt; = 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T&lt;sub&gt;r&lt;/sub&gt; = 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figures 4-19, 4-20 better demonstrate the effect of the increase in solution space on computation time for both algorithms;
The results show that GeoSMA—even though it can find the optimal solution—is not suitable for real time applications especially in a future where sensors will be everywhere around performing all various tasks. SGeoSMA on the other hand could find a near optimum solution and an optimum one in many cases in a much shorter time and thus is the more suitable one for UbiComp applications.
Figure 4-21 compares the percentage of sensors in the final solutions for GeoSMA, SGeoSMA, MBR and crowdsourcing (CS) averaged over 10 runs for every solution space. Results show that both GeoSMA and SGeoSMA outperform both approaches in the literature; they can always find a better solution—less number of sensors—than using the full set of sensors that can perform the task which is provided by using MBR. They can also find a solution that can perform the task unlike CS as they do not propose unrealistic assumptions similar to CS. Even though GeoSMA can outperform SGeoSMA in terms of numbers of nodes in the final solution, it can be much slower as already described in previous results.

Figure 4-21 Allocation Algorithms Comparison with different Solution Space
Table 4-6 is different from the two previous comparisons. The aim here is to compare the effect of increasing the region of the task on the allocation algorithms. While previous research has ignored the effect of the region of tasks when allocating sensors to the task, results in the table show that increasing the region of the task results in less ability for the sensors cover to the whole task area even when considering all sensors that can contribute to the task. This result shows that it is possible for the user’s task to exceed the capabilities of the sensors in the network, in which case a collection of sensors need to collaborate to perform the task. In these cases, we do not try to optimise the collection of the sensors that can perform the task, with the aim of trying to cover as much as possible from the region of the task. When the algorithms could find a solution space that can cover the region of the task, then they try to optimise it using the two proposed algorithms. Results in this case regarding the computation time cannot be generalised because of the little number of successful solution Spaces, and because in many cases the algorithm will terminate before visiting all possible solutions since the region of task is much larger than the sensing range of a sensor, i.e. a single one sensor cannot perform the task.
4.4 Conclusion

In this chapter, we targeted the problem of sensor mission assignment from a spatial perspective. We aim at allocating sensor nodes to a task according to their spatial properties and the spatial relationship with the region of the task. To accomplish that, the first step is to represent the spatial properties of both sensor nodes in the network as well as the tasks. We extend the semantic sensor network ontology with spatial properties that are integrated using GeoSPARQL ontology. Having modelled and represented the spatial properties of the sensors and the task, we are able to reason about their spatial relationships as a base towards finding the set of sensors that can contribute to a specific task in the system in a near optimal set of resources.

We present and prove two allocating algorithms for sensor mission assignment, GeoSMA based on the Branch and Bound algorithm, and SGeoSMA based on the Selfish Gene theory of evolution. The aim of both algorithms is to find the optimal set of sensor nodes that can participate to a task requested by the user of the system. For both algorithms, we assume the optimal set to be the set with the minimum number of sensor nodes that can cover the region of the task. To evaluate our algorithms, we compare them to the only two approaches that consider the spatial properties of resources and tasks for task allocations. These are: spatial crowdsourcing and event detection in WSNs based on Minimum Bounding Rectangles (MBR). MBR is also the only approach that considers the task to occupy a region of space rather than one point. The results of the simulation shows that our algorithm outperforms these two approaches. In many cases, especially with a high density of nodes, the algorithms can find a single node to perform the task rather than a large set of nodes. However, with the increase in the problem size, the time of GeoSMA increases exponentially, thus we have proposed SGeoSMA, an evolutionary algorithm for spatial task allocation in WS&ANs. We compare the new algorithm with GeoSMA, the results shows that SGeoSMA can find a near optimal solution –in most cases an optimal one– to the task allocation problem in shorter time compared to GeoSMA when the problem size is large. In the results, we have also studied the effect of changing the spatial properties of sensors and the task on the allocation algorithms. Results showed that while increasing the sensing radius of the sensors would result in more sensors being capable of detecting the task, which we then optimise to a smaller set with our algorithm; increasing the spatial region occupied by the task makes it less likely to have a set of sensors capable of performing the specified task.

Regardless of the importance of the problem of spatial mission assignment in WS&AN, which has been receiving an increasing interest, the spatial properties of sensors and tasks are not the only factors that can affect allocation algorithms. Other factors, like the remaining energy of the sensor node as well as the observation accuracy, must also be considered to guarantee high utilization of the network, especially with the existence of more than one task competing for the same resource.
The problem of multiple task allocation is considered in Chapter 5 and an extended algorithm is presented. Another important aspect of mission assignment in WS&AN that we have not considered in this chapter is the allocation of multiple missions to assets in the network where missions have different importance and can compete for the available resources in the network.

In the next chapter, we extend the problem definition of sensor mission assignment to consider allocating competing missions to sensors in the network and to consider the remaining energy of the nodes to better utilize the performance of the network.
Chapter 5

5 Spatial Weapon Target Assignment for Mission Assignment in Wireless Sensor & Actuator Networks

The spatial allocation algorithms proposed in the previous chapter (GeoSMA, SGeoSMA) deal with one task arriving at the network at a time. They also only consider the spatial properties of assets and tasks like their location, the space they occupy, and functional regions, as the factors that affect the mission assignment problem. The problem arises when more than one task arrive at the network; the challenge is to find the best allocation of sensors to the various tasks and optimize the performance of the network as a whole. While Branch and Bound algorithms are efficient in finding the optimum solution, we have shown how their computation time can grow exponentially with response to the problem size even though the problem size is still relatively small with only one task arriving at the network and with less than 10 available resources (sensors) to perform it. The genetic algorithm we propose to overcome this limitation of GeoSMA was able to find an optimal or near optimal solution faster than Branch and Bound based algorithms.

A realistic approach to resource allocation in WSNs, which would be used in future UbiComp applications, would not only consider a spatial based allocation algorithm but also more factors that can lead to a better utilization of the network and its components. Such factors can include information about different properties of assets and tasks in the network, such as the energy of the nodes and the importance of the task. With more than one task arriving at the system, the problem size increases and tasks might have to compete for the usage of specific assets in the network (see Figure 5-1) depending on the priorities dictated by those factors.

In this chapter, an extended selfish gene algorithm is presented and proven in order to solve the problem of multiple sensor mission assignment problem with the goal of improving the performance of the network by maximizing its utility while reducing the number of nodes assigned to missions to preserve their energy. Our approach is inspired by the weapon target assignment optimization problem in the field of military operation research.
5.1 The Weapon Target Assignment Problem

A general description of the WTA assignment has been presented in Chapter 2. In this chapter we look more into the mathematical formulation of the problem and its use as a base for resource allocation in WSNs.

5.1.1 Weapon Target Assignment Problem Formulation

The mathematical formulation of the WTA problem is the following [110]: Let \( T \) be a set of targets with indices 1, 2, 3, \( \ldots \), \( T \), and let \( W \) be a set of weapon types with indices 1, 2, 3, \( \ldots \), \( W \). Every target \( T_j \) in the set of targets has a corresponding value \( V_j \) that represents the importance of the target. Let \( AW_i \) be the number of available weapons of type \( i \) to be assigned to targets, and let \( p_{ij} \) be the probability of destroying target \( j \) by assigning a weapon of type \( i \) to it. The probability of survival of a target \( j \) after assigning \( x_{ij} \) number of weapons of type \( i \) to it is given by \( q_{ij}^{x_{ij}} \), where

\[
q_{ij} = 1 - p_{ij}
\]  

(5.1)

A target can be assigned more than one weapon at the same time, and it can be assigned weapons of different types as well. The WTA problem aims at finding the number of weapons of type \( x_{ij} \) to be assigned to a target \( j \) with the aim of minimizing the survival chance of all targets.
The problem is formulated as:

\[
\text{Minimize } \sum_{j=1}^{T} V_j \left( \prod_{i=1}^{W} q_{ij} \right) \quad (5.2)
\]

Subject to

\[
\sum_{j=1}^{T} x_{ij} \leq AW_i \text{, for all } i = 1, 2, \ldots, W \quad (5.3)
\]

\[
x_{ij} \geq 0 \text{ and integer for all } i = 1, 2, \ldots, W \text{ and for all } j = 1, 2, \ldots, T \quad (5.4)
\]

In the formulation, it can be seen that the aim is to minimize the survival expectation of targets while insuring that only the available number of weapons of a specific type (and not more) are used. Other restrictions can be added to the problem, which can influence the allocation algorithm, such as an upper and a lower bound on the number of assets of type \(i\) that can be assigned to a target \(j\), or an upper and lower bound on the total number of assets that can be assigned to a target, or even bounds on the survival expectation of the targets in the network.

Different algorithms have been proposed aiming at finding the near optimal solution for weapon target assignment like [45, 110, 112]. However, similar to the mission assignment problem that was considered in ISR, the proposed algorithms do not consider the spatial properties of the different assets and tasks in the system. All approaches only consider other properties, such as the level of damage the weapon can cause to the target.

In the light of those limitations, the aim is to introduce a new algorithm for allocating sensors to the different tasks in the network to optimize the use of the network by taking into consideration their spatial properties as well as their energy consumption and the importance of tasks. The approach taken formulates the mission assignment problem as a bi-level optimisation problem aiming at efficiently utilizing the use of the network when assigning sensor nodes to the tasks, while at the same time trying to preserve its energy consumption.

### 5.2 Weapon Target Assignment Approach to Mission Assignment in Wireless Sensor Networks

In the previous chapter, GeoSMA and SGeoSMA were presented; two algorithms for mission assignment that aim at finding an optimum solution to the problem of spatial task allocation in
WSNs, finding the optimal solution was considered to be finding the minimum set of sensors that can cover the region of the task. Optimising resource allocation in the network using both algorithms can help optimize the use of the network by omitting redundant resources and only assigning the most suitable sensors to the task. This approach preserves more resources (sensors/actuators) to perform other tasks that might arrive at the network.

In wireless sensor networks, due to their characteristics, different factors can play a role in deciding the benefit and detriment of using specific sensors for a task such as the energy of nodes, their sensing ranges, and the importance of the tasks, etc. In order to optimize the performance of the network, the problem of mission assignment in WSNs is formulated similar to that of WTA and then further extended with additional restrictions.

By analogy, we consider the sensor nodes in the network to be the weapons in the network; e.g., temperature sensor, pressure sensor, etc. The targets are the tasks required by the end user that make use of sub-sets of those sensors. Let $s_1, s_2, \ldots, s_n$ be the number of sensors available in the network which can be assigned to one of the tasks. Let $t_1, t_2, t_3, \ldots t_m$ be the number of tasks required by the end users, and every task in the network has a profit value $V_j$ which represents the value of performing the task. $P_{ij}$ is the probability that a sensor $S_i$ can perform the task $T_j$. In order to allow addressing the energy constraints of a WSN in the problem formulation, we add an energy-based constrain to the problem formulations. Let $E_{ij}$ be the energy required by sensor $i$ to perform the task $T_j$.

$E_{ij} = e_c + e_t$  \hspace{1cm} (5.5)

where $e_c$ is the required energy to perform the computation of the user’s task $T_j$ and $e_t$ is the energy required to transmit its own data via radio. Because we are not looking into routing algorithms but only into the task allocation, we are not considering the sensor nodes that are only assigned to perform routing of other sensors data [104]. We assume that every sensor can be assigned to one task but every task can have more than one sensor assigned to it. The formulation of the problem becomes the following:

$$Maximise \sum_{j=1}^{m} V_j \left(1 - \prod_{i=1}^{n} (1 - P_{ij})\right).$$  \hspace{1cm} (5.6)

Subject to;

$e_i \geq (e_c + e_t)$ for all $S_i = 1, 2, \ldots, n$,  \hspace{1cm} (5.7)

$\sum_{i=1}^{n} x_{ij} \leq 1 \hspace{1cm} \forall j = 1, 2, \ldots, m$  \hspace{1cm} (5.8)
Chapter 5. Spatial WTA for Mission Assignment in WS&ANs

Where $e_i$ refers to available energy at the node $i$ making the first restriction. This means that the available energy of the node is bigger or equal to the necessary energy to perform the sensing and then to route the data. $x_{ij}$ refers to the assignment of a sensor of type $i$ to the task $T_j$, where $x_{ij} = 1$ means that the sensor $S_i$ is assigned to the task $T_j$. Thus, the third restriction means that each sensor ($S_i$) is assigned to only one task but it is possible for some sensors to not be assigned to any task. The reason that some sensors might not be assigned to any task is that the tasks in their sensing range might be assigned to other sensors and they cannot contribute to other tasks. Current approaches for WTA consider the simple WTA assignment problem, in which the third constraint (5.8) is restricted to allow at least one weapon to be assigned to every task, the restriction can be found in formula (5.9):

$$\sum_{i=1}^{n} x_{ij} = 1 \quad \forall j$$  \hspace{1cm} (5.9)

Before targeting the problem of mission assignment from a spatial perspective, we compare the current approach in the literature [163] based on a genetic algorithm with ours (selfish gene algorithm) for solving the problem of weapon target assignment.

Evolutionary Optimization (EO) procedures have been already successfully implemented as a valid approach for solving the WTA problem using Genetic Algorithm (GA) [163-165]. In the next section we introduce the selfish gene algorithm for WTA.

5.2.1.1 The Selfish Gene Algorithm for Multiple Task Allocation

The selfish gene algorithm has already been described in the previous chapter. However, the algorithm as described has a limitation: as the algorithm evolves, the probability of some alleles to appear in a locus increases while the probabilities of others decrease. This results in the algorithm evolving to a local optimal solution. This is not the case for spatial allocation, as only one task is considered and rewarding/ penalizing the genes and as a result the evolution of the chromosome is based on spatial properties. However, when considering more missions in the network it is more likely that SG would evolve to a local optimum, and thus, the algorithm needs to be iterated multiple times to explore various local optimal and find the best solution. This enhanced algorithm is called the multi-star selfish gene algorithm. In every iteration, a new seed is randomly chosen and the selfish gene algorithm is run again. In each new run, the reward/penalize factor is reduced to allow some variety in the search process, all the other factors in the algorithm are reset and the selection probability is set back to the initial value.
Another effect of how the algorithm evolves and increases allele probabilities is that when the algorithm reaches a local optimal solution, it is less likely for the chromosome to evolve much better. Therefore a different stop criterion than the number of generations is used. This is called the steady-state test. In this test, once every locus of the genome has at least one allele with a selection probability bigger than a predefined value, the algorithm stops as it is not likely to evolve to a better solution than the current one. The steady-state formula can be seen in Equation 5.10:

$$\forall i, \max(a_{ij}) > P_i$$  \hspace{1cm} (5.10)

where \(a_{ij}\) is the probability of allele \(a_{ij}\) to appear in the genome and \(P_i\) is the steady state factor; it is suggested that a value bigger than 0.95 is a sufficient value for the steady state factor [162].

The multi-star selfish gene algorithm used for WTA problem can be seen in Figure 5-2.

5.2.1.2 Implementation

For implementing the algorithms, we consider the genomes of the chromosome to be the combination of a weapon/task, where the locus represents the weapons and the allele in every locus represents the task that is assigned to the specific weapon. For example the genome (4 2 1 1 5) means that task 4 was assigned to sensor 1, task 2 to sensor 2, task 1 to both sensors 3 and 4, etc. Unlike other approaches for WTA that consider that every weapon is able to perform all tasks in the network, we accept that some sensors cannot perform the task (this can be due to their spatial properties or energy consumption which would be detailed more later). We perform that by assuming that the detection probability (equivalent to the kill probability) of a sensor to a task that it cannot perform is 0. The two variations of genetic algorithms that were described in Chapter 2 as well as the multi-level selfish gene algorithm are implemented and compared in the following:
5.2.1.2.1 Initialisation

The initialisation step is different between the genetic algorithm and the selfish gene algorithm as the former one operates based on an initial population while the latter does not. The initialization step for the genetic algorithm randomly generates a population of chromosomes which form the potential initial solution space for the assignment problem. As for the selfish gene algorithm, when the algorithm starts, the first step is to find the set of tasks that every sensor can contribute to and their detection probability. The selection probability of the tasks to the sensors which is different from the detection probability is initialised at this step and is \( \left( \frac{1}{\sum_{i=1}^{n} n} \right) \) where n is the number of tasks that the sensor can contribute to. While the detection probability of sensors does not change and is used to measure the fitness of a specific genome according the objective function, the selection probability changes throughout the different generations of the algorithm to give the task with highest probability more chances of being chosen for reproduction.

5.2.1.2.2 Selection

For selecting alleles of individuals (which are tasks for different sensors) during the tournament we use the Roulette Wheel Selection (RWS). The selection probability of a gene or an individual P(i) is calculated with the Equation (5.11); where n represents the total number of allele that can occupy a locus, and it is the number of tasks as we assumed that all sensors can contribute to all tasks, and f(i) is the fitness is the fitness of sensor i (in this case the selection probability).

\[
P(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}
\]  

(5.11)

5.2.1.2.3 Reproduction

Reproduction for genetic algorithm happens using the crossover and mutation operations to generate offspring that were previously described, while reproduction for selfish gene happens implicitly by changing the selection probability of different alleles in every generation following a tournament of two genomes as explained in the previous chapter.

The genetic algorithm evolves for a predefined numbers of generations while the selfish gene algorithm iterates for a specific number of iterations and in each of them evolves until the steady-state condition is met.

5.2.1.3 Simulation

We compare the multi-star Selfish Gene algorithm to a GA with One Point Crossover and EX crossover because GA is the most popular algorithm to solve the WTA. New research keeps
amending and introducing new operators to the algorithm. We implement the algorithms using Java 1.7, and perform our test on a 3.2 GHz processor with 8GB RAM PC. We identify different scenarios with different parameters based on the existing scenarios introduced for GA weapon target assignment, like [44, 163]. The parameters are the mutation probability $P_m = 0.4$, the number of initial population (which is the maximum number between the numbers of targets and weapons considered), and the number of generations (which is 2000). Tasks are generated with random importance in the range $[0, 100]$, and the probability of sensors to detect the task is a randomly generated between $[0, 1]$ to reflect the percentage of the detection probabilities which can also be 0 if the sensor cannot perform the task. The reward and penalise factor for the Selfish Gene (SG) algorithm is $\epsilon_i = 0.3$, and the steady-state factor $p_i = 0.96$. The first result compares 10 runs [163] of the three algorithms with different scenarios, and the average results are reported. The number of iterations for the selfish gene is set to be 15. The reason for the choice of 15 is because the algorithm was run for different numbers of iteration up to the problem size and it was noted that, when the number iterations increases (above 20), it results in an increase in the computation time of the algorithm. This is because with every iteration the reward/penalize functions are reduced and as a result it takes longer to reach a local optimum in every iteration. The increase in the fitness of the solution was not balanced with the improvement in the fitness of the solution, thus we found that keeping the number of iterations at 15 was enough to reach a near optimal solution. For the three algorithms, the same set of targets with their value or importance as well as the sensing probabilities of the sensors for each task are randomly generated, allowing some sensors not to be able to detect some tasks. Both genetic algorithms use the same initial population to evolve to the optimal solution.

In Table 5-1, we report the best fitness value, the average fitness value, and the CPU time for each of the algorithms (along with the standard deviation). $S$ reports the number of sensors and $T$ is the number of targets in every scenario.
Table 5-1 Simulation Results for Randomized Data Scenario. Results are averaged over 10 trials

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Algorithm</th>
<th>Best Fitness</th>
<th>Avg Fitness (Stdev)</th>
<th>CPU Time (sec) (Stdev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S = 6</td>
<td>GA (OPC)</td>
<td>351.484</td>
<td>314.369 (23.3)</td>
<td>&lt;1</td>
</tr>
<tr>
<td>T = 6</td>
<td>GA (EX)</td>
<td>351.484</td>
<td>328.152 (25.66)</td>
<td>&lt;1</td>
</tr>
<tr>
<td></td>
<td>SG</td>
<td>349.802</td>
<td>349.175 (1.139)</td>
<td>&lt;1</td>
</tr>
<tr>
<td>S = 10</td>
<td>GA (OPC)</td>
<td>487.705</td>
<td>479.206 (30.578)</td>
<td>&lt;1</td>
</tr>
<tr>
<td>T = 10</td>
<td>GA (EX)</td>
<td>523.026</td>
<td>458.843 (26.768)</td>
<td>&lt;1</td>
</tr>
<tr>
<td></td>
<td>SG</td>
<td>529.643</td>
<td>527.443 (2.68)</td>
<td>&lt;1</td>
</tr>
<tr>
<td>S = 20</td>
<td>GA (OPC)</td>
<td>997.737</td>
<td>973.590 (25.107)</td>
<td>7.056 (0.147)</td>
</tr>
<tr>
<td>T = 20</td>
<td>GA (EX)</td>
<td>966.434</td>
<td>913.777 (25.466)</td>
<td>6.953 (0.114)</td>
</tr>
<tr>
<td></td>
<td>SG</td>
<td>997.328</td>
<td>975.3581 (13.428)</td>
<td>7.934 (1.55)</td>
</tr>
<tr>
<td>S = 50</td>
<td>GA (OPC)</td>
<td>1988.296</td>
<td>1881.393 (60.324)</td>
<td>604.795 (1.631)</td>
</tr>
<tr>
<td>T = 50</td>
<td>GA (EX)</td>
<td>1719.087</td>
<td>1663.137 (33.43)</td>
<td>600.735 (2.554)</td>
</tr>
<tr>
<td></td>
<td>SG</td>
<td>2473.786</td>
<td>2390.862 (42.804)</td>
<td>358.591 (36.8617)</td>
</tr>
</tbody>
</table>

The results in the table show that using Selfish Gene algorithm to solve the Sensor Mission Assignment can better utilize the use of the network than Genetic Algorithm with both crossover operators do. As the problem size increases, the performance of the selfish gene improves more than the regular genetic algorithm with a shorter processing time. The reason for that is the selfish gene algorithm does not depend on a population of individuals which are regenerated in every generation. With the increase in problem size, without increasing the number of generations or the initial population, the GA would most likely not be able to find a near optimal solution. This can be overcome by increasing the population of the algorithm, however, this increases the computation time of the algorithm.

Table 5-2 compares the algorithms for a larger solution space, the stop criteria for a bigger solution space is the time of operation rather than the number of generations. We run the three algorithms for one hour and compare the fitness –the benefit– of the allocation algorithms.
Table 5-2 Average Fitness Value after One Hour Run
Results are Averaged over 10 Trials

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Scenario</th>
<th>Average Fitness (Stdev)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S = 80</td>
<td>T = 80</td>
</tr>
<tr>
<td></td>
<td>S = 100</td>
<td>T = 80</td>
</tr>
<tr>
<td></td>
<td>S = 120</td>
<td>T = 80</td>
</tr>
<tr>
<td>SG</td>
<td>3282.212 (24.408)</td>
<td>3588.346 (28.82025)</td>
</tr>
<tr>
<td>GA (OPC)</td>
<td>2565.531 (55.773)</td>
<td>2791.187 (72.9842)</td>
</tr>
<tr>
<td>GA (EX)</td>
<td>2253.418 (37.141)</td>
<td>2479.26 (54.44545)</td>
</tr>
</tbody>
</table>

The results also show that the selfish gene algorithm outperforms the genetic algorithm when the solution space is large. The reason for that is the dependency of the genetic algorithm on the number of population as well as the number of generation which was already discussed in the previous results.

The fitness curve can also be seen in Figures 5-3, 5-4. They compare the progression of all three algorithms throughout generations through two different scenarios. Figure 5-3 considers a scenario with 50 sensors and 50 targets, while Figure 5-4 considers 20 sensors with 20 targets. It can be seen the selfish algorithm outperform the two other genetic algorithms and can reach a better gain of the objective function. It can also be seen that the selfish gene algorithm can reach the best fitness much faster than both genetic algorithms as the problem size increases (Figure 5-4). The graph also shows the operation of the iteration of selfish gene where at every iteration a new seed is randomly picked and the algorithm evolves starting from the new seed. It also shows how the length of every iteration increases with the number of iterations because the penalise/reward function reduces to allow versatility in the solution but also makes the algorithm take longer to find the current optimal solution.
Figure 5-3 Fitness Curve for $S = 50$, $T = 50$. Results are averaged over 10 Trials.

Figure 5-4 also shows that, for the selfish gene algorithm, the best solution can be reached at any iteration and not necessarily at the last one. This is because the algorithm evolves into a local optimum and thus the need for iteration.

Figure 5-4 Fitness Curve for $S = 20$, $T = 20$. Results are averaged over 10 Trials.
Selfish gene algorithm outperforms current state of the art for WTA in terms of fitness and processing time especially when considering spatial constraints making it more suitable for WSNs. Implementing the restriction on sensor’s ability to perform specific tasks using a genetic algorithm would require either: i) a special crossover or mutation operators which make sure that no task is allocated to a sensor that cannot perform it or ii) a correction mechanism for every generation to fix such a mistake in the allocation. This problem does not happen when using the selfish gene algorithm because every locus of the gene in the genome can only be occupied by the allele that can be in that specific locus. This means that when implementing the selfish gene algorithm for resource allocation, only tasks that the sensor can perform can be allocated to that sensor.

5.3 Multi-Objective Optimization for Sensor Mission Assignment

The formulation of the weapon target assignment problem is based on increasing the probability of destroying the target assets or minimising the destruction of friendly ones, and so the destruction probability of each asset is the main element for weapon allocation. In WSNs, the aim is to find the set of sensor nodes that can perform a specific task –similar to the WTA problem. However, WSNs are expected to operate for a long period of time even without energy resources. An allocation algorithm for WSNs should take such a factor into consideration. Thus, formulating the problem of mission assignment extends that of weapon target assignment to cover all these characteristics and restrictions.

We have already formulated the problem of SMA similar to that of WTA in the previous section. In Chapter 4, we have tried to allocate the minimum number of sensors to a task to allow more sensors to be free for additional tasks and to reduce energy consumption by keeping redundant sensors asleep. In this section, the aim is at integrating both approaches and extending the problem formulation with additional restrictions that aim at reducing the number of sensors assigned to every task and to include the energy constraints of the sensors. To include additional restrictions and information in the resource allocation approach, these variables (available energy) need to be modelled in the system. One of the reasons for choosing semantic sensor network ontology (SSN) for modelling sensor nodes in the network is that it describes all various aspect of the operation, platform capabilities, as well as restrictions of the sensor nodes, which were not considered in other ontologies.

SSN does not include an energy management concept yet; however, the authors [138] propose a possible extension to include different energy resources that can be available in the ontology and how they can be added to the description of the sensor node. This possible extension is shown in Figure 5-5.
We consider two main features to formulate the problem of mission assignment as multi-objective optimisation:

- **Spatial Region**: The spatial region of a sensor node has already been considered in the previous chapter as an important indicator of the ability of the node to perform a specific task. The importance of having spatial region as a factor of the utility function is to remove the restriction, which results in hardcoding every sensor node to a specific area, which in turn results in limiting the use of the network to only those tasks that have been predefined.

- **Energy**: The remaining energy in every sensor node is another aspect that drastically affects the life span of the network. The network usually aims at balancing energy consumption amongst the nodes in the network to extend the life time and avoid having coverage holes in the environment. Node energy as a factor of utility is unique to every sensor in the network according to the usage history/pattern of the node.

Formulating the problem of mission assignment to consider both factors and to reduce the number of sensors assigned for every task are conflicting objectives. Thus, we formulate the problem as a bi-level mission assignment objective. At the task level, the aim is to reduce the number of sensors for every task, and at the network level the aim is to improve the gain from assigning sensors to tasks.
5.3.1 Bi-level Optimisation Problems

Bi-level optimisation problems are a special type of multi-objective optimisation problems where one or more objectives are embedded (nested) within the constraints of another optimisation problem. Bi-level optimization is characterised by the fact that the decision making at one level influences the decision at another level. This means that some attributes at one level are controlled by other units or optimisation objectives in another level. The outer optimisation problem is called the upper level problem while the inner problem is called the lower level optimisation problem. A valid solution for a bi-level optimisation problem requires that the result of the upper level problem be only feasible if it is an optimum for the lower level problem. The general formula of a bi-level optimisation problem can be found in the following equations, where $F$ represents the upper-level optimisation function, $f$ is the lower-level optimisation function, $G$ and $g$ are constraints functions, while $x$ represent the upper-level decision variables and $y$ is the lower-level decision variables.

$$\min_{x \in X} F(x, y)$$

Subject to:

$$G(x, y) \leq 0$$ (5.13)

$$\min_{y \in Y} f(x, y)$$ (5.14)

Subject to:

$$g(x, y) \leq 0$$ (5.15)

$$x, y \geq 0$$ (5.16)

Equation 5.12 is the upper level optimisation problem. Amongst its constraints, a lower level optimisation problem (5.14) can be found.

Bi-level optimisation have been studied widely in the current literature [166, 167]. They have been applied to various ranges of applications, like environmental policies [168], aircraft design [169], robotics [170] and others [171, 172]. It has also long been used for modelling allocation problems like allocation of water resources, housing allocation, etc., as well as other research areas [173]. However, it has not been used to formulate the problem of weapon target allocation or sensor mission assignment, as the main research into WTA has yet considered only a single optimisation problem.

5.3.1.1 Bi-level Sensor Task Assignment

As already discussed, we formulate the problem of sensor task assignment as a bi-level optimisation problem. We consider the leader to be the network of assets as a whole whose aim is to increase the
gain of task assignment, while the followers are the sets of assets that can perform each task. Their aim is to minimise the number of assets that can be used to perform the required task.

The main problem that the algorithm tries to solve is the one in Equation (5.17). The equation means that the aim is to maximise the benefit of the network as a whole by assigning different assets (sensors and actuators) to the different tasks in the network. The problem has different restrictions, (i) every asset can be assigned to a maximum of one task but it can be assigned to none (Equation 5.18), and (ii) minimise the number of assets that are assigned to every task (Equation 5.19). This second optimisation problem also has some constraints: (a) the sensing region of the sensor and the sensing region of the task must intersect (Equation 5.20-a), (b) the asset should be assigned to a task (Equation 5.20-b), and (c) the sensing region of the sensor should not be contained in the sensing region of another sensor (Equation 5.20-c). The rationale for the last equation is to not assign two assets that can cover the same task to one task but only use one of them. The formulation of the problem can be found in the following equations where: $V_j$ is the value of the task, $P_{ij}$ is the detection probability, $x_{ij}$ is the assignment of asset $i$ to task $j$, its value is equal to 1 if the asset is assigned to the task and 0 otherwise, $S$ is the set of assets assignment to tasks, $S_i(r)$ and $T_j(r)$ are the areas that represent the operational region of the asset (sensing region for a sensor) and the area occupied by the task respectively:

$$\text{Maximise } \sum_{j=1}^{m} V_j \left(1 - \prod_{i=1}^{n} (1 - P_{ij})\right)$$  \hspace{1cm} (5.17)

Subject to;

$$\sum_{i=1}^{n} x_{ij} \leq 1 \hspace{1cm} \text{for } j = 1, 2, ..., m$$  \hspace{1cm} (5.18)

$$\text{Minimise } \sum_{i=1}^{n} (S_i * x_{ij}) \hspace{1cm} \text{for } j = 1, 2, ... m$$  \hspace{1cm} (5.19)

Subject to;

$$S_i(r) \cap T_j(r)$$  \hspace{1cm} (5.20-a)

$$x_{ij} = 1$$  \hspace{1cm} (5.20-b)

$$S_i(r) \not\subset S_k(r) \hspace{1cm} \text{for all } i, k = 1, 2, ..., n$$  \hspace{1cm} (5.20-c)

Many approaches have been introduced for solving bi-level optimisation problems, many of those are based on genetic algorithms [174, 175]. Since we have used genetic algorithms for weapon
target assignment, we extend our approach and propose a new bi-level evolutionary algorithm based on the selfish gene algorithm to solve the problem of bi-level sensor task allocation.

One challenge of a bi-level mission assignment problem is that some tasks might be competing for the same resources; with more than one resource might be needed for every lower level optimisation. Further with the assumption that every sensor can be assigned to only one task, the allocation is more challenging. To solve this problem, we propose to use one form of biological altruistic behaviour called *kin selection*, which performs a local search to choose the better genes to pass through generations, even at the cost of an individual’s survival.

### 5.3.1.2 Genetic Algorithms with Local Search

Local search is the process of searching neighbouring solutions to a current solution for a better one. In genetic algorithms for example, local search would exchange the locations of different genes within the current chromosomes in the search for a better solutions (this is called neighbour solution). The chromosome is compared to its neighbours and the fittest chromosomes are the ones that will continue to evolve through the genetic process.

Local search integrated with genetic algorithms has been considered solving optimisation problems, and different integrations of genetic algorithms with local search approaches have already been implemented in the literature [176, 177]. The advantages of local search is to explore neighbouring solutions to a current solution to provide a variety of the solution in a local manner before moving to the next generation. This can help converge to an optimum solution much faster.

As local search explores neighbouring solutions (in the case of genetic algorithm, neighbouring chromosomes) to the current solution. Kin selection (explained in the following section) can be considered as one type of local search;

#### 5.3.1.2.1 Kin Selection

Altruism is a type of social behaviour where individuals would perform an altruistic action at a cost of their survival but beneficial to other relatives. Kin selection is one type of altruism and it is the evolutionary strategy that favours one’s siblings for reproduction rather than its own for the benefit of the whole group. One remark made by John B.S. Haldane to explain kin selection behaviour was –as quoted in [178], page 82:

*“Would I lay down my life to save my brother? No, but I would to save two brothers or eight cousins”*

The difference between kin selection and other local search approaches is that: even though the individual looks for a better chromosome amongst its siblings, it considers the benefits that can be
brought for the group as a whole, making it a local search which considers the fitness at the level of the chromosome rather than the genes.

5.3.1.2.2 Hamilton Rule of Altruism

In 1964, W.D. Hamilton proposed a mathematical treatment of the kin selection theory which is known as the “Hamilton rule” [179, 180]. It specifies when a chromosome gives up on reproducing for helping its sibling with reproduction. It defines that this happens when such an altruistic behaviour results in a more likelihood for a specific gene to pass through generations and survive the evolution. Hamilton rule is:

\[ rB > C \]  

(5.21)

Where:

- \( r \) is the level of relation between the chromosome and its sibling, this is measured by the proportion of shared genes between the two chromosomes;
- \( B \) is the benefit and it refers to the additional offspring of the recipient as a result of the altruistic behaviour from the sibling chromosome;
- \( C \) is the cost to the altruist, and it refers to the number of offspring that cannot be produced due to the altruistic behaviour compared to the number of offspring that could have been produced if altruistic was not performed.

We propose introducing kin-selection for the upper level optimisation problem when tasks will be competing for resources to make sure that sensors are assigned to tasks that brings most benefit to the network as a whole.

Before introducing the operation of the algorithm, we need to define the criteria for the detection probability as we consider spatial based mission assignment, and thus spatial based detection probability.

5.3.1.3 Sensor Detection Probability

Unlike the genetic algorithms that have been proposed for solving the WTA problem in literature, and the selfish gene algorithm that we propose using in this chapter, the considered problem has additional constraints; that is the energy constraints of the nodes. Additional constraints on the problem formulation are introduced to target the problem.

The probability of a sensor to detect an event \( p_{ij} \) –or the utility that a node can offer a mission– has so far been mainly verified as a binary function based on the distance between the sensor and the task and the sensing range of the sensor [181] using the following Equation:
Chapter 5. Spatial WTA for Mission Assignment in WS&ANs

\[
P_{ij} = \begin{cases} 
1 & \text{if } d(s_i, t_j) < S_r \\
0 & \text{Otherwise}
\end{cases}
\]  
\hspace{1cm} (5.22)

where \( d(S_i, T_j) \) is the distance between the location of the sensor and the location of the task, and \( S_r \) is the sensing range of the sensor.

Another approach into finding the detection probability of tasks was introduced in [182]. The authors divide a circle of radius equal to the sensing range and centred at the location of the task into rings, all sensors whose locations fall in the same ring as the task have the same detection probability—in this case 90%. Sensors in the second range have a probability of 80% and so on. Similar to the binary function, it assumes that the task is a 2D point in space and considers the distance from the sensor to the task as a base for calculating the detection probability.

Both approaches assume that each task occupies one point in space, which is not the assumption we make about tasks in the network. In reality, just as sensors and actuators have functional space where they can operate, each task occupies a space which is more than one point. This is certainly the case when reasoning about properties of objects because objects do occupy space, like a room, a building, etc.

None of the algorithms which looked into the coverage of a sensor/actuator of an object considered the object or task to span over space, neither did any approach in crowdsourcing. Thus, assuming that the shape of the sensing field is regular, we make the following assumption regarding the probability or coverage of a sensor to a task;

\[
P_{ij} = \begin{cases} 
1 & \text{if } \text{Contains} \left( S_i(r), T_j(r) \right) \\
0.5 & \text{if } \left( \text{Within} \left( T_j(r), S_i(r) \right) \lor \text{Overlaps} \left( T_j(r), S_i(r) \right) \right) \land \text{Distance} \left( S_i, T_j \right) \leq S_r \\
0.25 & \text{if } \left( \text{Within} \left( T_j(r), S_i(r) \right) \lor \text{Overlaps} \left( T_j(r), S_i(r) \right) \right) \land \text{Distance} \left( S_i, T_j \right) \geq S_r \\
0 & \text{Otherwise}
\end{cases}
\]  
\hspace{1cm} (5.23)

where \( S_i, T_j \) refer to the centre location of the sensor and the task respectively, and \( T_j(r), S_i(r) \) are the space occupied by the task and the functional space of the sensor/actuator respectively.

We have already extended the SSN ontology with spatial information in a previous chapter. Having introduced the detection probability of the sensors, in the next section, our proposed bi-level genetic algorithm is proposed to solve the problem of multi mission assignment in WSNs.
5.3.2 Geospatial Bi-level Selfish Algorithm for Sensor Mission Assignment Algorithm (BiSMA)

Bi-level Selfish Mission Assignment (BiSMA) algorithm starts by trying to find the set of all sensors in the network which can contribute to either of the tasks. This is done by querying the ontology for assets whose sensing or functional range intersects with the region of the task. The resulting set of assets is the solution space of the algorithm. It then tries to optimise the solution by choosing the minimum set of sensors for every task which can better utilize the performance of the network.

When querying the ontology for the sensors that can contribute to every task, the algorithm also finds the detection probabilities for every sensor to every task based on the formula (5.24). This is important as the detection probabilities differ from task to task.

After finding the solution space for every task in the network, the algorithm tries to optimise the solution. The algorithm tries to solve the two level optimisation problem using nested selfish gene algorithms on the upper and lower level. For every task in the lower level, BiSMA runs the selfish gene algorithm, results from every run are used by the upper level algorithm to optimise the solution also based on the selfish gene algorithm.

Kin-selection is used by the upper level optimisation problem when there are competing tasks. The algorithm searches all possible neighbouring solutions by alternating the shared resources between the tasks and calculating the overall benefit (the upper level objective). The solution which produces the best results is the one passed back to the lower level selfish gene algorithms at every task level to penalize/reward the solution and search for a new one. The operation of BiSMA algorithm can be seen in Figure 5-6 where SP is the selection probability (following the selfish gene algorithm), u indicates the upper level optimisation problem, and l represents the lower level.
Figure 5-6 BiSMA Flowchart
The implementation of BiSMA is similar to that of the selfish gene for the follower’s problem in terms of its operations like reproduction and selection. However at the high level, the algorithm does not operate as an evolutionary algorithm, but it is used as a local search that tries to optimize the allocation of assets to tasks to increase the gain from performing these tasks.

At the leader level we formulate the tasks as an array for tasks, the solution for each one of them – based on the solution space calculated at the follower problem – is another list of all possible sensors or assets. An explanation of the algorithm procedure in steps can be seen in Figure 5-7.

Figure 5-7 BiSMA procedure

Note that the algorithm does not use Hamilton’s rule exactly as it is, it does not use the level of relatedness between two sets of solutions, but only compares the benefit and cost of each altruistic genome. The reason for this selection is that when two tasks share more than one sensor, it might be more beneficial for the altruistic task to give up one asset to the recipient task rather than give...
up both of them. Thus, level of relatedness is not used and the altruistic task might not give away all its shared genes.

5.3.2.1 Validation

Only one approach in the literature has considered spatial mission assignment with competing tasks [182]. The approach proposed by the author is a greedy approach which aims at assigning assets to the task which has the highest profit value first (defined in section 5.2) and then move to the next one. We implement the greedy approach to compare with BiSMA. In the greedy approach, selfish gene algorithm is used to find the solution for the tasks starting with the ones of a highest profit value. However, assets that are assigned to a specific task are not allowed to be reassigned to another—as can be seen in the restrictions of the problem formulation.

The simulation environment is similar to that used in the previous section when running the multi-star selfish gene algorithm. Sensors are generated with random radius in the range [0-100] and the probability of sensors to detect the task is calculated based on their spatial relation following the Equation 5-23. Tasks, however, are randomly generated, but in the range [40-60] with random radius between [1-6] and with values in the range [1-100]. The reason for limiting the generated tasks to the small region is to force intersection between them, and as a result, some computation between the tasks for resources.

Figure 5-8 shows one run example of the algorithm, with three tasks competing for resource in the network. The figure shows that BiSMA can accomplish a higher fitness than that reached by the
greedy algorithm because it is able to perform more tasks than the greedy algorithm. The reason for this allocation is the altruistic behaviour that some tasks perform (in this case task 1).

Figure 5-9 also shows that BiSMA shows the performance of the BiSMA algorithm in terms of resource allocation. It can be seen that, the first task is assigned 6 sensors which made it not possible for the greedy algorithm to assign any more sensors to other tasks. In contrast, BiSMA assigned only 4 sensors for the first task, leaving 2 for the third task, resulting in a gain in the overall fitness of the network. However, T2 was not assigned any asset in the network as its profit value is low and so assigning sensors to it did not benefit the network as well as assigning the assets to other task.

![Figure 5-9 Number of Assigned sensors](image)

We also run the algorithms for 10 times for different scenarios –with different numbers of task– and we record the average fitness achieved by each of the algorithms. Table 5-3 displays these results. They show that BiSMA outperforms the greedy algorithm and can reach a higher fitness level. The reason for that is that BiSMA can better allocate and balance the allocation of assets to tasks in the network to increase the network fitness by introducing the concept of altruism.
### 5.4 Conclusion

In this chapter, the aim is to extend the spatial mission assignment problem in WSNs to include additional restrictions like the energy of the nodes. Another aim is to consider multiple spatial mission assignment. To accomplish that the problem was formulated based on the weapon target assignment problem by considering sensors as weapons and tasks as the targets, while the detection probability is analogy to the “kill probability” in WTA. Unlike regular WTAs, not all sensors can perform all tasks in the network because of their spatial restrictions. Another assumption made is that not all sensors have to be assigned to tasks.

First the use of multi-stars selfish gene algorithm was introduced to solve the general problem of weapon target assignment. Simulation results show that the multi-star selfish gene algorithm outperforms the two genetic algorithms introduced in literature. The reason why the selfish gene algorithm outperforms current state of the art is the fact that it does not count on an initial population to evolve, and as a result the solution space which affects the evolution of the algorithm is large and allows all various potential options.

After proving that selfish gene is the most suitable algorithm for answering the mission assignment problem when considering limitations of the assets, the problem was extended to multi-task mission assignment.
assignment with spatial and energy constraints. Since the considered constraints are contradicting, the problem is formulated as a bi-level optimisation problem.

Since the main aim criteria of the problem considered is the spatial properties of both assets and tasks, and because functional space of assets as well as space of objects in a smart environment will intersect, the tasks are assumed to compete for the resources in the networks. Thus their spatial regions would intersect. BiSMA, as a bi-level selfish gene algorithm with altruistic behaviour was introduced to solve the bi-level competing task allocation. The algorithm was compared with a greedy algorithm—the only approach for spatial competing task in literature—that tries to satisfy the task with the highest value first. Simulation results shows that BiSMA outperforms the greedy algorithm, it can reach a better fitness for the operation of the network as a whole and it can satisfy more tasks than a greedy algorithm does.

Existing approaches introduced into allocating tasks to sensors consider user’s knowledge about the tasks and the sensors around, they still use human interaction to identify the assets needed to perform a specific task, thus the importance of the allocation algorithms that were introduced in this chapter. While many approaches for resource allocation have considered multiple tasks, none of them have looked into the spatial properties of tasks and sensors. In a future environment where smart devices will be available everywhere, it is very important to consider their spatial properties and relationship as it lays in the heart of controlling objects. In the next chapter, we introduce a language that makes use of the allocation algorithms introduced in this chapter to enable users to control their environment, and reason about its properties and objects.
Chapter 6

6 High Level Programming Language for Wireless Sensor/Actuator Networks

Since their invention, computing devices have moved from being tools for scientist and researchers to being integrated in everyday life. While they can mostly be noticed in work spaces and schools in the forms of computers with keyboard, screen, mouse, etc. computing devices can be seen everywhere around us, in fridges, cd players, cars, mobile phones, etc. Still, most of these devices are separate from each other, without any interaction with other devices or the environment.

UbiComp introduces the idea of allowing computing devices in any environment to communicate and interact with each other and with users around. Most work in this area have focused mainly on making these devices self-configured using machine learning approaches so they do not require the users to pay any attention to the system. An important aspect of enabling such an environment that has not received the same attention in research is how a user can utilize their systems without any need for specialists or programmers. With many heterogeneous computing devices and sensors that enable users to control their smart environments, the challenge remains to make it easier for end users to specify the behaviour they require from their environment. This has caused limited usability of UbiComp technology.

One of the basic challenges to a wider deployment of UbiComp systems has been the current approach taken to build these systems. First of all, current UbiComp systems tend to be either too simple which limits their scalability and functionality, or too complex for end users to develop, and it is the developer who takes control on designing the system and making decision for the end users. Second, most of the software tools are domain specific; they are hard coded for a specific application which limits their reusability in other domains.

Trying to make programming and controlling the environment easier for end users has been a subject of research recently. Many high-level, simple programming languages and tools have been introduced to enable domain experts program wireless sensor networks and UbiComp, as reviewed in Chapter 2. While these languages could abstract away from the complexity of programming
UbiComp devices like sensors and actuators for example, their limitation is that they still consider the user of the system as a developer but on a different abstraction layer. It means that the users still need to know about all the technology of the system – all the sensors, actuators, etc. – and when they develop their applications they will be reasoning about and programming the behaviour of such technologies. This approach works for domain experts who know what information they want from their environment and how the system should work to obtain the knowledge they require. While this approach has been dominant in the literature, it is far from being able to make end-users of the system programmers for its behaviour. In a future UbiComp where prospective users include everyone, even those without domain knowledge or programming skills, enabling people to develop their own applications and environment behaviour becomes even more challenging (C1, Chapter 1, Section 1.5, (p.6)).

In a future where science and technology need to be hidden in the environment but also openly available, a change is needed. New development tools and programming languages should describe domain knowledge and enable end-users to build their own applications around their environment according to their needs (O1, Chapter 1, Section 1.6, (p.7)). Domain specific programming languages can be helpful to establish such a shift in computing, where a language can be truly about real world objects when programming real world objects. One recent approach into solving the problem was proposed in [183, 184] where a graphical user interface was developed on top of semantic service mash-up for a simple smart environment configuration.

In Chapter 3 we have discussed the design goals of such a system, these include: the need for a simple, task centric, flexible language and efficient task allocation algorithm. Separating the user’s application from the asset code is essential to provide the required level of invisibility and hide complexity. This is due to the fact that programming the sensors & actuators in the network will remain a complicated process with the various internal mechanisms that needs to be specified for the network to operate, while users’ code needs only to describe the required behaviour from the environment. To enable separating users’ code from the system code, it is essential to be able to find the appropriate sets of sensors & actuators which need to be programmed to achieve the user’s goal each time, thus algorithms for spatial mission assignment in WS&AN are introduced in Chapters 4 and 5 (TC1, TC2, TC3, Chapter 1, Section 1.8, (p.8-9)).

In this chapter, we present the Environment Programming Language (EPL), a novel high level programming language to facilitate reasoning about properties of real world objects and programming their behaviour by novice programmers. EPL manages to hide away technology from end users and allow them to control their environment without the need for knowledge about the underlying technologies which makes such manipulation possible (TC4, Chapter 1, Section 1.8, (p.9)).
Chapter 6. High Level Programming Language for WS&ANs

6.1 Environment Programming Language

6.1.1 Overview

EPL language features a declarative syntax based on real world objects –domain dependant– and their properties such as a room, a person, temperature, presence, etc., to allow describing real world applications with low complexity. This level of abstraction was proposed because previous research found that users are more interested in objects of real world when they aim at designing and controlling their environment than in the technology used to control them –the motivation and previous research are discussed in Chapters 1&2. Semantic web provides a unified powerful framework for representing, capturing and reasoning about properties of ubiquitous computing [185]. EPL uses ontologies of real world objects and technologies to allow end-users to focus only on the logic of a program and its required behaviour from the environment.

The use case that is used to illustrate the system is that of a smart building. Using a text editor, a user should be able to define the behaviour of his/her environment without the need to deal with the technology in the environment e.g. check and adjust the temperature in a room, check if someone is present in another room, etc. The system is domain dependant, which means that the user should be able to use a text editor to control any smart environment that he/she wishes, as long as they have the right ontology to describe the environment. Examples include using the system in a smart office, shopping centre, airport, etc. While these environments have different objects and different properties that the user might want to manipulate, the system design should be able to satisfy user’s requirements in different environments. A user should also be able to define a more complex behaviour in the system by being able to introduce new properties of real world objects based on existing ones.

Figure 6-1 describe the architecture of our system and how the domain programming language (DSL) integrate with the ontologies and the reasoning algorithms introduced in the previous chapter.

![Figure 6-1 System Architecture](image-url)
Because our language aims at simplifying environment control for end-users by hiding information about the underlying technology, the system is separated into multiple layers. This enables defining necessary functionality of the system while hiding it from the user for easier manipulation. The user interface allows the user to specify the required behaviour without the need to specify the assets that will perform it. The allocation layer uses internal resources stored in the ontology and predefined functions and relations to perform resource allocation required for mapping the user’s required behaviour to available assets.

We start by describing the domain problem and giving formal specification for the various parts of the system and DSL.

6.1.2 Problem Domain

The first step to develop our language is to define the problem domain and the main notions in terms of behaviour. The system architecture in Figure 6-1 shows a distinction between three components of the language: a user interface where the user can specify the required behaviour, an allocation layer where the user request would be processed along with values provided by the third component—the ontology models—to find the task allocation. We describe the components of the system in the following:

6.1.2.1 User Interface

As the aim of the language is to be easy for non-developers, yet be applicable for a wide range of domains by integrating real world objects into the language, at the core of the language we find objects of the real world and their properties.

Objects. An object is the basic entity that an end-user would reason about and manipulate its properties. The following concepts of objects of real world are considered:

- \( \text{Obj} \) : a finite set of objects available in the domain of interest
- \( \text{Prop} \) : a finite set of all properties available in the environment.
- \( \text{ObjProp} : \text{Obj} \rightarrow \mathcal{P}(\text{Prop}) \), the set of properties of an object

Properties of objects can have different data types that end-users can enquire about, such as a number for representing about the temperature of a room and a Boolean value to represent the presence of someone in the building, etc. When users manipulate some objects of their environment they change their status. We consider two status values for objects:

- \( \text{Status} = \{\text{On, Off}\} \)
- \( \text{CurrentObjStatus} : \text{Obj} \rightarrow \text{Status} \)
Chapter 6. High Level Programming Language for WS&ANs

ObjPropValue = \{\text{Number, Boolean}\}

**Actions.** The status of the objects and their properties as well as their values (e.g. temperature value) can be manipulated with the reasoning, activation and deactivation actions which we refer to as ActionType

ActionType = \{\text{Read, Activate, Deactivate}\}

**Restrictions.** Controlling real world objects and their properties can be specified under user defined restrictions (e.g. activate heater of office where temperature > 20)

A restriction is a tuple (con, action)

\(\text{con} \in \text{Condition}\)

\(\text{action} \in \text{ActionType}\)

We consider two types of conditions: Spatial restriction (like room above stairs) and expressions (temperature > 30)

\(\text{SpatialRest} \colon \text{SpatialRelation} \rightarrow \text{Boolean}\)

\(\text{Expressions} \colon \text{ObjPropValue} \rightarrow \text{Boolean}\)

### 6.1.2.2 Ontology Models

**Asset Model.** To manipulate real world objects and reason about their properties, a collection of assets or resources –sensors/actuators– with various capabilities are implemented within the system.

\(\text{Res} \) the set of resources available in the system

\(\text{ResStatus} \colon \text{Res} \rightarrow \text{Status}\)

\(\text{ResCap} \colon \text{Res} \rightarrow \wp(\text{Prop}) \) the set of properties a resource can manipulate

**Spatial Model.** All objects in the system as well as all the resources have spatial properties. We formulate their spatial location, area and operational region using GeoSPARQL as their geometric model. The data type used for representing spatial entities in GeoSPARQL is well-known text (WKT) Literal

\(\text{ObjectArea} \colon \text{Object} \rightarrow \text{WKT}\)

\(\text{ResOpRange} \colon \text{Res} \rightarrow \text{WKT}\)

GeoSPARQL –introduced in Chapter 2 and 4– does not only introduce a representation to the spatial data but also include functions and relations to reason about the interconnections between the objects with spatial data representations. We consider qualitative relations between simple features.
similar to what was considered in previous chapters—spatial calculus to represent the spatial relations and functions. Two types of spatial relation are considered based on GeoSPARQL: (i) relations between two real world objects and (ii) relations between an object and an asset;

\[
\text{GeoSPARQLSpatial} = \{\text{ObjSpatial, ObjResSpatial}\}
\]

\[
\text{ObjSpatial} : \text{ObjArea} \times \text{ObjArea} \rightarrow \text{SF}
\]

\[
\text{ObjResSpatial} : \text{ObjArea} \times \text{ResOpRange} \rightarrow \text{SF}
\]

such that, for all \(o_1, o_2 \in \text{Object}\)

\[
\text{SF} = \{\text{Equals}(o_1, o_2), \text{Disjoint}(o_1, o_2), \text{Overlaps}(o_1, o_2), \text{Touches}(o_1, o_2), \text{Contains}(o_1, o_2), \text{Within}(o_1, o_2), \text{Intersects}(o_1, o_2)\}
\]

**Decision.** The spatial calculus (data representation, relations and functions) that were introduced in GeoSPARQL consider only a two dimensional space. Extending that to 3D would require introducing a new data type and new calculus for spatial reasoning which has not been yet introduced for semantic reasoning. To enable extending the spatial relation to include two additional relations (above, below) we extend the semantics representation of objects and resources with a new property—namely elevation—and project the objects on a 2D space to reason about their spatial relations.

\[
\text{3Drelations} = \{\text{above, below}\}
\]

\[
\text{SpatialRelation} = \{\text{GeoSPARQLSpatial, 3Drelation}\}
\]

The semantics of 3D relations extends that of GeoSPARQL to include additional condition corresponding to the elevation of the object, they are included in the allocation layer because they are not a part of GeoSPARQL.

### 6.1.2.3 Allocation Layer

The allocation layer uses the user request as well as the ontology model values to map the user’s request to a set of assets that can perform the required task. However, the mapping request function (MapReq) is not modelled in the DSL because the main goal of the language is to abstract away from technology:

\[
\text{MapReq} : \text{ObjArea} \times \text{ObjProp} \times \text{ResCap} \times \text{ResOpRange} \rightarrow \wp(\text{Res})
\]

While MapReq finds the set of assets that can perform a task as a base solution, the aim is to optimise the solution to include the minimum set of resources, thus we introduce allocation request function (AlReq). The semantics of the MapReq function is specified as follows:
Chapter 6. High Level Programming Language for WS&ANs

[MapReq] (oa, op, rc, rr)

\[ t_{\text{MapReq}} = \{ r \mid \forall r \in \mathcal{P} \land (\exists rc \in \mathcal{P} \land rc = op) \land \text{Intersects}(oa, rr) = T \} \]

Where \( r \) is a resource, \( \mathcal{P} \) is a set of resources, \( rc \) is a resource capability (a property it can manipulate) and \( op \) is the object property, while \( oa \), and \( rr \) are the object area and resource range respectively and they deal with the spatial properties.

The allocation functions tries to find the minimum set of assets that can satisfy the user request if it exists or it will use the full set of assets that can contribute to the task:

\[
\begin{align*}
[\text{AlReq}](oa, op, rc, rr) & = \\
& \begin{cases} 
\text{min}[\text{MapReq}](oa, op, rc, rr) & \text{if } (\exists r \in \mathcal{P})\text{Contains}(oa, \bigcup_i rr_i) \\
[\text{MapReq}](oa, op, rc, rr) & \text{if } \neg(\exists r \in \mathcal{P})\text{Contains}(oa, \bigcup_i rr_i)
\end{cases}
\end{align*}
\]

Restrictions. The GeoSPARQL spatial restrictions are evaluated using GeoSPARQL ontology model as it specifies the rules for reasoning about the spatial relations. As for the additional 3D relations, they are projected on a 2D surface and reasoned about using GeoSPARQL and additional restrictions.

\[
3\text{Drelation} : \text{Obj} \times \text{Obj} \to \text{Boolean}
\]

\[
3\text{Drelation}(o1, o2) = \begin{cases} 
\text{above} & \text{if } (\text{Ele}(o2) > \text{Ele}(o1) \land \text{Intersects} (o1, o2) = T) \\
\text{below} & \text{if } (\text{Ele}(o2) < \text{Ele}(o1) \land \text{Intersects} (o1, o2) = T)
\end{cases}
\]

Where \( \text{Ele}(o) \) is the elevation of the object \( o \) available in the ontology.

6.1.2.4 Formal Abstract Syntax

We start by describing the elementary notions in the system in terms of behaviour to obtain the abstract syntax of the language.

Task. The most elementary notion in the system with observable behaviour is a task. Examples of tasks are: read temperature of office1, activate doorlock of office, etc.

Definition 1: Since every task is associated with objects and properties, we consider them as the context of the task and we refer to them in the description of the task, thus a task is: \( t_{p,o} \).

Definition 2: To be able to reason about real world objects, the system makes use of sets of assets available in the environment like sensors and actuators. Every elementary task needs to use a set of assets to be performed. When these assets are in use by one task they are not available to other tasks in the system. However, when the task is completed they are released back to the system. Thus we distinguish between the beginning and end of the task. Let \( T^b \) be the set of all starting tasks, \( t^{b}_{p,o} \) refer to the beginning of task \( t \), \( T^e \) the set of all ending tasks and \( t^{e}_{p,o} \) the end of task \( t \).
Definition 3: In some situations we might not want to observe the behaviour of the system like the case when a condition is not satisfied, in this case we use the process term $\tau$, where $\tau \notin T^b \cup T^e$. We consider the set of all tasks in the system as $T = T^b \cup T^e \cup \tau$.

Process relations. There are two types of relations between processes in the network. It is possible for two processes to perform concurrently whenever it is possible to allocate assets to both processes to optimise the performance of the network. However, some processes can only happen after proceeding process has finished, as they might depend on the results obtained from the proceeding processes.

For each type of relation we introduce a specific operator. Let $p, q$ be two processes,

Definition 4: A sequential execution of the processes is expressed as:

$$p \cdot q$$

Definition 5: Concurrent processes are expressed as:

$$p \parallel q$$

Aggregation. Since the proposed language consider manipulating properties of real world objects and considers the spatial properties of these objects, and because the aggregation of some objects can result in other objects, the aggregation of different processes is considered as another behaviour of the system. The aggregation of processes would fork the concurrent behaviour of a task, and applies them to different objects which makes up the aggregated objects. We take as an example the description of a building in the IFC (industry foundation classes) model. The model considers the building to be an aggregation of floors, and a floor is an aggregation of spaces where these spaces can be rooms, kitchen, office, etc. Categorizing space following this model should be a part of the domain ontology as is the case for the IFC model which defines the aggregation of space as a part of the description of a building.

Definition 6: We refer to the aggregation of processes as:

$$i^b_{\sigma(obj), p}$$

To be able to perform an aggregation of tasks, it is important that the object of the process is an aggregation of different objects; we refer to the condition for forking aggregated behaviour as:

$$\sigma(obj) = \bigcup_{i} obj_i \quad i \in \mathbb{N} \ (the \ number \ of \ aggregated \ objects)$$

Choice. Executing some actions can be conditional to some criteria, in which case the system will test the condition and execute different tasks accordingly. An example can be “activate the heater of office if temperature < 18”.

116
Definition 7: Assuming a condition function c a choice operator will assign one process to every outcome of the decision function.

\[
\bigvee_{c_n}(p_1, p_2, ..., p_n) = \begin{cases} 
    p_1 & \text{if } c_1 = \text{true} \\
    p_2 & \text{if } c_2 = \text{true} \\
    \vdots \\
    p_n & \text{if } c_n = \text{true}
\end{cases}
\]

Definition 8: The decision on which process to perform is based on the state of the system, thus there is a need to store the state of the system at all times like the values of some properties or the status of the objects. Assume \( S \) to be the set of states or values, and let \( V \) be the set of variables at a specific situation (like the state of an asset or the property of an object ... etc.). We define \( V \rightarrow \) to be the set of variables evaluations. Consider \( \delta \in \xi \) to be a list with all the evaluations of the current state of the system.

Resources. It has already been mentioned when introducing tasks that every task in this system model requires a set of resources (assets) to be performed, these resources will be allocated to the corresponding task when the task starts and they are released when the task finishes. Let \( R \) be the set of all available resources in the network. Let \( R_a(t) : T \rightarrow P(R) \) be the set of resources allocated to task \( t \), and let \( R_r(t) : T \rightarrow P(R) \) be the set of resource released when the task \( t \) is executed. Assuming that \( R_A \) is the set of all available resources, these resources needs to be modelled in the evaluation list of the system state (\( \delta \)). We define the following premise for allocating resources for a task \( t \); \( R_A \geq R_d(t) \)

New Task. Some properties of the real world might not be integrated in the ontology of the domain, and they might require different types of resources to evaluate. Thus, we propose task composition to allow defining new properties and their associated tasks and properties composition. Let’s say for example that a suspicious behaviour on the road is the detection of a vehicle with speed faster than a certain threshold. The property in this case is restricted to specific speed limits. To detect a suspicious vehicle the system needs to find all moving cars and compare their speed to the specified threshold. Introducing task composition is important as it allows the reuse of these tasks in the system.

Definition 9: To enable task composition we introduce process equation. However, the name of the new task should not be one of the names of the start and finish tasks of the system. Let \( NT \) be the set of names for new tasks \( NT \cap T = \emptyset \), and let \( nt \in NT \). We define task composition with the equation:

\[
nt \equiv p
\]

The formal abstract syntax of the language can be seen in Table 6-1.
Chapter 6. High Level Programming Language for WS&ANs

Table 6-1 Formal Abstract Syntax

<table>
<thead>
<tr>
<th>Process term</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p ::= \tau )</td>
<td>Skip</td>
</tr>
<tr>
<td>( t_{o,p}^b )</td>
<td>beginning of task ( t ) to reason about property ( p ) of object ( o )</td>
</tr>
<tr>
<td>( t_{o,p}^e )</td>
<td>end of task ( t ) to reason about property ( p ) of object ( o )</td>
</tr>
<tr>
<td>( t_{a(o),p}^b )</td>
<td>start of task ( t ) to reason about property ( p ) of aggregation of objects ( o )</td>
</tr>
<tr>
<td>( p \cdot p )</td>
<td>Sequential tasks</td>
</tr>
<tr>
<td>( p \mid p )</td>
<td>Concurrent tasks</td>
</tr>
<tr>
<td>( V_{c_n}(p_1 \ldots p_n) )</td>
<td>Choice</td>
</tr>
<tr>
<td>( np \equiv p )</td>
<td>New task</td>
</tr>
</tbody>
</table>

6.1.2.5 EPL Dynamic Semantics

We consider a process in EPL to be the tuple \( \langle p, \delta \rangle \), where \( p \) represents a process, and \( \delta \) is the variable evaluation. We consider \( \delta \) to contain all information about the current state of the system including the list of available resources for tasks and the evaluation of some conditions.

The dynamic semantics of the EPL language can be found in table 6-2. We describe the operations briefly in what follows:

**Skip**: When projecting the abstract syntax of the language, the skip task (\( \tau \)) is a non-observable task with no behaviour. The task does not use any resources in the network and does not release any.

**Begin task**: The start of the task is the action that starts the execution of the primitive task \( t_{o,p}^b \). To perform the task, the set of resources \( R_o(t) \) must be available, and the evaluation list is updated upon the use of the required resources.

**End task**: At the end of the execution of a task \( t_{o,p}^e \), the system will release the resources used for the task and update the evaluation function. The task does not have any condition to be executed.
### Chapter 6. High Level Programming Language for WS&ANs

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(skip)</td>
<td>$R_A(t) \geq R_a(t)$</td>
</tr>
<tr>
<td>(begin task)</td>
<td>$\langle t_{a,p}, \delta \rangle \xrightarrow{t_{a,p} \cdot R_a(t)} \langle t_{o,p}, \delta \rangle$</td>
</tr>
<tr>
<td>(finish task)</td>
<td>$\langle t_{o,p}, \delta \rangle \xrightarrow{t_{o,p} \cdot R_f(t)} \langle \checkmark, \delta \rangle$</td>
</tr>
<tr>
<td>(task aggregation)</td>
<td>$\langle t_{\sigma(o), p}, \delta \rangle \xrightarrow{\beta, \alpha} \langle t_{o_1,p} \cdot t_{o_2,p} \cdots \cdot t_{o_n,p}, \delta \rangle$</td>
</tr>
<tr>
<td>(Sequential process 1)</td>
<td>$\langle p, \delta \rangle \xrightarrow{\beta, \alpha} \langle p', \delta' \rangle$</td>
</tr>
<tr>
<td>(Sequential process 2)</td>
<td>$\langle p, \delta \rangle \xrightarrow{\beta, \alpha} \langle \checkmark, \delta' \rangle$</td>
</tr>
<tr>
<td>(choice 1)</td>
<td>$\langle p_c, \delta \rangle \xrightarrow{\beta, \alpha} \langle p', \delta' \rangle$</td>
</tr>
<tr>
<td>(choice 2)</td>
<td>$\langle p_c, \delta \rangle \xrightarrow{\beta, \alpha} \langle \checkmark, \delta' \rangle$</td>
</tr>
<tr>
<td>(concurrent process 1)</td>
<td>$\langle p, \delta \rangle \xrightarrow{\beta, \alpha} \langle p', \delta' \rangle$</td>
</tr>
<tr>
<td>(concurrent process 2)</td>
<td>$\langle p, \delta \rangle \xrightarrow{\beta, \alpha} \langle \checkmark, \delta' \rangle$</td>
</tr>
<tr>
<td>(concurrent process 3)</td>
<td>$\langle q, \delta \rangle \xrightarrow{\beta, \alpha} \langle q', \delta' \rangle$</td>
</tr>
<tr>
<td>(concurrent process 4)</td>
<td>$\langle q, \delta \rangle \xrightarrow{\beta, \alpha} \langle \checkmark, \delta' \rangle$</td>
</tr>
</tbody>
</table>

$\sigma(o) = \bigcup_{i=1}^{n} o_i$
Chapter 6. High Level Programming Language for WS&ANs

Table 6-2 Dynamic Semantics

| Task aggregation. | The rule for task aggregation can be seen in Table 6-2. It shows that if the resources required to execute a task are available and the object of the task is an aggregation of more than one object, the execution of the task is the concurrent execution of the same task on each of the aggregated objects. |
| Sequential Process. | The sequence of process p.q behaves as q if the process p has terminated successfully after performing the action β and using the set of resources α (rule sequential process 2). If the process p when executing the actions β and α becomes p’ then the sequence of process becomes p’.q. |
| Choice. | The choice selects executing one process amongst the available choices based on the evaluation of the decision function. Performing the chosen process would either terminate successfully or result in a new process p’. |
| Concurrent Process. | The rules concurrent process, defines that either of the processes p and q will be executed to terminate or proceed to p’,q’. |
| New task. | When defining a new task, we generate a new identifier and substitute it with the name of the process and perform the action on the task. |

6.2 EPL Concrete Syntax

The abstract syntax and semantics that are introduced in the previous section describe the basic behaviour of the system. The concrete syntax which end-users would use to manipulate real world objects is the following:

```plaintext
grammar EPL;
application
  :   APP IDENT
  | programBlock
END IDENT
;
programBlock
```
Chapter 6. High Level Programming Language for WS&ANs

:   (definitionStatment | commandStatment)*
;

definitionStatment
:   constant
|   variable
;

constant
:   THRESHOLD IDENT '=' value
;

variable
:   NEW IDENT
;

commandStatment
:   componentUse
|   assignmentStatment
|   ifStatement
|   whileStatement
;

assignmentStatment
:   IDENT ':=' expression
;

ifStatement
:   IF expression THEN commandStatment+
    (ELSEIF expression THEN commandStatment+)*
    (ELSE commandStatment+)?
    ENDIF
;

whileStatement
:   WHILE expression
    commandStatment*
    ENDDO
;

parameterList
:   repetition
|   duration
|   time
;

repetition
:   EVERY expression UNIT
;

duration
:   FOR expression UNIT
;

time
:   AT NUMBER ':' NUMBER
;

filterCondition
WHERE expression
;

term : value
| (' expression ')
| property
| IDENT
| commandStatement
| newPropertyStatement
;

value : NUMBER
| BOOLEAN
;

newPropertyStatement : LET IDENT BE '{' expression+ '}'
;

weakOperation : (NOT)* term
;

comparission : weakOperation (('=|'/='|'<'|'>|'<='|'>='|ABOVE|BELOW|NEXT)
weakOperation)*
;

expression : comparission ((AND|OR) comparission)*
;

componentUse : READ objectProperty parameterList* filterCondition?
| MAKE objectProperty TO value parameterList? filterCondition?
;

objectProperty : property OF object
;

property : IDENT
;

object : IDENT
;

// Lexer rules
fragment DIGIT  : '0'..'9';
fragment LETTER : ('a'..'z'|'A'..'Z');
Chapter 6. High Level Programming Language for WS&ANs

6.2.1 EPL Use-Scenarios

The concrete syntax of the language includes the basic operations that were described in the abstract syntax like the component use statement, which reflects the actions in the abstract syntax, and the condition statement, etc.

However, the concrete syntax have additional components which has no effect on the EPL syntax and semantics, like the Time component, which defines the duration and/or repetition of the tasks. These components are used after finding the set of assets that can be assigned to tasks to program them with the specific required behaviour. Because these components are used when generating the execution code for the assets (not a part of this thesis) they are not included in the EPL semantics.
To illustrate the use of the time components, consider an example for reading temperature of a room:

```
1  Application temperature
2  Read temperature of room1 every 10 sec for 1 day
3  End temperature
```

Figure 6-2 Temperature Monitoring Application

The user specifies the behaviour that he/she requires from the environment, in this case “retrieve the temperature of room1 every 10 seconds for a period of 1 day”. In such an example, and unlike other high level programming languages, the user does not need to specify the sensor that needs to perform the action: this can be carried out by the system using the Selfish gene algorithm introduced in the previous chapter to find the set of appropriate sensors.

Manipulating objects in real world using the actuators uses a similar syntax but with different keywords – in this case “make” instead of “read”. This can be seen in the concrete syntax of the language under the parser rules (or the grammar) for `componentUse`.

Table 6-1 shows some applications using EPL and describe their component use and operations. The syntax of the language will be described in the following section.

<table>
<thead>
<tr>
<th>Element</th>
<th>Use-case scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component use</td>
<td><strong>Application heater</strong>&lt;br&gt;<strong>make</strong> heater of kitchen to on at 12:30&lt;br&gt;<strong>End</strong> heater</td>
<td>Specifies that the heater in the kitchen area needs to be set to on at 12:30. The key word <code>make</code> indicates a program to be implemented on one of the actuators. The program would find the property or object and its location, the information will be used to allocate the actuators that can perform the specified action which then would then be translated into a code to be deployed on the actuator.</td>
</tr>
<tr>
<td>New property statement</td>
<td><strong>Let</strong> suspicious-Behaviour of garage be&lt;br&gt;<code>[read speed of car where carspeed &gt; threshold]</code></td>
<td>Define a new property of a garage which is not a part of the ontology, the new property defines a suspicious car to be a car driving at a speed higher than a predefined threshold. Defining a new property makes use of the underlying network to detect a property of the car (speed) that satisfy a particular condition (higher than threshold)</td>
</tr>
<tr>
<td>Filter condition</td>
<td><strong>Make</strong> lock of door to on below floor5</td>
<td>The filter condition imposes restriction on the area where an action is to be implemented. In this case, the application asks to lock all the door that are located below the fifth floor of the building.</td>
</tr>
</tbody>
</table>
6.3 Evaluation

6.3.1 EPL Formal Semantics Evaluation

This chapter presents an approach to formalize the dynamic semantics of a calculus for manipulating real world objects, which hides away the information about the underlying system from the user of the system.

First, the basic components and the notions of behaviour that is expected from the system for manipulating real world objects are identified. Once all the basic notions are presented, they were projected into building an abstract syntax for the EPL language. We defined the formal dynamic semantics which specifies the behaviour of the system. That is, for every notion, we have built one or more deduction rule. The rules present an incremental approach for defining the compound behaviour of the system.

Formal semantics is used as a basis for building an interpreter for the language. While the interpreter still does not translate the required behaviour from the end-user into a machine code to deploy on the required assets of the system, it translates the tasks into a set of missions that use the selfish gene algorithms introduced in the previous chapters to find the sets of assets which can perform each task. These assets, along with the behaviour of the system are interpreted later into a machine code to be deployed on the corresponding assets.

The interpreter is built using ANTLR (Another Tool For Language Recognition), a parser generator which can define the rules for translating the commands of the language to other general purpose programming language. The reason for choosing ANTLR is because it was built on top of Java and it can translate the user code into Java code. Since Jena, and Parliament—which are used for semantic and spatial reasoning in previous chapters—are both built on top of Java, ANTLR translates the user code into a set of missions which uses the defined algorithms.

The interpreter is used with a set of ontologies following the ontology integration presented in Chapter 4: IFC-based ontology for modelling space, SSN ontology for WS&AN and GeoSPARQL. Issuing a task like “read temperature of livingRoom” is performed by the interpreter and returns the minimum set of temperature sensors in the living room.

6.3.2 EPL Usability Evaluation

As the main purpose of the language or development tools is to be easy for non-programmers to develop their own programs and control their environment without the need for assistance from developers. The main test conducted on these tools is Users Experience evaluation (UE)—also
called User Experience Assessment (UEA). In UE, a random number of people were asked to develop applications using the tool and were asked to fill a questionnaire about the tool. Questions were asked to know how easy it was to develop an application using the tool, was it easy to find what they require, etc. Testing the usability of the different tools have not received any attention by the developers, and it has been only considered in evaluating the media cube tangible language. Usability is defined by ISO (International Organization for Standardization) as:

“The **effectiveness**, **efficiency** and **satisfaction** with which specified users achieve specified goals in particular environments”.

Even though it is important to use UE for evaluating a new programming language –especially a user-centric one–, developers need to not only count on UE for evaluating their tools or programming language, but also evaluate its usability on theoretical bases without user experience evaluation for many reasons. First users are not free all the time to evaluate the tool and when they evaluate the tool it should be clear of problems. Second, evaluating the tool with a low number of users might not detect all problems because users have different sets of problems that they aim to test, and with high numbers of potential users, a few tests will not cover all problems that are not tested. This problem is more challenging with a domain independent tool. Third, initial UE evaluation cannot evaluate the tool when the user becomes experienced with the system and wants to develop more complicated application. A good evaluation tool should be able to uncover problems in the design of the language which cannot be detected by UE.

To evaluate the usability of HCI we chose a tool called Cognitive Dimensions of notations (CDs). Introduced by T. Green and M. Petre [186, 187] as a broad-brush discussion framework for evaluating the usability of information based artefacts. Unlike other HCI evaluation methods, like cognitive walkthrough (which evaluates the interaction between the user and the system (like user interface) and thus requires a goal and specific set of steps making them not very suitable for programming languages); “a cognitive dimensions approach” is more suitable for evaluating programming languages when not much interaction is needed. However, cognitive dimensions can be used for both interactive and non-interactive systems. CDs provide a framework where our language—a regular programming language—can be discussed and compared to the interactive approaches introduced in literature.

CDs aim at answering the following question: “Are the users’ intended activities adequately supported by the structure of the information artefact?” They evaluate the system on 13 different dimensions and determine the suitability of the system to different tasks. One disadvantage of CDs is that they do not provide quantitative data about the system, as the CD framework is a discussion tool for evaluating the notation of the system. However, because CDs cover wide range of dimensions in the system, such as abstraction, visibility, viscosity, etc, they provide better coverage.
for evaluating the system than a UE does. Both approaches are important and complementary to each other, however, because UE does not cover all the various aspects that CDs cover, we chose to use CDs to evaluate our system.

### 6.3.3 Cognitive Dimensions

To evaluate a system using cognitive dimensions, the components of a system need to be identified, because evaluating the usability of information artefacts differs when examining different layers of the system. The components of every information artefact are: the notation, environment, medium and possible sub-devices. We compare EPL with four approaches that we described in the literature review; these are: CAMP [49], playing with the bits [68], ESPranto SDK [90], and media cubes [20]. We aim at using simple examples when applicable to clarify the evaluation.

EPL is a regular programming language and so its notation is that of a plain text. While the notation for CAMP is words on magnetic pieces, playing with the bits and ESPranto use jigsaw pieces. ESPranto, however, has another notation which is plain text at a lower level of development. The environment for all approaches is a UI, text editor of EPL and ESPranto macro programming, and graphical for the rest. The medium for all development tools is a screen. Unlike the previous approaches, media cubes is a tangible programming language –after the sketches are created and the program development happens using the cubes–. The notation of the cubes are the actions performed that can be performed by the cube such as do, when etc. The environment is the cubes themselves and the medium is the physical space where they are being used.

Different types of activities can be evaluated using CDs and no usability evaluation can be processed without defining what activity the artefact will be used for. The activity types that can be evaluated using CD are: incrementation, transcription, modification, exploratory design, searching, and exploratory understanding. Since we are considering tools to develop and program new applications, we are mainly interested in exploratory design.

The cognitive dimensions of the development tools are: abstraction, closeness of mapping, consistency, viscosity, visibility and juxtaposability, hidden dependencies, progressive evaluation, premature commitment and enforced look ahead, and secondary notation.

#### 6.3.3.1 Abstraction

Abstraction is treating a group of elements as one entity. Abstraction results in expanding the notation of the information artefact by introducing new terms.

CAMP deploy abstraction-tolerant system, meaning that the user can define new abstractions but it is not necessary. In CAMP, users can assign a new magnetic piece with specific name for every
program that they create and add it to the list of already available magnetic pieces. An example of defining a new abstraction is to define a dinner, period for example, as: “dinner happens between 7 PM and 9 PM ”, the new magnet dinner can then be used when building a new application like: “show me kitchen at dinner”.

Similarly playing with the bit allows to combine devices together to create a new device which can be exported to the database and be used as a device. For example, a doorbell can be connected to a camera and a display to create a new surveillance device. EPL and Espranto macro programming also allow abstraction by allowing new functionalities to be added to the system by programming new functions. In EPL, new properties of devices can be created and associated with its objects which then can be used and reasoned about as a new property. An example of a new property which can be exported to the system is:

```plaintext
let suspiciousBehaviour of garage be
(read speed of car in garage where CarSpeed > CarThreshold )
```

Figure 6-3 Defining new Property with EPL

*Media cubes* does not allow abstraction at the cube level because again once the sketch is deployed on the cube it can be used to build a specific program by an ordered set of interactions but no new components of new functionalities can be added to the cube. However, the functionality can be copied to another cube.

### 6.3.3.2 Closeness of Mapping

Represents the closeness of the representation or its restriction to the domain.

*CAMP* was introduced for the smart home scenario and all the words on the magnets are defined to the specific domain that they can be used in. *Playing with the bits* on the other hand, imports its concepts from a data base and creates jigsaw pieces based on the concepts of the database, which makes it not limited to a specific domain. *ESPranto* is also not closed of mapping (restricted or limited to a domain) as users can connect sensors and actuators and verify their behaviour under different environments and domains.

EPL is based on importing the ontology of a specific domain and enabling the user to develop applications in that domain, however EPL can deal with ontologies from any domain it is not domain dependant –similar to *playing with the bits*.

*Media cubes* is closed of mapping because it was designed to allow controlling home devices like an alternative to controls of VCR, TV, etc.
6.3.3.3 Consistency

Consistency implies that the same semantics of the development tool components are represented using the same syntactic form.

All the development tools that we consider in this comparison are consistent as the elements are always represented in the same way. This makes learning the development tool easier as the user gets more used to the tool.

6.3.3.4 Viscosity

Viscosity is the system’s resistance to change, it describes how difficult it is for users to make changes to their program.

Viscosity is very low when considering CAMP because there are little dependencies and no order for the commands in the language making changing the program very easy. Figure 6-4 shows a very simple example on how to change an application from one time to another using the CAMP interface.

Changing the application to a new time that is not already a part of the interface as a magnet is more complicated but it can be simplified using abstraction and creating new components to be added to the system, however, this will be discussed under the abstraction dimension.

When considering jigsaw piece development tools – playing with the bits and Espranto graphic editor –, changing the program is fairly easy. However, using jigsaw pieces makes it not possible to insert a new component into the middle of the program, the user would need to connect the jigsaw pieces where the new component needs to be added while paying attention to dependencies. Figure 6-5 shows the graphical user interface for developing applications using ESPranto SDK, having dropdown menus makes changing an application very easy.
With all the introduced visual programming languages viscosity is low and easy in small applications, as the application gets bigger which it would still be easy to change viscosity, it takes a long time to change every component of the program.

In textual languages like EPL and *ESPranto macro programming*, viscosity is higher as it requires rewriting or rearranging the order of commands, users also need to consider the syntax of the language.

An example of two programs written using EPL can be seen in Figure 6-6, while A is simple program and its viscosity is low, as the program becomes more complicated like B, it becomes risky to make changes to the program because the user needs to write all the changes.

Still viscosity is much lower in all development tools than that in media cubes, the reason is that the sketches for the programs are usually written using a computer and transferred to the cube to program the environment, this results in low visibility—the user cannot see the code—making it much more complicated to change the program.

```
program heater is
  set heater of office1 to on
end heater.
```

```
program suspiciousBehaviour1 is
  constant PersonThreshold = 2;
  constant CarThreshold = 40;
  let suspiciousBehaviour of garage be
    (read speed of car in garage where CarSpeed > CarThreshold ;)
  let suspiciousBehaviour of building be ((
    read speed of person in room where PersonSpeed > PersonThreshold ;) or
    (read suspiciousBehaviour of garage ;))
end suspiciousBehaviour1
```

Figure 6-6 Application Development using EPL
Viscosity is considered harmful for exploration activities where the user define new functionalities and programs to be added to the system. As development tools for controlling different aspects one’s environment, a development tool should not have high viscosity as it limits what the user can do.

One possibility to improve the viscosity of EPL language is to introduce a new annotation like a graphical user interface similar to the other development tools. This step has not been introduced yet and is one of the future work of the thesis.

### 6.3.3.5 Visibility & Juxtaposability

Visibility is the ability to view all the components of the system easily, while juxtaposability is the ability to place two components next to each other.

CAMP, *playing with the bits* and *ESPranto graphic editor*—in general all jigsaw pieces based UI—have good visibility if the domain or the number of items is small, as the number of pieces or magnets increases the visibility can drop dramatically. CAMP for example can have low visibility because it allows users to define new words or magnets which eventually will all be on the screen. *playing with the bits* imports devices from a database as jigsaw pieces so also as long the database is not very big, the visibility is high. Figure 6-7 shows an example about the different components and how they can be displayed in CAMP tool. Even though the user can see all the different components of the program and environment, increasing the number of components results in low visibility.

![Figure 6-7 CAMP Visibility](image-url)
Media cubes on the hand have no visibility at all; once the sketches are deployed there is no way to view the program. EPL is similar to any programming language, the visibility is good when the program is not very large. At the same time, syntax highlight helps visibility and distinguishing the different syntactic categories in the program.

Juxtaposability is good in all the GUI development tools as they allow having more than one component aside on the screen. EPL also has good juxtaposability because it allows more than one window for programming at the same time.

### 6.3.3.6 Hidden Dependencies

Hidden dependency is a non-visible relationship between two components of the system, where a change to one of them causes changes to the other. Pointers are considered as a one day dependency, visual programming languages introduce dependencies but they are explicit and not hidden. Playing with the bits for example with its jigsaw pieces, introduces dependencies where the action of one component affects the other similar to directed graphs.

EPL and ESPranto macro programming, like other textual programming language have hidden dependencies which we can see, in its simplest form, the assignment statement; one variable is assigned a value and is used elsewhere in the program. CAMP allows defining new components which can be used in other programs –already described in the abstraction dimension–, so do playing with the bits when composing new devices from existing ones. Both are types of hidden dependencies.

EPL also has hidden dependencies related to its underlying information structure. EPL allows reasoning about properties of objects like temperature of a room, the dependency between the property (temperature) and the object (room) that exists in the ontology, which is used by EPL to construct the program.

Media cube as a tangible programming language without the ability to read its program have all its dependencies hidden, however the dependencies in media cube are usually very few and dependant on the operation of the program in the environment.

Even though hidden dependencies have their advantages in simplifying a new program, when using an existing one, they also increase the risk of error. Adding the new dependencies as components to the tool increases the functionality of the tool as the user develops more with it but an underlying change to one component can cause a lot of damage as it can change the functionality and behaviour of other components and the program as a whole. Highlighting the dependencies between different components can be one approach to make updating and finding dependencies easier.
6.3.3.7 Progressive Evaluation

This refers to the ability of the system to check the work done at any time without the need for the program to be ready.

None of the approaches support progressive evaluation—including EPL; so far all approaches need the program (or a complete part of it) to be finished to be able to check it at run time.

6.3.3.8 Premature Commitment and Enforced Look-ahead

This refers to the constraints on the order of performing actions before proper information is available.

CAMP does not have premature commitment as the program can be written in any order. Playing with the bits, on the other hand, has little premature commitment because not all jigsaw pieces can be connected together, but there is a dependency between them. The fact that the component is presented in a jigsaw piece style makes premature commitment little, because even though the user needs to think ahead about which component can be connected, the shape of the pieces restrict and allow the connection to other pieces. The same arguments apply to ESPranto SDK graphical user interface.

EPL and ESPranto macro programming—similar to any programming language—do have premature commitment as the user needs to think of the order of actions to create a program, but that is dependent on the underlying environment as well.

Media cube strongly encourages building the program from button to top, starting with the smallest component, which means the user needs to think quite well ahead before building a program.

Premature commitment is very harmful especially for end-user development tools when the user is not an experienced programmer who can think ahead on the order of the operation required to build a program. Premature commitment makes developing an application very hard and challenging for such users and might drive them away from using the tool.

6.3.3.9 Secondary Notation

Secondary notation usually refers to another notation used by a programming language or development too. Secondary notations have no meaning to the compiler and no effect on the program, but they make it easier to read the program. An example of secondary notation is the indentation of programming languages as well as syntax colouring.

In CAMP, for example, the secondary notation can easily be found, as it groups the different magnetic words under different categories to make it easier for the user to construct their program.
CAMP’s 5 categories are: who, what, where, when, and general. The interface has different colours for the different categories to ease the process of finding words for the users.

*Playing with the Bits* does not have such secondary notation as all components are displayed in the same window, whereas ESPranto graphical interface uses another kind of secondary notation by using the notion of lists.

EPL and ESPranto macro programming, like other textual programming languages, provide secondary notation in terms of indentation and white spaces, which makes it easier to read the program.

Tangible languages like media cubes do not use such secondary notation; however, developers interpret external notes attached to the cube, such notes are a sort of annotation to the program and can be considered as a secondary notation.

### 6.3.3.10 Discussion

The qualities of EPL makes it suitable for developing most programming tasks of UbiComp environment, like, having no closeness of mapping which makes it suitable for a wide range of applications, its abstraction which makes it more suitable for non-programmers. However, other qualities still need further improvement to make it more suitable for future programming of UbiComp application by end-users such as, premature commitment to make sure that the developing a program is as easy as possible for end-users, progressive evaluation to make sure that different component are operating as intended before putting the program all together, viscosity to make developing a program easier by allowing changes to be performed easily.

### 6.4 Conclusion

In this chapter, a formal programming language was introduced to allow manipulating real world objects, while hiding away its technology from end-users. The formal syntax and semantics of the language were presented, as well as a concrete syntax of the language. An interpreter of the language was deployed which allows translating an end-user program into a mission required from the various assets in the network. The spatial mission assignment algorithms (already introduced in Chapter 5) is then used by the system to find the set of assets which needs to be programmed corresponding to the translated mission.

The proposed language has the advantages of being an abstract language that hides away the technology of future UbiComp from prospective users while dealing with task allocation and network resource optimisation in the background. The level of abstraction proposed goes one level beyond what’s already introduced in literature, by hiding not just the complexity of technology but
also the existence technology itself. This align the EPL language with how end-users expect their smart environment to be, and how they envision their interaction with objects of their world.

The notation introduced for the language (textual right now) was evaluated against a list of qualities to identify its strength and weaknesses as an end-user development tool. The set of qualities used to evaluate the EUD is called cognitive dimensions. Evaluating EPL with CD have showed that the language could provide better abstraction than other EUD and could be used for different domains. It also showed that there are other aspects of the language that could be improved to provide a more suitable EUD for UbiComp.

The following chapter reviews the introduced framework for programming WS&AN in full, points out what have been done and proven through these thesis and that possible future extension for the thesis.


Chapter 7

7 Conclusion and Future Directions

In this section, we revisit the whole system, discuss the novel contributions that it has introduced, their advantages and how they add to the state of the art. It also outlines a range of areas that the contributions of this thesis have opened up and describes the scope and directions of research that may be pursued in future projects.

In summary, this work introduced a top-down model to allow regular users to control the ubiquitous computing environment through the following contributions:

- A high level programming language for real world objects using wireless sensors/actuator networks in future ubiquitous computing environments.
- An integration of domain ontology and sensor ontology with spatial representation to allow reasoning about tasks with spatial properties.
- Two algorithms for task allocation in wireless sensor networks based on branch and bound optimisation algorithms and evolutionary algorithms using the spatial relations with tasks in the network.
- An algorithm for multiple spatial task allocation in sensor networks, with the aim of utilizing the performance of the network based on the kin selection theory of altruism in evolution and using the spatial relations between tasks and sensors.

7.1 Conclusion

With advances in technology, smart devices are everywhere around us, microprocessors are already integrated in so many objects that we stopped thinking about them as computing devices. With increasing numbers of smart devices a person owns or that are available everywhere, important questions are: How to deal with all these advanced technologies and how to use all the various devices from different vendors with different purposes? How to mediate the use of technology in one’s life while still taking advantage of all the potential use of the current technology and how they can make our lives easier.

In this work, we introduce a top-down generic model for developing smart environment applications in future ubiquitous computing. The aim is to make it easier for people and potential users of
UbiComp to take control of their environment easily by keeping technology and all technical details out of the way, while still allowing users to take full advantage of the technology.

Much research has been directed to keeping technology and the technical details hidden from the users making it easier for users to control their smart environment. The two levels of abstraction (hiding the technology and hiding only the technical details) have taken different approaches: for hiding the technology, machine-learning mechanisms have been introduced, whereas higher-level programming has been introduced to hide the technical details. Despite the success of machine learning techniques in hiding technology from users, it is not suitable to automatically control smart environment always; and users need to be able to control their smart systems when necessary. Higher-level programming, on the other hand, has mainly looked into hiding computation, communication protocols and details from the users while keeping the users interacting with computing devices. This approach works very well for scientists and domain experts who know their domains, how they want the technology to operate and what information/behaviour they require from their systems.

While both approaches do provide important steps towards enabling smart environment and ubiquitous computing, they still do not provide the required level of technology invisibility expected from an ideal smart environment that allows users to operate their system. The system that we propose does provide higher-level development tools that hide the technology from the end-users while allowing them to control their space when they need to. Even though our approach does not incorporate machine-learning techniques, both approaches are complementary to each other for future development. In his book “Digital Ground”, McCullough discusses the importance of participating users in the development of their systems, he says: “Know when to eliminate an obsolete “legacy” operation, when to automate, and when to assist an action. Know how to empower, not overwhelm” [42].

To enable end-users to take control of their smart spaces while hiding the technology from them, the technology needs to be integrated and represented to the system. This is accomplished in our approach by connecting the SSN ontology –to represent sensors/actuators in the system– to the domain ontology using the property of objects which the sensor/actuator can detect/manipulate.

This integration would hide technology from the users by enabling them to reason about domain related information provided by the sensors. However, one limitation of only connecting the technology representation to the domain representation is that the use of the system is limited to an already defined set of actions –as already seen in literature [49, 73]. To overcome this limitation, resource allocation approaches need to be used to find the sensor or set of sensors that are capable of performing the action required by the end user. The integrated ontologies take the system one step closer to domain independency by allowing about the system to choose the type of sensors that
can perform a specific action. The problem, however, is more complicated. Since sensors and objects have different locations—they can further be dynamic—, there is a need to know which sensors/actuators are able to perform the task based on their spatial properties. We integrated the two previously mentioned ontologies (SSN, Domain ontology) with GeoSPARQL, which is an ontology for representing and reasoning about spatial properties and their relationships. We use all three ontologies for allocating resources to the different tasks imposed by the end-users. To the best of our knowledge, no other approach in the current literature has considered the spatial region occupied by tasks as a factor for resource allocation, and so no spatial resource allocation approaches have been introduced. The ones that have considered spatial properties of the task have been mainly interested in one point location for tracking purposes.

While finding the set of sensors/actuator capable of performing a specific task is essential to perform a user’s task, the characteristics of the system should also be taken into account. For WS&ANs the most important characteristics to consider are their limited resources. Current resource allocation approaches for WSNs have taken this into consideration, aiming at extending the life span of the network by reducing the number of sensors assigned to tasks. However, they only consider (one or more) predefined tasks and they cannot cope with the issue when new tasks are imposed to the network. We introduce two optimisation algorithms that optimise spatial resource allocation of WS&ANs by finding the minimum set of assets that can perform any task, based on branch and bound and genetic algorithms.

To further optimise the performance of the network we also introduce a nested genetic algorithm with the aim of improving the utilization of the network when assigning assets to more than one task, while also reducing the number of sensors assigned to each task. By using the introduced algorithm, the system guarantees assigning the most suitable and minimal set of sensors to every task while taking their available energy level into account and keeping as many sensors as possible free to be assigned for additional tasks.

A disadvantage of the algorithm, even though it could guarantee to find a near optimum solution for the mission assignment problem, is that it uses an estimation of the sensor detection probability based on the spatial relations between the range of the task and the sensing/actuating region of the asset. While this estimation provides a better detection probability than the binary probability, it is still not ideal and not the accurate one. A proper sensor detection function that can make use of the spatial relations as well as other operational properties of every assets can be introduce to guarantee an accurate detection probability.

The system introduced allows end-users of ubiquitous computing to control their environments and participate in defining the various behaviours they require from different objects and aspects in their environments. The system also hides the technology which enables such manipulation from the
users. Hence, a system allows end-users to focus on the application they want to build and the required behaviour from their environment rather than focusing on the technology they need to use and how they can use it, which can be very demanding for non-expert users.

Because the system is built around an integrated ontology and uses the concepts of the domain ontology as its main objects that users manipulate in the high level language when reasoning about their system, the approach is generic and can potentially work in any environment –assuming the domain ontology is integrated with SSN and GeoSPARQL. Such a system when fully implemented will enable users to control any smart environment they have control over without the need for learning how to use new systems and new development tools at possibly different levels of abstraction.

7.2 Future work

This work has aimed at introducing a framework to manipulate real world objects in UbiComp based on a higher-level abstraction language, integration of ontologies and spatial resource allocation. While this has been achieved in this work, there are still many areas that the work can be extended to include and build upon. These are summarised as follows:

7.2.1 Testing

The EPL language has been introduced and tested for usability using the cognitive dimensions framework. While the evaluation demonstrates the usability of the proposed language, and since the language is an end-user language, testing user experience performance of the system testing is an essential part of any further development of the system.

7.2.2 Resource Allocation

The task allocation algorithms that we introduced can be further developed and optimised in two main directions: (i) spatial task allocation, (ii) additional factors for resource allocation.

7.2.2.1 Spatial Task Allocation

In terms of spatial properties and spatial task allocation, the algorithm estimates the detection probability of the sensors to various tasks based on the spatial relations between both regions, assuming that both the sensors and the task have regular regions regarding the space they occupy as well as their operational ranges. A better approach could actually measure the area of intersection between the task region and the sensor capability to find exactly the detection capability of the asset.
Different tools can be used to measure the intersection area between two regions as well as their intersection percentage, e.g. ArcGIS: a *Geographic Information System* (GIS) for creating maps, compiling, analysing and manipulating geographical information.

### 7.2.2.2 Additional Factors

The allocation problem that we have been able to solve using evolutionary algorithm tries to improve the performance of the network. This is accomplished by increasing the gain from assigning assets to tasks while at the same time increasing the life span of the network by reducing the number of assets assigned to every task, leaving redundant nodes in a sleep mode to reserve their energy.

The factors that we have taken into consideration when assigning assets to tasks are: the type of the task –which asset can perform the specified task–, and the detection probability of assets to tasks –based on the spatial intersection. In a real life implementation, additional features can have an effect on the performance of the network, which have not been taken into account in this work yet. These include but are not limited to: network topology –routing energy requirements–, operational range holes –due to any obstacles in the surrounding area–, remaining energy of the node (the formula already defined in Chapter 5, equation 5-5), and measurement capability under different situations.

The topology of the network is an important factor to consider when reserving the energy of the sensors. As data need to be transmitted through the network, some sensors that do not participate in the required task might need to only contribute to networking tasks such as routing each packet towards its destination. Packet transmission is the task that consumes the highest amount of energy in sensor nodes, and that is why it needs to be optimised to extend the life span of the network.

Obstacles in the range of a sensor node can affect its measurement capabilities. Static obstacles can be represented in GeoSPARQL by defining inner polygons inside the operational range polygon of the sensors; however, having moving obstacles is more challenging to solve.

Measurement capability of sensors are also affected by their surroundings: a camera for example cannot provide a reliable measurement when it is dark compared to when there is light. Taking into consideration the different elements and situations that can affect the capabilities of the sensors can be accomplished using SSN. The SSN ontology does define measurement capabilities of the sensors under different situations. If such information is to be provided in the ontology, it can be used for better utilizing the performance of a network.

All these various factors can be integrated in the optimisation problem to better allocate sensors to tasks; a weighted optimisation function can enable the users to define priorities for the allocation.
either to reserve energy and use the minimum set of nodes or to increase the reliability of the algorithm regardless of the number of sensors in use.

### 7.2.3 Time Integration

The proposed system integrates GeoSPARQL, an ontology to enable representing and reasoning about the spatial properties of objects with domain and SSN ontologies. A further possible extension to the system is the integration with a time ontology that enables representing and reasoning about time properties. Such an integration would enrich the system with flexible scheduling and will allow allocating tasks to various assets in the network anytime, anywhere.

Similar to the lack of spatial representation of sensor’s and task’s regions, the temporal properties of sensors and tasks are ignored when modelling domain ontologies and sensors ontologies. Time is considered as an external index that captures occurrences, and continuous time has not been modelled. An ontology that enables modelling the dynamic nature of reality is *Basic Formal Ontology* (BFO); a domain ontology can be mapped to BFO to represent both occurrences and continuous actions. We have proposed the extension of semantic sensor ontology (SSN) with the basic concepts introduced by BFO to allow spatio-temporal reasoning about sensors and their operations [188]. GeoSPARQL with other time ontologies –like Allen’s interval algebra– can be aligned with BFO to allow end-users to control the schedule of their system’s behaviour.
Bibliography


Bibliography


