REDUCING COMPLEXITY IN DEVELOPING WIRELESS SENSOR NETWORK SYSTEMS USING MODEL-DRIVEN DEVELOPMENT

Aymen J. Salman

School of Computing, Science and Engineering

College of Science and Technology

University of Salford, Salford, UK

Submitted in Partial Fulfilment of the Requirements of the

Degree of Doctor of Philosophy

Augest, 2017

Table of Contents

List of Figures

List of Tables

List of Code Snippets

List of Equations

List of Abbreviation

3SUF	3-Scale User Feedback
5SUF	5-Scale User Feedback
ADC	Analog-to-Digital Convertor
AM	Active Message
AN	Analysis
ANTLR	ANother Tool for Language Recognition
ArchWiSeN	Architecture for Wireless Sensor and Actuator Networks
AST	Abstract Syntax Tree
ATL	Atlas Transformation Language
BPML	Business Process Modelling Language
CFG	Context-Free Grammar
CG	Code Generation
CGC	Code Generation Component
CIM	Computation-Independent Model
CPC	Code Parsing Component
CTP	Collection Tree Protocol
DA	Data Aggregation
DARPA	Defence Advanced Research Projects Agency
DE	Domain Experts
DF	Data Fusion
DI	Dependency Injection
DSL	Domain-Specific Language
DSML	Domain-Specific Modelling Language
DSN	Distributed Sensor Network
EMF	Eclipse Modelling Framework
ENBNF	Extended Backus-Naur Form
ENVML	Environment Modelling Language
Eobject	Eclipse Object
EOS	Embedded Operating Systems
EU	End Users
FQAD	Framework for Qualitative Assessment of DSLs
G	Group of Nodes-Level
GCC	GNU Compiler Collection
GPL	General Programming Language
GQM	Goal/Question/Metric
HAA	Hardware Abstraction Architecture
HAL	Hardware Adaptation Layer
HCM	Halstead Code Complexity Measurement
HIL	Hardware Interface Layer
HPL	Hardware Presentation Layer
IDE	Integrated Development Environment

IoT	Internet of Things
IQ	Introductory Questions
JVM	Java Virtual Machine
LOC	Lines of Code
LWiSSy	Domain Language for Wireless Sensor and Actuators Networks Systems
M2T	Model-to-Text
MCL	M-Core Control Language
MCU	Microcontroller Unit
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MOF	Meta Object Facility
Moppet-FM	Moppet-Feature Modelling
Moppet-PE	Moppet-Performance Estimation
N	Node-Level
ND	Novice Developers
Net	Network-Level
NODEML	Node Modelling Language
OMG	Object Management Group
OTAP	Over the Air Programming
PIM	Platform-Independent Model
PSM	Platform-Specific Model
PTIR	Participant's Tasks Implementation Results
SAML	Software Architecture Modelling Language
SenNet	Sensor Network
SMS	Systematic Mapping Study
SNRA	ISO/IEC 29182 – Sensor Network Reference Architecture
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SUS	System Usability Scale
T2M	Text-to-Model
ULP	Ultra-Low Power
UML	Unified Modelling Language
UX	User Experience
VM	Virtual Machine
WSAN	Wireless Sensor and Actuator Network
WSN	Wireless Sensor Network
XML	Extensible Markup Language

ACKNOWLEDGMENTS

First of all, over the past three years, I have received valuable support and encouragement from my supervisor at the University of Salford. Therefore, I need to state my unwavering gratitude to **Dr Adil Al-Yasiri**, who has provided me with perpetual support, expert advice, and for showing unlimited patience and trust in myself.

I would like to thank my father (**Dr Jaber Aziz)** and my mother for their love, care and support in making me the person that I have become, and to give my endless thanks for their persistent encouragement throughout the duration of my education, both past and present, although my appreciation cannot compare with their sacrifices and the unconditional motivation that they have instilled.

Without the unwavering support, patience and understanding of my loving family (**Maisa** & **Mustafa**), this PhD would not have been possible. You will get me back soon!

I gratefully acknowledge the generous financial support of The Iraqi Ministry of Higher Education and Al-Nahrain University in sponsoring me through my research.

Above all, I would like to thank Almighty Allah for giving me the strength and wisdom to complete this research.

DECLARATION

I unequivocally declare that the contents of the present research are of original quality, apart from any specific references that are made regarding other researchers. This thesis has not been previously submitted for consideration to my current university or any other one. The entire content of the research is my own personal work and nothing has been formulated in collaboration with any other person, unless this is clearly specified.

Signature: _____ Date: ____/____/____

Aymen J. Salman

ABSTRACT

Wireless Sensor Network (WSN) is a collection of small and low-powered gadgets called sensor nodes (motes), which are capable of sensing the environment, collecting and processing the sensed data, and communicating with each other to accomplish a specific task. Moreover, all sensed and processed data are finally handed over to a central gathering point called a base station (sink), where all collected data are stored and can be reviewed by the user.

Most of the current methods concerning WSN development are application or platform-dependent; hence it is not a trivial task to reuse developed applications in another environment. Therefore, WSN application development is a challenging and complex task because of the low-level technical details and programming complexity. Furthermore, most WSN development projects are managed by software engineers, not application field experts or WSN end users. Consequently, WSN solutions are considered expensive, due to the amount of effort that has to be put into these projects.

This research project aims to reduce the complexity in developing WSN applications, by abstracting the low-level technical and programming details for average developers and domain experts. In this research, we argue that reducing complexity can be achieved by defining a new Domain-Specific Language (DSL) as a new application development and programming abstraction, which supports multi-levels modelling (i.e. network, group, and node-level). The outcome of this work is the definition of a new language called SenNet, which is an open source DSL programming abstraction that enables application developers to concentrate on the high-level application logic rather than the low-level complex details. SenNet was developed using the principles of Model-Driven Development (MDD) and macro-programming. Developers can use SenNet as a high-level programming abstraction to auto-generate a ready-to-deploy single node nesC code for all sensor nodes that comprise the SenNet application. SenNet gives developers the flexibility they need by offering them a broad range of predefined monitoring tasks and activities, enabling developers to develop different application types such as Sense-Forward (SF), and Event-Triggered (ET); besides providing a set of node-level and in-network data processing tasks. The current SenNet version is configured to generate nesC code, yet SenNet can be set up to produce and generate any programming language such as Java, or C++, by reconfiguring the code generator to produce the new language format, without changing the language design and produced semantics.

Various tests and user study have been used to evaluate SenNet's usability and functional suitability. Evaluation results found that SenNet could save 88.45% of the LOC required to be programmed by a developer, and 87.14% of the required vocabularies. Furthermore, results showed that SenNet could save 92.86% and 96.47% of the program length and volume respectively. Most of the user study participants (96%) found SenNet to be usable and helps to achieve the required WSN application with reduced development effort. Moreover, 82% of the participants believe that SenNet is functionally suitable for WSN application development. Two real-world business case studies developed were used to assess SenNet's appropriateness to develop WSN real applications, and how it can be used to develop applications related to data processing tasks.

Based on the final evaluation results, it can be concluded that our research has been successful in introducing SenNet as a new abstraction to reduce complexity in the WSN application development process.

KEYWORDS

Wireless Sensor Networks, Application Development; Wireless Sensor Networks Application Development; WSN Application; Domain-Specific Language; Model-Driven Development; WSN Programming; WSN Application Development; Multi-Level Abstraction; Macro-Programming; Node-Level Programming; Network-Level Programming; Code Generator; Node-Level Application Development; Network-Level Application Development; Group of Nodes-Level Application Development; Domain Experts; Sensor Network; SenNet Meta-Model; Code Parsing Component; Code Generation Component.

Chapter 1: Thesis Introduction and Research Methodology

1.1 Introduction

In recent years, WSNs have received considerable attention in the research community and this is one of the most researched technology areas in the last decade (Rawat, Singh, Chaouchi, & Bonnin, 2014). In terms of business opportunities, it is expected that the WSN market will grow from $0.45 billion in 2011 to $2 billion in 2021 (Harrop, 2012; Harrop & Das, 2012).

A Wireless Sensor Network (WSN) can be defined as a group of sensor nodes (motes); each node consists of one or more sensors, memory to store the sensed data, a radio for sending and receiving data and a processor. The essential capabilities of the sensor node are to sense, compute and communicate data from the environment in which the motes are working. The aim of a WSN is to combine all these functions into one single small chip. Generally, WSNs have limitations in terms of resources, such as power and memory. These limitations and the uncontrolled nature of this type of network have hampered efforts to develop the technology and, due to its compact nature, one of the complex tasks in a WSN is its programming and application development (De-Farias, Brito, et al., 2016; De-Farias, Li, et al., 2016; Lopes & Martins, 2016).

The general practice of Application development in WSNs is far removed from Software Engineering practices and standards, which are built monolithically, becoming very complex to understand and managed through the development phase (Intana, Poppleton, & Merrett, 2014). Therefore, a clear separation of concerns is required.

1.2 Motivation

1.2.1 Research Motivation

According to an MIT Technology Review article (Culler, 2003), WSN was considered to be *"one of 10 emerging technologies that will change the world"*. Afterwards, community interest and involvement in WSN increased and became similar to the Internet revolution (Riva & Borcea, 2007). Many research foundations around the globe have concentrated their

endeavours on different aspects related to WSN technology. Today WSNs have been produced for an extensive variety of applications for environment monitoring, industries, health, building, structural monitoring, military and security surveillance (Bellifemine et al., 2011; Yaacoub et al., 2013; Zhang et al., 2011; Zouinkhi et al., 2014).

WSN is becoming more and more integrated into larger computing infrastructures (Julien & Wehrle, 2013). Despite the challenges that WSN developers experience at every step in their application development, the research community is continuously working to introduce new solutions for WSN challenges. It has been noted that many programming approaches have been proposed, and many operating systems have been introduced as an abstraction level, such as TinyOS, Contiki and LiteOS, to be used by application developers.

In the face of all the efforts spent in this field, up to now designing and implementing software for WSN has been considered a complicated and challenging task (Antonopoulos et al., 2016; Cecchinel et al., 2016; Delamo et al., 2015; Essaadi et al., 2017; Lopes & Martins, 2016; Ouadjaout et al., 2016).

De-Farias et al. (2016) argue that the challenges in WSN application development are due to the lack of appropriate abstraction layers used to reduce complexity in application development. Accordingly, more research should be focused on this area. The motivation of this research is to make further contribution to this field and present an approach to WSN application development in order to reduce the complexity found in this area.

1.2.2 Motivations for a Proposed Solution

Many technologies, techniques and methods can be integrated and formed in a way to produce solutions for our research problem. But, some factors have encouraged us to build and introduce the proposed solution in this research:

- Finding a method to hide and personalise specific knowledge such as WSN low-level technology background and the required programming expertise, for people who do not have this required knowledge, is the particular contribution of this research. The Model-Driven Development (MDD) approach offers a solution to the problems mentioned. It offers advantages such as a separation of concerns, hiding low-level details and raising the level of abstraction (Hailpern & Tarr, 2006; Tei et al., 2015). So, using the MDD approach to developing Domain-Specific Language (DSL) is one of the best solutions to reach this

proposal (Essaadi et al., 2017). A DSL may allow users with minimum knowledge to program and develop the required application or configure the system appropriately (Meana-Llorián et al., 2016).

- Many researchers focused on using TinyOS (Levis et al., 2005) embedded operating system and nesC (Gay et al., 2003) programming language as foundation for their research projects by defining an abstraction development or programming (Essaadi et al., 2017), such as the work in Ravichandran et al. (2016) and Tei et al. (2015). Using TinyOS and nesC as a basis to build an application development abstraction is very useful (Salman & Al-Yasiri, 2016a, 2016b) because:

    1. Utilising TinyOS components and services. TinyOS is one of the most frequently used operating systems in WSN applications because of its maturity and continuous technical support. So, there is no need to re-build new components similar to TinyOS components.

    2. TinyOS presents a Hardware Abstraction Architecture (HAA) (Handziski et al., 2004) that includes a Hardware Presentation Layer (HPL), a Hardware Adaptation Layer (HAL) and a Hardware Interface Layer (HIL). This HAA enables TinyOS to support a broad range of sensor node hardware types. The updated list of supported hardware can be found on the TinyOS website[1]; more detailed discussion regarding HAA can be found in section **4.4.1**.

1.3 Analysis of Challenges in WSN Development

The research community has investigated in depth the different challenges of developing WSN applications, such as the work presented by Mottola & Picco (2011b), Randhawa (2014), Malavolta & Muccini (2014), Kumar et al. (2014), Mahmood et al. (2015), and Essaadi et al. (2017).

In this thesis two problems are identified, the first is the complexity in application development caused by the nature of the WSN technology and the limitations of the existing application development environments and processes. The second problem is related to the type of developers, and their knowledge gap in WSN programming skills and technology background,

[1] http://tinyos.stanford.edu/tinyos-wiki/index.php/Platform_Hardware

which needs to be filled to enable those developers to build WSN applications successfully. In the following sections these two problems are explained and analysed in depth.

1.3.1 Challenging Dimensions of WSN Application Development

The main WSN application development challenges can be presented in two dimensions: accidental complexity and inherent complexity; the main difficulties incorporated within these dimensions are shown in Table 1-1.

Table 1-1: Summary of WSN Application Challenges

Dimension	Challenging Aspects	
Accidental Complexity	AC1	The lack of adequate development methodologies
	AC2	Restricted Software Architectures
	AC3	WSN programming complexity
WSN Inherent Complexity	IC1	In-network processing or distributed computing
	IC 2	Sensor node mobility
	IC 3	Network Robustness
	IC 4	Location awareness
	IC 5	Multiple Gateway setups

1.3.1.1 Accidental Complexity Dimension

This dimension represents the limitation in programming tools and techniques because of the overall complexity of WSN development (Cecchinel et al., 2016; Essaadi et al., 2017). This has led to a gap between application developers on the one side, and application users or application domain experts on the other. Despite the importance and value of WSNs, software application development is nevertheless viewed as one of the primary challenges that faces WSN deployment (Ferro, Silva, & Lopes, 2015; Lopes & Martins, 2016), and many of the proposed software architectures for WSN have fallen short when it came to deployment, because of:

AC1. **The lack of adequate development methodologies**. Many of the developed software applications for WSNs have failed in field deployment (Antonopoulos et al., 2016; Geihs, Mottola, Picco, & Römer, 2011; Intana, Poppleton, & Merrett, 2015). This is due to the lack of development methodologies (De-Farias, Brito, et al., 2016; Lopes & Martins, 2016), which threatens confidence in the correctness, dependability and performance of the produced software (Julien & Wehrle, 2013). Losilla et al. (2007) explained that most of the current proposed architectures for WSNs concern a specific domain and were built from scratch following an experience-based method. Besides,

they are focused on the implementation steps, and they do not use any software methodology through their development. Moreover, to date, there is no *de facto* standard for application development in the WSN domain (Delicato, Pires, & Zomaya, 2014).

AC2. The existing restricted software architectures. WSN applications' software architectures can adopt many architectural styles according to the required application nature, such as service-oriented architecture (SOA), distributed database, or mobile agent architecture. The ISO/IEC 29182 standard (ISO/IEC, 2014a) has defined a reference architecture for WSN, which highlights middleware as the most common architectural style for WSN applications. Each one of these architectural styles needs a special type of requirement in addition to a specific programming characteristic. Many researchers have proposed various types of fixed software architectures and their suitable programming approaches. Most proposed architectures are restricted to a specific architectural form or template that may not fit with all WSN applications, such as in WASP (Bai et al., 2011). Where fixed seven-standard architectural templates have been defined, and each one supports a set of WSN activities, the developer has to start the application developing process by choosing the most appropriate architecture template, where the developer will be restricted to the WSN activities that are given by this architecture template.

AC3. WSN programming complexity. Developers have to deal with the status of the sensor node in terms of memory addresses and registers, because of the limited resources in this technology (Cecchinel et al., 2016; Essaadi et al., 2017; Mottola & Picco, 2011), which leads to the fact that the programming details are too complex, difficult to learn and maintain. Most of the current proposed programming solutions and development processes do not have the required abstraction level that satisfies domain application experts and average developers. A sample of academic articles that recognise this type of difficulty shown in APPENDIX B.1.

1.3.1.2 Inherent Complexity Dimension

This dimension represents the challenges in programming some WSN features, due to their complex nature. Adding or changing program variables and hardware registers may be easier to handle than mobility, which is considered very difficult to program (Essaadi et al., 2017; Ouadjaout et al., 2016). There are many criteria that have not been considered in most of the programming solutions and development processes:

IC1. In-network processing or distributed computing (Chandra & Dwivedi, 2015; Malavolta & Muccini, 2014; Tei et al., 2015).

IC2. Sensor node mobility: Most of the current proposed solutions consider static nodes only (Beal, Dulman, Usbeck, Viroli, & Correll, 2012; Gu, Ren, Ji, & Li, 2016; Malavolta & Muccini, 2014).

IC3. Show robustness in the network when hardware fails or message loss (Afanasov, Mottola, & Ghezzi, 2014; Chandra & Dwivedi, 2015; Mottola & Picco, 2011).

IC4. Location awareness, where Malavolta & Muccini (2014) surveyed 16 different MDD and DSML projects and found that 13 out of 16 did not support location awareness.

IC5. Multiple Gateway (Sink) setup in shared sensor network, and how the network will decide the data to which Sink node should be forwarded (Beal et al., 2012; De-Farias, Li, et al., 2016).

1.3.2 Potential Developers of WSN Application

The main factors needed to develop a successful WSN application can be broken down into:

- WSN technology background: which represents information related to low-level details, such as sensor node hardware types, sensors hardware types, available routing protocol, how to convert sensor readings into an acceptable informative data form, sensor nodes communication details, memory details included in the sensor nodes, besides the reading and writing mechanisms.
- WSN programming skills: which represents the ability to use a low-level programming language to program all the above details.

According to the literature, potential WSN application developers can be categorised as below and shown in Table 1-2 and Figure 1-1:

Table 1-2: Potential WSN Developer Groups

Potential Developer Groups	
PD1	WSN Experts & Scientists
PD2	General Application Developers & Programmers
PD3	Novice Developers (ND)

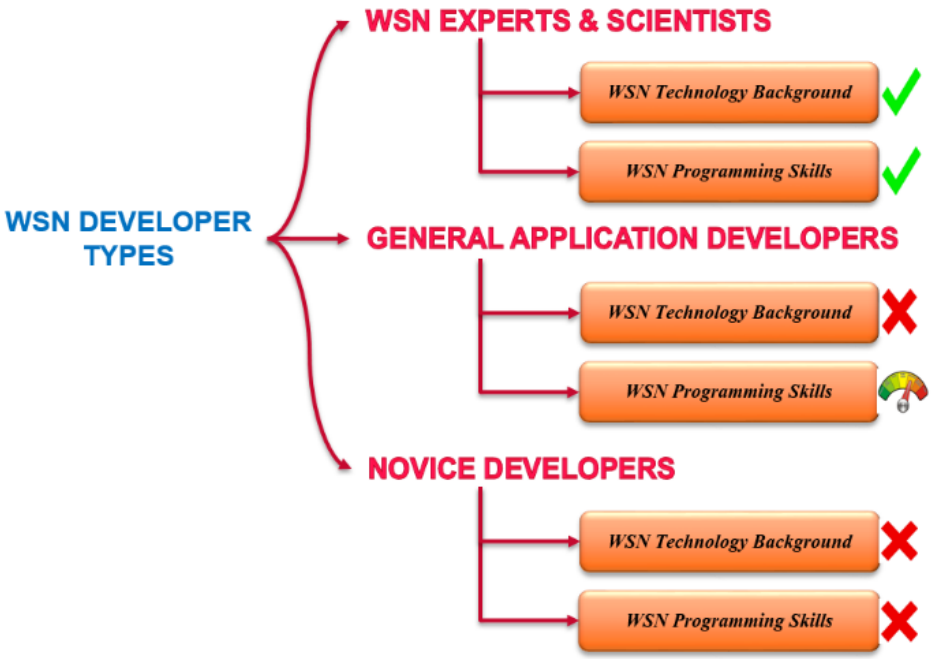

Figure 1-1: WSN Application Developers Technology Background and Programming skills

PD1. WSN Experts & Scientists: This group includes the researchers and scientists responsible for developing this technology, who therefore have all the required information for WSN technology low-level details and programming.

PD2. General Application Developers & Programmers: This group of developers includes general developers and programmers who have experience in general programming languages such as C, C++ and Java (Elsts, 2013). This group may have some skills in low-level programming, or they may acquire these skills quickly, but they do not have a WSN technology background.

PD3. Novice Developers (ND): This group of users includes Domain Experts (DE) and End Users (EU). Many researchers agree and use this term for specific WSN users with no or limited WSN technology and programming skills. Therefore, they target their research effort to support this type of users. A sample of the work focused on Novice developers can be found in APPENDIX B.2. This type of developers include Domain Experts (DE), who are people experts and scientists in fields other than WSN, such as civil engineers, biologists and environmental scientists not involved in networking (Ammari, 2014; De-Farias, Li, et al., 2016). Those people are willing to develop WSN applications to invest the WSN functionality in their domain projects. Therefore, a DE always needs to be involved (Kabac, Consel, & Volanschi, 2016). Accordingly, there

is a gap between the WSN application high-level requirements (the application-level knowledge) and the complexity of operations in the underlying network infrastructure (the network-level knowledge). Many researchers concentrate their research efforts on facilitating the WSN application development process for DE, such as the work presented in APPENDIX B.3.

Members of PD2 and PD3 are lacking in WSN technology background and programming experience. Therefore, members of these groups should acquire technical background and learn new programming languages to program sensor network applications. According to Robins, Rountree, & Rountree (2003), beginner programmers take about ten years of practice to write an application efficiently, which means that there is a big difference between a programmer that can write an application and another that can program the application in an efficient way. Therefore, the gap between beginners and experts is huge in WSN application development, because WSN developers not only need to improve their programming skills, but also to acquire technology background.

According to the aforementioned facts, this research is targeting PD2 and PD3 among potential WSN developers.

1.3.3 Demonstration Case Study (Scenario)

Considering these application development challenges in the WSN technology, it is clear that PD2 and PD3 of the potential developers in WSN will have difficulty in developing a successful application without hiring a highly expensive WSN developer expert. That will lead to many small projects ending before they start. As an example, take the extracted piece of code (Buonadonna, Tolle, & Gay, 2010) shown in APPENDIX C. This source code is part of the program that has to be implemented to develop a Base station application (Sink). From a beginner developer's[2] point of view, this piece of code will be a spaghetti code, which will not be an easy task to be developed or maintained by a beginner developer such as Domain-Expert or a general application developer.

[2] Beginner Developer: this term will be used through the course of this thesis to denote **PD2** and **PD3** developers from the Potential Developers for WSN Application

1.4 Research Problem

The function of a WSN is to observe the surrounding environment (Marques, da Silva Teofilo, & Rosa, 2013), whereas the original vision of WSN consisted of a large number of randomly distributing small devices over a large area to enable ad hoc measurements (Oppermann, Boano, & Römer, 2014). WSNs have had a high impact on our daily life activities, and as a result, the research community has tried to invest most of their knowledge and their research efforts to leverage and abstract the development and the programming of WSN applications.

However, according to the WSN application development challenges presented in sections **1.3.1** and **1.3.2** it has been noticed that these efforts have not solved the complexity of the application development for WSNs and could not make it affordable for all types of developers.

Taking all the factors mentioned above into account, the following research problem has been identified:

"How can developers who have no or limited knowledge of WSN technology background and programming skills be empowered to efficiently develop a WSN application using a development abstraction that provides the required expert knowledge for the WSN technical low-level details and programming experience".

1.5 Research Aim and Objectives

In accordance with the field of WSN programming and the application development challenges and limitations that were discussed in the previous sections, our research aim is to:

"*Investigate the possibility to reduce the complexity in WSN application development by introducing a new application development abstraction using model-driven development.*" According to the research problem statement and research aim, three of the research objectives have been addressed:

O1. Introduce a new model that logically links the application scenarios and tasks with the sensor network elements and the available resources. This model will be the basis to implement the second objective.

O2. Define syntax and semantics for a new Domain-Specific Language (DSL) that can be used to facilitate the development process for WSN applications, especially for developers with limited or no WSN programming experience and technology background. The new DSL required characteristics can be summarised as:

A. Provide a high-level abstraction for complex tasks and operations.

B. Embed data processing tasks and activities.

O3. Ensure the new model and language is usable and fit for purpose to develop WSN applications.

1.6 Research Scope

Our research focus to help beginner developers to develop their required projects (small to medium size projects) as they are facing difficulty in developing a successful application without hiring a highly expensive WSN developer expert. That will lead to many small projects ending before they start. Besides, focusing on the environmental monitoring applications, because they are considered the essential application types that related to WSN technology.

1.7 Contributions to Knowledge

Our most significant research contribution can be summarised as defining a new development abstraction that helps developers who do not have the required knowledge and skills to develop their required WSN applications. Introducing a high-level and flexible development environment will encourage and give developers the necessary confidence to develop different WSN applications, which will lead to a high impact on WSN technology uses. Therefore, our research contribution can be divided into three sub-contributions:

CK1. **SenNet Meta-Model**. This meta-model represents a high-level semantic model that can help beginner developers to successfully develop a WSN application, besides introducing the logical relationship between the application scenarios and the existing domain objects. SenNet Meta-Model can be divided into Application Configuration, and Network Configuration, which reflects how the logical thinking in specific application scenarios will be reflected on the real network components; more details regarding this sub-contribution can be found in Chapter **3**. This step contributes to **O1**.

CK2. SenNet Language. Use Model-Driven Development (MDD) to embed the SenNet Meta-Model and produce a SenNet language that can be used to facilitate the development process for WSN applications, especially for developers with limited or no WSN programming experience and technology background. The main SenNet characteristics are summarised below; more details regarding this sub-contribution can be found in Chapter **4** and Chapter **5**, and this step contributes to **O2**.

A. Using the macro-programming mechanism to abstract complex tasks and operations for the purpose of capturing the required application logic and the flow of data.

B. Enable developers to develop a node and network level applications, providing them with different modelling scopes (node, group of nodes, and network).

C. Generate ready-to-deploy single node nesC source code for all sensor nodes that comprise the SenNet application.

D. Define network and node-level data processing functions and present them as ready-to-use tasks.

CK3. Usability and Functional Suitability Evaluation. Investigate and prove the new language's usability by beginner developers (i.e., developers with no or limited knowledge of WSN technology background and programming experience). In addition, evaluate the new language capability to reflect WSN domain concepts and scenarios. The proposed model (SenNet Meta-Model) is evaluated inherently, as it is deep-rooted in SenNet language. More details regarding this sub-contribution can be found in Chapter **6**. This step contributes to **O3**.

Table 1-3 shows the main WSN challenges that are targeted and how they are mapped into research objectives, as well as to the contribution of this thesis.

Table 1-3: Mapping Research Challenges to Research Objectives and Contributions

Research Challenges	**Accidental Complexity Dimension**			**Inherent Complexity Dimension**					**Potential Developers**			
	AC1	**AC2**	**AC3**	**IC1**	**IC2**	**IC3**	**IC4**	**IC5**	**PD1**	**PD2**	**PD3**	
Research Objectives	O1	-	O2 (A)	O2 (B)	-	-	-	-	-	O2	O2	O3
Contribution	CK1	-	CK2 (A, B, C)	CK2 (D)	-	-	-	-	-	CK2	CK2	CK3

1.8 Research Methodology

The aim of this research is to investigate the research problem identified in section **1.3** and propose a development and programming abstraction that will enable beginner developers to develop their required applications for the WSN domain. Scientific experimental research strategy has been adopted for our research as shown in Figure 1-2. Our research uses the qualitative/quantitative method, as the research problem formalisation uses historical data (documents analysis), while the final results were produced quantitatively using analytical methods, experiments and quantitative user study. The research strategy followed is scientific experimental. According to Novikov & Novikov (2013), scientific research includes three main phases:

- **Design Phase**: This phase includes deciding the problem domain, formalising the research problem and identifying research aim and objectives.
- **Technological Phase**: This phase includes preparing the required theory, developing the proposed system and finally applying the evaluation methods.
- **Reflexive Phase**: This phase is related to finalising the results and preparing the final guidelines, then presenting them to the research community.

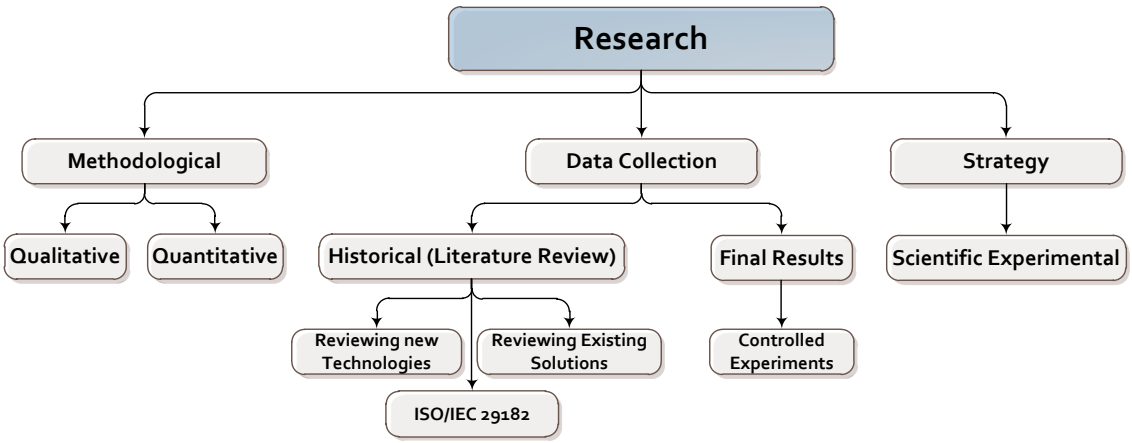

Figure 1-2: Research Paradigm

Each phase has been divided into a number of stages; a brief description of each stage can be found below, and Figure 1-3 shows the research stages as a flowchart.

1. **Design Phase**
   A. **WSN Application Development:** This was the first stage in our research: identifying WSN application development as the main research domain.

B. **Reviewing Literature:** Investigate earlier researchers' work about WSN application development and programming to get a good understanding and to know the state of the art of WSN; furthermore, to highlight the drawbacks of this technology.

C. **Formalising the Research Problem:** This stage identified the needs of WSN for new techniques to develop WSN applications, recognising the urgent need to enable beginner developers to play a bigger role in the process of developing WSN applications and maintenance. In accordance with these findings, the research aim has been addressed as defining a new semantic model and a new DSL for the WSN domain, as well as a set of intentions such as enabling in-network processing in the suggested language.

2. **Technological Phase**

A. **Investigate the Existing Application Development and Programming Approaches:** This stage was the investigation and analysing work that was applied to current application development techniques related to WSN, and highlighting their limitations.

B. **Develop SenNet Meta-Model:** This stage was identifying a semantic model that reflects the main WSN application activities, tasks and their logical relationships to network elements, by analysis of the ISO/IEC29182: Sensor Network Reference Architecture (SNRA) standard, as well as the application development methods and techniques proposed by the research community.

C. **Define SenNet Language:** This stage was defining the required SenNet language syntax and semantics. This was done using MDD and macro-programming techniques to increase the level of abstraction that would be offered to the developers.

D. **Evaluating the Proposed Language:** The assessment and verification of SenNet language are crucial to this research, where its usability and functional suitability should be ensured. Many assessments and testing methods were applied to examine SenNet language according to a predefined evaluation plan.

E. **SenNet Meta-Model and Language Revision:** According to the findings and outputs of the evaluation plan, amendments and modifications were applied to the proposed SenNet meta-model and language. This stage helps to enhance the results that are produced by this research.

3. **Reflexive Phase**

A. **The final conclusions and recommendations:** The last phase of this research is to record the final findings and recommendations.

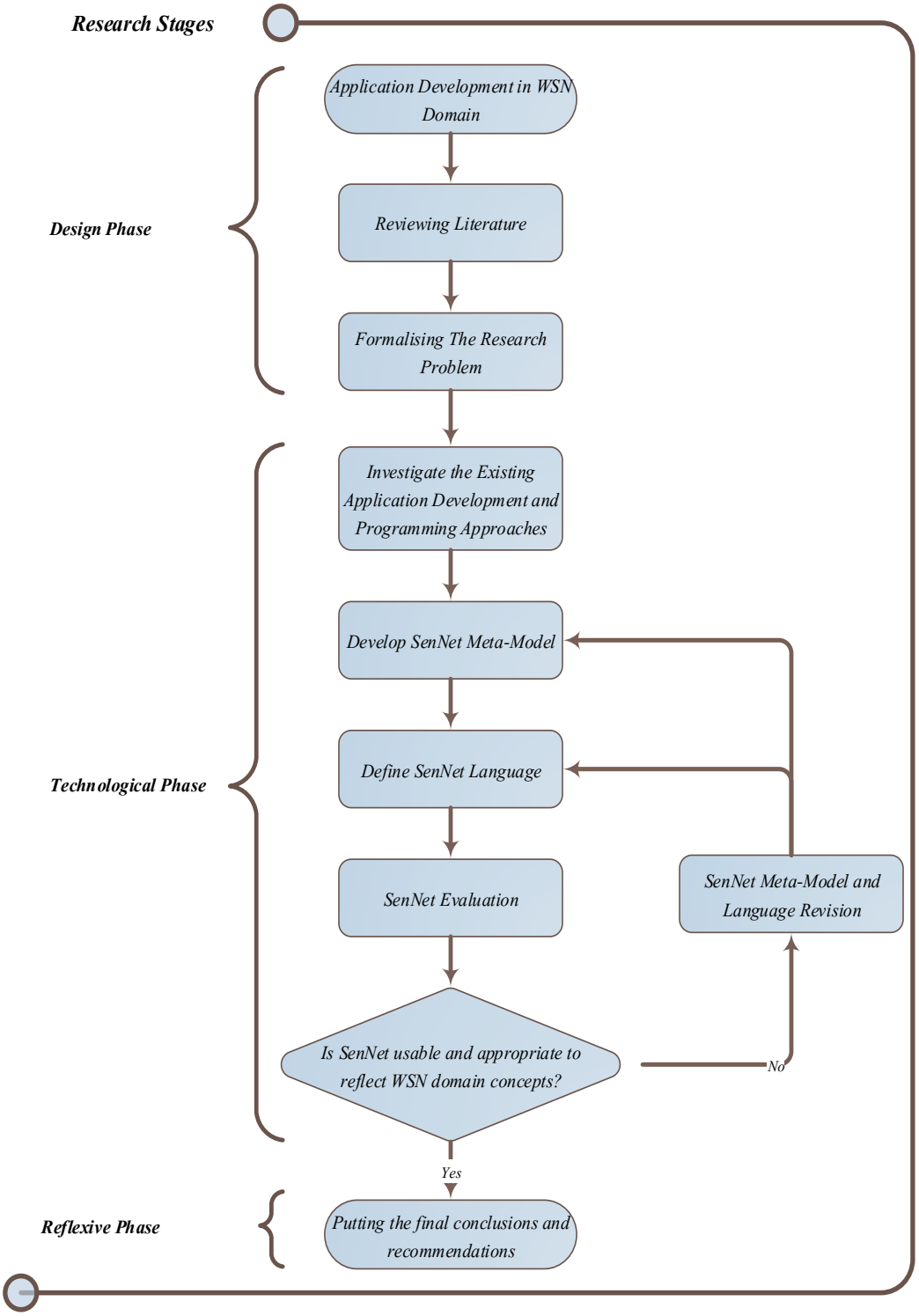

Figure 1-3: Research Main Stages

1.9 Publications

- Salman, A. J., & Al-Yasiri, A. (2016). SenNet: A Programming Toolkit to Develop Wireless Sensor Network Applications. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (pp. 1–7). Larnaca-Cyprus: IEEE. http://doi.org/10.1109/NTMS.2016.7792476.
- Salman, A. J., & Al-Yasiri, A. (2016). Developing Domain-Specific Language for Wireless Sensor Network application development. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)* (pp. 301–308). Barcelona, Spain: IEEE. http://doi.org/10.1109/ICITST.2016.7856718.

1.10 Thesis Outline

The remainder of this thesis is organised as follows:

Chapter 2

This chapter presents a general background for Wireless Sensor Network (WSN) and lists the most popular platforms and Embedded Operating Systems (EOS) used in the WSN domain. It then outlines how the complexity problem can be solved using Software Engineering and explains the main approaches used to resolve this issue. Next, it discusses the main strategies and approaches used to define application development solutions in the WSN domain, before summarising the considered solutions in terms of a set of feature criteria. Finally, it outlines the final design requirement, which is set up as the main design requirement for our proposed solution.

Chapter 3

This chapter outlines the proposed SenNet meta-model that builds on the basis of the ISO/IEC29182: Sensor Network Reference Architecture (SNRA). This meta-model links the application with the network domains. The main application types supported by this meta-model are then shown, before finally describing how this meta-model is implemented.

Chapter 4

This chapter discusses how SenNet works (internal view), by considering first the Text-to-Model (T2M) transformation rules, which include the lexical and syntactical analysis stage and the semantic analysis stage. This is followed by explaining the Model-to-Text (M2T) transformation rules and process, which constitute the source code generation process. Finally, it discusses how SenNet can be updated and expanded.

Chapter 5

This chapter presents our proposed application development (SenNet) from a user's (beginner developer) point of view. SenNet's main components are illustrated: CPC and CGC components; then SenNet editor, general application syntax, and the main tasks offered are discussed.

Chapter 6

This chapter outlines first the evaluation plan adopted to evaluate SenNet and prove its usability and functional suitability. Many methods have been included in this assessment plan, which are discussed in detail later in this chapter, such as user study and the source code complexity analysis that is measured by calculating the LOC and HCM metrics to evaluate SenNet source code using four different scenarios. Finally, we summarise all the results according to a set of defined metrics at the beginning of this chapter, which shows that SenNet is usable and suitable to develop WSN applications.

Chapter 7

This chapter concludes this thesis; it discusses the research outcome, the final conclusions and finally outlines the possible future work.

Chapter 2: Background and Overview

2.1 Introduction

This chapter gives a general WSN background including the most popular WSN platform and TinyOS as one of the successful embedded operating systems in the WSN domain. This is followed by discussing how the complexity problems could be solved using Software Engineering principles, giving a brief description of the main approaches used to address this problem (MDD and DSL). Finally, we have outlined the general strategies used by the research community to define a development abstraction.

2.2 Wireless Sensor Network

The beginnings of WSNs date back to the 1980s, when the first trend in WSNs was the Distributed Sensor Network (DSN) project prepared by the Defence Advanced Research Projects Agency (DARPA) (Wang & Balasingham, 2010). Later, with the engineering revolution in semiconductors and communications, the new WSN began in a couple of projects targeting environment monitoring, such as the WSN deployment at Great Duck Island in 2002 (Mainwaring, Culler, Polastre, Szewczyk & Anderson, 2002).

Since 2004, reported evidence for WSNs has increased significantly; many researchers such as (Oppermann, Boano, et al., 2014) arguing that, because of the commercialization of the first WSN sensor node (mote) Mica2, it has become a *de facto* standard for WSN research purposes, alongside the maturing of WSN system infrastructures such as TinyOS (Levis et al., 2005).

Currently, WSN have a broad range of applications, such as volcano monitoring (Werner-Allen et al., 2006), flood detection (Hughes et al., 2008), hazardous chemical detection (Bulusu, Heidemann & Estrin, 2000), and the Lofar Agro project (Baggio, 2005) for agricultural purposes; it is used for human body health in these projects (Hu, Jiang, Celentano & Xiao, 2008; Malan, Fulford-Jones, Welsh & Moulton, 2004; Kirbaş & Bayilmiş, 2012; Yuce, 2010; Chung, Lee & Jung, 2008) and in WiSA (Frey, 2008) for monitoring and controlling production cycles. According to ISO/IEC29182 SNRA standard (ISO/IEC, 2014b), WSN applications have been categorised as:

- Logistics and supply chain management
- Automation, monitoring, and control of industrial production processes
- Health care and medical applications at home and in hospitals
- Critical infrastructure protection and public safety
- Automation and control of commercial buildings and smart homes
- Automation and control of agricultural processes
- Intelligent transportation and traffic
- Environmental monitoring, forecasting, and protection
- Facility management
- Asset management
- Defence and military applications
- Homeland security

2.2.1 Most Popular WSN Hardware

A sensor node performs the required functionality by integrating several components: a low-power microcontroller unit (MCU) including RAM (for data saving) and ROM (for code saving) small memory chip, Analog-to-Digital Convertor (ADC) and a radio transceiver for data communication. Different sensors can be attached to the sensor node using an expansion connector. The most used power sources for this type of devices or nodes are batteries or solar sources. The main MCUs used in this technology are ATmega128 (Atmel, 2006) and MSP430 (Texas Instruments, 2008), manufactured by Atmel and Texas Instruments respectively. These two MCUs became the source for producing many WSN platforms, and this is due to several factors: Ultra-Low Power (ULP) energy consumption, scientific and research community support, open source compilers based on the GNU Compiler Compiler (GCC) compiler type, and the support of the TinyOS operating system (Kouche, 2013).

IRIS[3], Mica2[4], and MicaZ[5] are WSN platforms that use an 8-bit ATmega MCU from Atmel, with a slight difference for each one, such as in the internal memory or the frequency speed of the radio transceiver. TelosB (Polastre, Szewczyk, & Culler, 2005) is another WSN platform

[3] http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf
[4] https://www.eol.ucar.edu/isf/facilities/isa/internal/CrossBow/DataSheets/mica2.pdf
[5] http://www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf

that uses 16 bit MSP430 MCU from Texas Instruments. This platform is one of the more popular platforms in the research community.

SunSPOT (Simon, Cifuentes, Cleal, Daniels, & White, 2006) and IMote2 (Nachman, Huang, Shahabdeen, Adler, & Kling, 2008) are popular platforms too. SunSpot was developed by Sun Microsystems and uses a Squawk Java Virtual Machine (JVM) that runs directly on the ATMEL920T MCU. IMote2 is a WSN platform developed by Intel using an XScale processor PXA271. This platform is different from the usual WSN platforms that focus on low power and resources, it focusses on high computation activities, such as audio and video processing. This platform is one of the successful examples that ported to work with TinyOS.

Table 2-1: Sample of the Most Used WSN Platforms 2002-2012 (Thang, 2015)

Platform	Year	MCU	OS & Programming
Mica2	2002	ATMega 128L 8bit	TinyOS, nesC, SOS, MantisOS
BTnode	2003	ATMega 128L 8bit	TinyOS, nesC, C, NutOS
MicaZ	2004	ATMega 128L 8bit	TinyOS, nesC
TelosB	2005	T1 MSP430 16 bit	C, nesC, Contiki, TinyOS, SOS, MantisOS
Eyes	2006	T1 MSP430 16 bit	C, PEEROS
IMote2	2007	Intel PXA271 16 bit	C, nesC, Linux, TinyOS
SHIMMER	2008	MSP430 16 bit	TinyOS, nesC
Zolertia Z1	2011	MSP 430 16 bit	C, nesC, TinyOS, Contiki
WaspMote	2012	ATMega1281 8 bit	C, WASPMote ISE, WASP firmware

2.2.2 TinyOS

Many embedded operating systems have developed to support embedded systems generally and WSN particularly. TinyOS (Levis et al., 2005) is one of the open-source operating systems designed particularly for WSN, which are built from a set of reusable components. It supports an event-driven concurrency model due to the use of split-phase technique (this technique is explained in section **3.4.1**) and a new scheduler design. Moreover, many other facilities such as Hardware Abstraction Architecture (HAA) (this architecture is described in more details in section **4.4.1**) lead to TinyOS supporting multiple microcontroller families. Levis & Gay (2009), summarised the main TinyOS communication types as shown in Table 2-2.

Table 2-2: Communication Types Supported by TinyOS

Communication type	**Single/Multi-hop**
Serial	Over serial
Active message (AM)	Single-Hop
Collection Tree Protocol (CTP)	Multi-Hop
Dissemination	Multi-Hop

TinyOS is considered the *de facto* operating system for WSN (Akyildiz & Vuran, 2010; Ouadjaout et al., 2016). Delamo et al. (2015) have surveyed IEEE Xplore, ACM Digital Library and Google Scholar systematically, finding that TinyOS returns 85% of the references concerning WSN, and argued the high popularity of using TinyOS was because it was maintained by relevant research groups and companies. Accordingly, most of the proposed new abstractions developed to reduce complexity in WSN application development used TinyOS and nesC as a foundation to build their abstracted solutions (Essaadi et al., 2017).

TinyOS is implemented in the nesC (Gay et al., 2003) language that supports the TinyOS components and concurrency model. The nesC language manipulates TinyOS components in a way similar to objects that encapsulate the state and couple the state with functionality. nesC is an extension version of C language, but it differs from C language in how components are linked to each other (Mozumdar, Gregoretti, Lavagno, Vanzago, & Olivieri, 2008). This language is considered challenging and complex (Luo, Abdelzaher, He, & Stankovic, 2006), since the developer should program more than one program file to develop an application (minimum three files) (Ravichandran et al., 2016), and the difficulty appears when a developer is trying to link new codes with existing ones (Levis, 2006).

2.3 How Can Complexity be Solved?

In the context of software systems, there is an opinion that *"Software entities are more complex for their size than perhaps any other human construct"* (Brooks, 1987, p.11). At this stage, the role of software engineering is to reduce complexity, which can be achieved through developing different tools, processes and methodologies.

Generally, in software engineering there are two directions to reduce complexity for a particular domain application, the first direction is to raise the level of abstraction, while the second direction is to increase the level of concreteness. Raising the level of concreteness will solve the problem partially, as it will address the complexity problem towards a particular type of applications, but not all domain applications. Therefore, raising the level of abstraction can represent the right solution to solve or reduce complexity in software projects (Kunii & Hisada, 2000).

Abstraction can be defined as *"A cognitive process of the human brain that enables us to focus on the core aspects of a subject, ignoring the unnecessary details."* (Ghosh, 2011a, p. 10).

Abstraction is a powerful and traditional way to shield the users of software systems against complexity, such as when an operating system hides the complexity of the underlying hardware (Jörges, 2013). Moreover, it allows developers and programmers to express their problems in a simple form by hiding complex and irrelevant implementation details (Áemília, Ľubomír, Sergej, & Ján, 2011; Damaševičius, 2006; Kollár, Pietriková, & Chodarev, 2012).

2.3.1 Model-Driven Development (MDD)

In day-to-day work, users and developers face a huge and constantly changing variety of platforms, tools, platforms and languages that need to be mastered (Jörges, 2013). As an additional level of abstraction, models allow the specification of a software system apart from the actual concrete implementation.

Models, modelling and model transformation have become the basis of the MDD approach (Geihs, Reichle, Wagner, & Khan, 2009; S.J. Mellor, Clark, & Futagami, 2003; Skubch, Wagner, Reichle, & Geihs, 2011; Weise, Zapf, Khan, & Geihs, 2009), where a developer should develop a model for the required developed system, then transform this model into a real system (executable software entity). This is applicable due to the revolution of generating code; code generation can fill the gap between the system abstraction (model) and the concrete implementation (Jörges, 2013). This is considered a specific form of model-to-text (Czarnecki & Helsen, 2006; Oldevik, Neple, Grønmo, Aagedal, & Berre, 2005) or model-to-source transformation (Hemel, Kats, Groenewegen, & Visser, 2010; Selic, 2003). The code generator is one of the specifications of Model-Driven Development (MDD) (Bézivin, 2005; Seidewitz, 2002). Currently, models are not only used for documentation and visualisation purposes (Jörges, 2013), but are also promoted as development artefacts.

Therefore, MDD can be used to develop special languages that focus on specific domains using one of these approaches:

- Using Model-Driven Architecture (MDA) approach.
- Using model transformation and code generators to develop a language by having (Marques, Balegas, Barroca, Barisic, & Amaral, 2012): (1) The language syntax in the form of meta-models; (2) The language semantics represented by model-to-text or source transformation.

2.3.2 Domain-Specific Language (DSL)

A domain-specific language (DSL) is a language tailored to a specific domain (Mernik, Heering, & Sloane, 2005). Using DSL is considered one of the most frequent techniques used to raise the level of abstraction (Mernik et al., 2005; Sun, Demirezen, Mernik, Gray, & Bryant, 2008). Mernik et al. (2005, p. 321) argued that DSL was *"enabling of software development by users with less domain and programming expertise, or even by end-users with some domain, but virtually no programming expertise"*. In addition, it is bridging the gap between business users and developers by encouraging collaboration through using shared vocabulary, because the main reason for software projects failing is the lack of communication between users (who know the problem domain) and developers (who develop the software system) (Ghosh, 2011b). Fowler (2009) classified DSL's according to the implementation approach:

- **Internal DSL** (also known as embedded DSL) is a DSL implemented as an extension to an existing GPL, such as Ruby (Carlson & Richardson, n.d.) and Haskell (Hudak et al., 1992). This type of DSL can access all host language constructs and infrastructure such as tools and libraries. Therefore, it is not necessary to build a new compiler or interpreter.
- **External DSL** is a DSL represented in a separate language to the main programming language it is working with. The advantage of external DSLs is that the developer of the DSL may define any possible syntax independently from the general syntax of the host language.

Using DSL offers many advantages, such as easier collaboration with business users, easier in maintaining and updating the DSL code, users focus on a tiny surface area of the application code, and it increases users' and developers' productivity (Ghosh, 2011b).

This type of language can be represented to the end user in the form of a textual notation, or using a model or graphical notations; this is called Domain-Specific Modelling Language (DSML). Many researchers prefer to develop DSLs with a textual notation because it would require more effort to build usable editors for the graphical or visual languages (Volter, 2011).

2.4 Approaches to Defining WSN Development Abstraction - Related Work

WSN application development and deployment are typically performed very close to the hardware level, and this leads to developers needing to focus on low-level system details (Mottola & Picco, 2011). Researchers have proposed different development abstractions to overcome WSN application development difficulties, with the ultimate goal of making application development easy (Sugihara & Gupta, 2008). These various development techniques facilitate the applications' development process, as some of them deal with how to access the sensed data in the sensor node, and another group are concerned with how to give orders and pass these along to the sensor nodes. Many surveys in the literature have focused on these techniques, such as Chandra & Dwivedi (2015), Essaadi et al. (2017), Farias et al. (2016), Laukkarinen, Suhonen, & Hännikäinen (2012), Malavolta & Muccini (2014), Mohamed & Al-Jaroodi (2011a), and Mottola & Picco (2011b). Figure 2-1 illustrates the current approaches used by the research community to define a development abstraction that concludes with setting a customised development and programming language that would help developers and users to develop WSN applications.

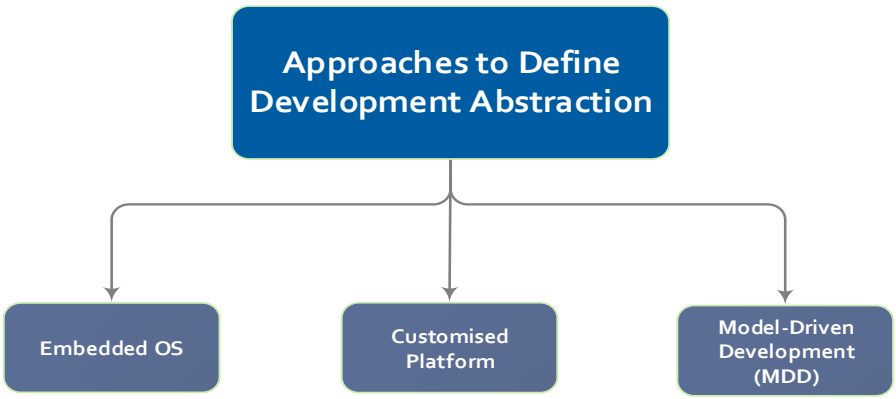

Figure 2-1: WSN Development Applications Approaches

2.4.1 Embedded Operating Systems

There is a special type of customised operating systems that supplies a set of services that can be utilised by the developer to develop their node-level applications. There are many examples of these operating systems, such as **TinyOS** (Hill et al., 2000), which is considered an event-driven operating system, which offers **nesC** programming language as one of the dominant

programming languages in the WSN domain based on C language, while **Mantis** (Bhatti et al., 2005), **TinyOS 2.x** (The TinyOS 2.x Working Group, 2005), **Nano-RK** (Eswaran, Rowe & Rajkumar, 2005) and **RETOS** (Cha et al., 2007) are considered thread-driven operating systems. Furthermore, some operating systems are designed to support both of these strategies, such as **Contiki** (Dunkels, Grönvall & Voigt, 2004) and **LiteOS** (Cao, Abdelzaher, Stankovic & He, 2008). The generated application using this type of abstraction is a node-specific application (Laukkarinen et al., 2012).

2.4.2 Customised Platform

Programming the sensor nodes is more complicated than programming conventional computer systems due to their constrained resources, high dynamics and inaccessible deployment environments (Leelavathi, Shaila, Venugopal, & Patnaik, 2013). Some researchers develop a customised software architecture, hardware, or both of them to enable users to use these specialised designs to develop their required applications. To allow users to use these customised architectures, researchers define a set of tailored instructions (DSL) that enable users to interact with these architectures to develop their applications.

The **galsC** (Cheong & Liu, 2005) is a node-level programming language developed for event-driven applications, which is a hybrid language that uses synchronous event-driven model and an asynchronous message passing model using TinyGALS middleware (Cheong, Liebman, Liu, & Zhao, 2003). This middleware and the defined language is based on the TinyOS operating system and nesC language. Using this language is considered very complex since it takes the sensor node as basic computation, communication, and actuation unit (Luo et al., 2006).

BASIC (Miller, Dinda & Dick, 2009) is a node-level imperative loop oriented programming language that was developed for simple WSN applications targeting domain experts and novice programmers. BASIC is implemented using uBASIC[6] interpreter developed using C language for embedded devices. One of BASIC's limitations is that it suffers a large interpreter overhead and the application source code should include the sleep command because it is not put the system in low-power by default (Elsts, Judvaitis, & Selavo, 2013).

[6] http://dunkels.com/adam/ubasic/

In 2009 Bai, Dick, & Dinda surveyed the existing WSN applications and identified eight characteristics for WSN applications: mobility, sampling, transmission, actuation, interactivity, data interpretation, data aggregation, and heterogeneity. Accordingly, they designed a fixed seven archetypes template as illustrated in Table 2-3, and presented a simple language called **WASP** that is targeted to one of these seven archetype templates. These seven templates and the simple WASP commands are designed for domain experts. The user has to start by choosing the most appropriate template according to the required application nature, after the user has started configuring the choosing template using WASP language. WASP compiler is written in Python and translates the language into the nesC program for TinyOS. The authors of WASP language have performed a user study to evaluate their language using 28 novice programmers. This language has many limitations, for example, to develop an application, the developer would be limited to the available functionalities offered by the selected archetype template. In addition, to program a network, WASP offers the developer a facility to define a job that will be assigned to all sensor nodes in the network.

Table 2-3: The Seven Archetype identified by Bai, Dick, & Dinda (2009)

Archetype	Size	Mobility	Sampling	Data transmission	Actuation	Interactive	Data interpretation	Data aggregation	Homo-geneous
1	7	stationary	periodic	periodic	N	N	*	*	Y
2	6	stationary	*	event-driven	N	*	Y	*	Y
3	4	mobile	periodic	*	*	N	*	*	Y
4	3	mobile	periodic	*	*	Y	*	N	N
5	1	stationary	hybrid	hybrid	N	Y	Y	Y	Y
6	1	stationary	event-driven	hybrid	Y	Y	N	Y	Y
7	1	mobile	periodic	event-driven	N	N	Y	Y	N

Dinam (Gordon, Beigl, & Neumann, 2010) is a platform solution that consists of two parts, uPart (Beigl et al., 2006) and D-Bridge (Gordon & Beigl, 2009). uPart is a special sensor node that is built based on the Arduino platform. It uses a Java-based web server already installed on top of the sensor node that can be accessed via any PC and starts programming each node using the Over the air programming (OTAP) concept, which is called D-Bridge (Gordon & Beigl, 2009). This uses Basic language (Kurtz, 1978) as the main programming primitives offered to developers. Dinam offers easy jobs for gathering sensed data, focused on configuration more than programming. Dinam was evaluated using a user study with five students with limited knowledge and programming skills. This solution lacks an integrated development environment (Gordon et al., 2010). Besides, installing the development environment inside sensor node will waste sensor node resources. The Dinam is limited to a particular sensor node hardware, and the reconfiguring process has to be done for each node

separately. Finally, the configuration interface does not allow the programmer to fine-tune low-level parameters for the sensor node (Kovatsch, Mayer, & Ostermaier, 2012).

PROVIZ (Chandrasekar, Uluagac, & Beyah, 2013; Ravichandran et al., 2016) is a suggested framework for graphical programming WSN applications, besides WSN network analysis (Chandra & Dwivedi, 2015). This framework uses a special service-oriented middleware called M-core (Valero, Uluagac, Venkatachalam, Ramalingam, & Beyah, 2012) installed on top of the TinyOS, which offers M-Core Control Language (MCL). MCL is a security domain-specific language that was developed using Python, where the final graphical presentation of the WSN will be translated into nesC program code using MCL. The network visualisation interface was implemented using QI GUI C++[7]. This framework focused on the network analysis and simulation rather than application development, as regarding application tasks and activities, this framework is offering limited capabilities.

SEAL (Elsts, Bijarbooneh, Jacobsson, & Sagonas, 2015; Elsts et al., 2013) is a node-level high abstract DSL that was proposed to be part of a new environment which included MantsOS middleware and ProFuNTG tools implemented for novice programmers and domain experts to help them develop WSN applications without help from WSN experts. The SEAL language includes two executable statements: read and use; three types of components can also be defined (sensors, actuators, and output). Python is used to write the SEAL language compiler, which generates the required code in C language that will be run by MantsOS middleware. The SEAL editor used Google Blockly to offer visual programming.

CrimeSPOT (De Roover, Scholliers, Amerijckx, D'Hondt, & De Meuter, 2013) is a declarative rule-based DSL for programming WSN applications on top of LooCI (Hughes et al., 2012), event-driven and component-based middleware that works with SunSPOT sensor nodes. CrimeSPOT enables the programming both of node and network level facilities. It also provides temporal facilities for working with data.

makeSense Framework (Daniel et al., 2013; Oppermann, Romer, Mottola, Picco, & Gaglione, 2014) is a unified programming framework and a compilation chain, offering high-level abstraction development that generates code ready for deployment on WSN nodes. This framework was built using Java targeting domain experts who have knowledge in Business Process Modelling Language (BPML). The authors define three layers: the first layer is a visual

[7] https://www.qt.io/developers/

modelling language based on BPML; the second layer is a Java-based macro-programming language; the final layer is a platform-specific (Contiki) executable code. This framework has many limitations, including the code generated consuming node memory, the network or application behaviour not being taken into account, and the current version can only be applied to a limited number of node types and simple application tasks.

Virtual Machine (**VM**) is a commonly used technique for platform independence and isolation. Some researchers used VM to develop new customised languages to program WSN platforms, but the most significant role of the virtual machine is in WSN re-programmability, which is the capability to inject new code dynamically into each node on site (Sugihara & Gupta, 2008). There are two approaches in this abstraction method, both using Java-based virtual machines to facilitate the development process for the developers; the first edition of this concept uses a virtual machine that is installed on top of the operating system, such as Mate (Levis & Culler, 2002). **Mate** is a virtual machine that runs on top of the TinyOS operating system, which provides a simple programming interface to sensor nodes. **Impala** (Liu & Martonosi, 2003) is another example using the same principle that focuses on re-programming, that was designed for ZebraNet project at Princeton University. This solution did not support large updates or complete software change and it was suitable for rich resources sensor nodes (Wang, Cao, Li, & Dasi, 2008). Alternatively, there is specialised hardware that runs a Java-based virtual machine such as a **SunSPOT** (Simon et al., 2006) platform, originated by Sun Microsystems, which uses Squawk virtual machine (Akyildiz & Vuran, 2010; Smith, 2007). SunSPOT has not gained high acceptance (Elsts et al., 2013), because it is heavyweight, and needs professional programming skills. In general, using a VM approach to building an abstraction development has many advantages, but also it has many disadvantages, for example the execution of instructions can introduce high overhead on nodes, and the application programming is not easy, which leads to poor usability (Delicato et al., 2014).

2.4.3 Model-Driven Development

While considering the WSN application development process as a complex task, many researchers try to ease this by suggesting and implementing particular abstraction layers and software processes that are capable of making the application development process easier (Taherkordi, Eliassen & Johnsen, 2013). According to Hailpern & Tarr (2006), model-driven development (MDD) is a software engineering approach consisting of model technologies to raise the level of abstraction. In the MDD process, developers describe applications with an

abstract model that is subsequently refined into concrete models. In the end, the concrete model is used to generate executable code (Shimizu, Tei, Fukazawa & Honiden, 2012). Most of the MDD developed for the WSN domain is divided into three layers, the first dealing with activities related to network scope, followed by the second layer that is used to configure a group or sub-network with certain operations and activities, finally the last layer is related to the sensor node scope of operations, such as the work presented by Dantas et al. (2013), Shimizu, Tei, Fukazawa, & Shinichi (2011), and Tei et al. (2015).

Many researchers use the MDD approach to tackle the complexity in developing WSN through using models and automatic transformation to generate code or for WSN analysis in terms of their requirements (Essaadi et al., 2017). One of the MDD initiatives is Model-Driven Architecture (MDA) defined by Object Management Group (OMG), which is based on three models: computation- independent model (CIM), platform-independent model (PIM) and platform-specific model (PSM). MDA transforms a CIM to a PIM and a PIM to a multiple PSMs using model transformations. Therefore, to enable users and developers to customise these models, a specific modelling language should be developed to help them manipulate each model and tailor it according to their application requirements.

The first example of an MDA initiative is the work presented by **Doddapaneni** et al. (2012); Doddapaneni and his team proposed a modelling framework that uses three different levels of modelling languages. The first layer is Software Architecture Modelling Language for WSN (SAML), which allows developers to define the software architecture of the WSN application. This layer is set structurally but not functionally. The second layer is the Node Modelling Language (NODEML) used for describing the low-level details of each type of node that can be used within a WSN, such as the implemented MAC protocols and routing protocols. The current version of this layer defined T-MAC as a MAC protocol and CC2420 radio by the Texas instruments. The final language is the Environment Modelling Language (ENVML), which allows designers to specify the physical environment in which the WSN nodes are deployed. By using these three modelling languages, a code generator (Acceleo) generates a software code that is designed to be run in a Castalia simulation to simulate the final application and extract the required analysis information. This framework is focusing on the network simulation and analysis using UML and Simulink for representing the network activities, rather than code generation, as they propose code generation as possible future work. The generated code for simulation cannot guarantee that design errors can be discovered. Moreover, they do

not define or present a particular WSN platform that could be appropriate to work with (Intana, 2015; Kumar & Simonsen, 2014; Malavolta & Muccini, 2014; Rodrigues, Batista, Delicato, Pires, & Zomaya, 2013), and the SAML language functionality has not been developed.

Architecture for **Wi**reless **Se**nsor and Actuator **N**etworks (**ArchWiSeN**) (Rodrigues, Batista, Delicato, & Pires, 2015; Rodrigues et al., 2013; Rodrigues, Delicato, Batista, Pires, & Pirmez, 2015) is another MDA instance that was developed in the Federal University of Rio de Janeiro. ArchWiSeN is an architecture framework for Wireless Sensor and Actuator Network (WSAN), used to develop WSN applications by domain and network experts. The authors have defined and developed two modelling layers, PIM and PSM, arguing that CIM layer can be done using any existing requirements analysis technique to collect and analyse the application requirements. Therefore, within the context of ArchWiSeN, they defined PIM to enable application developers to model the application logic, while PSM is defined to model the network configuration with the aid of a TOSSIM simulator. The ArchWiSeN authors assume that to develop WSN applications, the developer's team should include domain experts and WSN experts. Accordingly, the PIM layer will be modelled by the domain experts, who will model the application by assigning a specific task for each node in the network, while a WSN expert will model the PSM layer. All PIM and PSM modelling profiles are developed using UML and Eclipse's Papyrus, and to control the Model-to-Model (M2M) transformation between PIM and PSM layers they used Atlas Transformation Language (ATL). The final transformation process within ArchWiSeN is Model-to-Text (M2T) that translates the final PSM model into an executable code in nesC format, as the current version is targeting TinyOS platform. This process is done using Accelelo code generator and BNF transformation rules. To assess the framework's usability, they conducted a user study using 10 participants (Master Students of the Computer Science Course in the same university), and also developed some real business scenarios. The authors had not presented and shown that the generated code had been tested using real WSN sensor nodes, and all resulting code was assessed and analysed logically.

Domain **L**anguage for **Wi**reless **S**ensor and Actuators Networks **S**ystems (**LWiSSy**) (Dantas et al., 2013) is the second project in the Federal University of Rio de Janeiro to use an MDA approach to define a new abstraction development for a Wireless Sensor and Actuation Network (WSAN), targeting domain experts. LWiSSy has taken the same MDA infrastructure that proposed in ArchWiSeN with some modifications. LWiSSy includes three different

graphic models: the first modelling language deals with the WSAN structure functionalities that should be used by domain experts, and it includes network and node groups programming. The second model deals with the network behaviour, and it should be set by the domain experts, where it deals with the programming of node groups and single nodes, and it is responsible for designing the WSAN application behaviour. The third modelling language is targeting the network optimisation that should be set by network experts. After complete modelling of the three modelling layers, the application source code is generated using the same transformation rules employed in the ArchWiSeN. LWiSSy was evaluated using a simple case study, as well as a user study that included some undergraduate students in the Computer Science department of the same university.

To expand the ArchWiSeN framework, the Federal University of Rio de Janeiro research group developed **COMFIT** (De-Farias, Brito, et al., 2016) which is a cloud-based development environment focused on building WSN applications for IoT, using client–server and web-based architecture. ArchWiSeN is installed in a web server, that can be accessed remotely, to do all the modelling and analysis tasks. In COMFIT, the ArchWiSeN framework is updated in terms of supported simulations to include TOSSIM and Cooja to support IoT applications, and the PSM was also updated to include the Contiki platform beside the TinyOS. As possible future work, the authors intended to design new network-level tasks and activities using macro-programming.

An MDA approach was used by a research group at Waseda University (Shimizu et al., 2012; Shimizu, Tei, Fukazawa, & Honiden, 2014; Shimizu et al., 2011) to define a development abstraction for WSN application. The authors defined three PIMs architecture modelling languages (dataflow, group, and node-level) in a textual form. The user should define the first PIM layer, which in turn will generate the basic textual form of the other two PIMs. Consequently, the user should configure and customise the two generated PIMs using their defined customised languages. PIM to PIM translation is done according to a specific M2M transformation rules developed for this purpose. The authors have defined and customised a PSM layer that includes a set of a predefined TinyOS application templates updated by the Model-to-Text (M2T) transformation rules. This approach has many limitations, for example the proposed DSMLs can only describe a very simple sensing application with simple data processing activities, but not complex ones, such as adaptive behaviour. Moreover, the authors did not consider the participation of developers with different expertise areas and excluded

several important features of lower abstraction level specification, such as the communication protocol, the network topology, and the device types (Antonopoulos et al., 2016; Dantas et al., 2013; Rodrigues, Delicato, et al., 2015; Tunc et al., 2016). Furthermore, three different modelling languages are used to create applications, which presents a steep learning curve for new users (Shimizu et al., 2014).

After considering the previous work done by Shimizu et al. (2014), and to present a new solution, the same research team in Waseda University then proposed a **stepwise software development process for average developers** (Tei et al., 2015) targeting average developers to ease the development process of a simple monitoring WSN applications. They identify two types of concerns: network-related concerns and data-processing concerns, and providing support to reuse network related solutions designed by sensor network experts. The authors defined two modelling languages to be used by average developers and one modelling language to be used by network experts to include and embed the best practice solutions within the framework. So, to develop a simple monitoring application, an average developer should first use the one concerned with data processing activities that is a network independent language, then the resulting model will be translated internally into a group of nodes model that is network dependent, and presented again to the average developer to manipulate and configure the network related parameters such as the routing protocol. Finally, the resulting final model will be transformed using M2T transformation to generate the required code for this application, where their code generator (Acceleo) outputs programs written in nesC for TinyOS according to the best practice solutions embedded previously by the network experts. They evaluate their framework by implementing some case studies and a user study with four average developers. This proposed solution is designed according to the concept of the previous proposal with some modification to decrease the number of modelling languages that average developers are dealing with. However, according to their paper results, this new solution supports only simple monitoring applications, which are the most common and simplest type of WSN application. The data processing modelling language cannot express the application logic of event-driven WSN applications. The framework is only supporting the software development, but not real WSN developments. Moreover, their framework has a consistency problem between its models. Finally, this framework has a learning cost problem, because it uses three modelling languages.

Moppet Framework (Boonma & Suzuki, 2010; Pruet Boonma, Somchit, & Natwichai, 2013; Pruet Boonma & Suzuki, 2011) is one of the solutions defined according to Model-Driven Development (MDD) for rapid WSN application developments. Moppet defined two layers of modelling; the first modelling layer is Moppet-Feature Modelling (FM) that concerns the general application tasks and activities, which configured by the developer. The second layer is Moppet-Performance Estimation (PE), which uses event calculus and network calculus to estimate a WSN application's performance. Moppet deals with WSN management using a graphical and configuration interface, which helps developers to change the configuration of the application easily, then the framework generates the required nesC code that can be deployed with TinyOS. Moppet builds on top of the TinyDDS (Boonma & Suzuki, 2010b) middleware, which works with TinyOS operating system. Moppet focuses on estimating the network performance using calculus network calculations and not a simulation.

2.4.4 Principal Findings

Through the last decade, developing WSN applications has been a hot topic. Therefore, the research community has focused on presenting new methods employing the principles of Software Engineering. Many solutions have been presented, and Essaadi et al. (2017) have made a systematic mapping study (SMS) to survey the existing MDD-based languages in WSN domain, examining 1852 papers. Malavolta & Muccini (2014) also reviewed around 780 academic papers, to produce their final survey results. Therefore, we concentrate on discussing the latest, most cited, and inspired solutions that have a significant impact on this topic, as well as showing the limitations they may include.

Most of the researchers have used one of three strategies to define a new development method or a programming language; Table 2-4 summarises these strategies and their limitations. The first strategy is building a new embedded operating system that can help in abstracting the complexity of the hardware-level in developing node-level applications. However, the operating system needs continuous support. Moreover, the more WSN platforms are supported by this new OS, the more effective and useful it will become, which constitutes a big challenge.

The second strategy is customising a platform to construct a new architecture accompanied with a simple and abstracted set of commands that can be used smoothly and easily. However, this type of strategy is restricted to a specific type of platforms (Hardware or Software), and the customisation process is done according to a set of predefined application requirements, so

changing or updating these platforms due to a change in these requirements may cost more than building new platforms.

The final strategy, which is the MDD approach, is considered the best strategy because it raises the level of abstraction. This enables developers to develop less error-prone solutions, as it allows analysis in the early stages. Finally, the MDD approach introduces the concept of model transformation, which offers the code generation in simple and abstracted level and is much easier than building code generator in conventional programming languages. However, there is a trade-off between the number of modelling language layers and the user usability and learning.

Table 2-4: The Drawbacks of the Available Strategies to Define Development Abstraction

Solution Strategy	Limitations
Embedded OS	• Embedded OS used to develop node-level application • To develop a new Operating System, we have to consider the following points: 1. Technical Support 2. Hardware compatibility
Customised Platform	This solution strategy is restricted to a limited number of WSN platforms or application type as they customised according to a predefined set of application requirements. In some cases, changing or updating these requirements is more costly than developing new ones.
MDD	Using many modelling languages increases the complexity in language learning.

For deep analysis of the discussed solutions, we have adopted a set of feature criteria suggested by Essaadi et al. (2017) and Malavolta & Muccini (2014) in their surveys and as shown in Table 2-5. The selected criteria are listed below:

- **Strategy**: What is the strategy used to define the selected solution?
- **Language Type**: What is the type of language produced?
- **Modelling Scope**: What is the modelling capacity of the selected language? Possible values are node level (N) or group of nodes level (G) or network level (Net).
- **Goal**: What are the main purposes of the selected language? Possible values are code generation (CG) or analysis (AN).
- **Data Processing**: Is the selected language able to model the data processing? Possible values are Data Aggregation (DA), Data Fusion (DF) or both.
- **Concrete Syntax**: Does the language model WSNs graphically (Visual) or textually (TEXT) or using both of them (MIX)?

- **Target Platform**: If the goal of the selected language is code generation, on which target platform is the generated code supported?
- **Evaluation Method**: What is the method used to evaluate the language?
- **Developer Types**: For which developer types are the languages designed? Possible values are domain experts (including average developers and novice developers) and network experts.

Table 2-5: Application Development Approaches Summary

Solution	Strategy Used	Language Type	Modelling Scope	Goals	Data Processing	Concrete Syntax	Target Platform	Evaluation Method	Developer Types	Comments
galsC	Customised Platform	Embedded DSL	N	CG	DA	Text	TinyOS	Case Study	network experts	No Event-triggered Events
SensorBASIC	Customised Platform	Embedded DSL	N	CG	No	Text	MicaZ	User Study	Domain experts	• Large interpreter overhead • Sleep command problem
Dinam	Customised Platform	Embedded DSL	N	CG	No	Text	uPart	User Study	Domain experts	Waste sensor node resources
PROVIZ	Customised Platform	Embedded DSL	N	AN	No	Visual	M-Core (TinyOS)	Case Study	Domain experts	Limited Capabilities
WASP	Customised Platform	Embedded DSL (Python)	N, G, Net	CG	DA	Text	TinyOS	User Study	Domain experts	The language is limited to 7 architecture templates.
SEAL	Customised Platform	Embedded DSL	N	CG	DA	Visual	MantsOS	Case Study User Study	Domain experts	Limited Capabilities
CrimeSPOT	Customised Platform	Embedded DSL	N, Net	CG	DA	Text	LooCI	Case Study	network experts	Programming Complexity
makeSense	Customised Platform	Embedded DSL	N	CG	DA	Text	Contiki	Case Study	Domain experts	• Limited number of node types • Generated code consuming node memory • Simple application tasks • Network or application behaviour not taken into account
Doddapaneni et al. (2012)[8]	MDD (3 Layers Modelling)	External DSML	N	AN	No	Mix	N/A	Case Study	network experts	• SAML language not defined • No defined Platform
ArchWiSeN	MDD (2 Layers Modelling)	Embedded DSL (UML)	G	AN, CG	DA	Visual	TinyOS	Case Study User Study	domain and network experts	Generated code not tested using a real WSN sensor nodes
LWiSSy	MDD (3 Layers Modelling)	Embedded DSL	N, G, Net	AN, CG	DA	Visual	TinyOS	Case Study User Study	domain and network experts	generated code not tested using a real WSN sensor nodes
COMFIT	MDD (2 Layers Modelling)	Embedded DSL (UML)	G	AN	DA	Visual	TinyOS, Contiki	Case Study	domain and network experts	No Code Generation
Shimizu et al. (2014)	MDD (3 Layers Modelling)	External DSML	N, G, Net	AN, CG	DA	Text	TinyOS	Case Study	domain and network experts	• Supporting the software development, but not real WSN developments • No event-triggered jobs • High learning cost
Tei et al. (2015)	MDD (3 Layers Modelling)	External DSML	N, G, Net	CG	DA, DF	Mix	TinyOS	Case Study User Study	domain and network experts	• No event-triggered jobs • High learning cost • Consistency problem
Moppet	MDD (2 Modelling Layers)	Embedded DSML	N	CG	DA	Text	TinyDDS (TinyOS)	Case Study Simulation	Domain experts	Limited Capabilities

[8] By convention for unnamed languages, the authors are referenced instead

According to the results in Table 2-5, regarding **modelling scope**, 66% (10 out of 15) of the proposed languages or solutions support node or group-level modelling, which seems a rational percentage considering Essaadi et al.'s (2017) results, that found 67% of the proposed solutions were defined to produce node-level applications. There are two examples of using customised platforms that offered solutions supporting network-level modelling (WASP and CrimeSPOT); these two examples are restricted and limited solutions, as WASP is limited to seven archetype models or templates, and CrimeSPOT is restricted to LooCI middleware. On the other hand, some MDD examples, such as LWiSSY, Shimizu et al. (2014), and Tei et al. (2015) present solutions offering network-level modelling with fewer implementation restrictions, as these solutions target TinyOS supported platforms, but these three solutions use three modelling layers, which presents a usability problem.

Regarding Languages or solutions **Goal**, some solutions aim to produce application code ready for deployment to a real network, while other solutions are designed to help users to simulate the modelled network application and provide some statistics regarding network performance, which is one of the network expert's duties, rather than the domain experts.

Regarding **data processing**, there are two techniques, data aggregation and data fusion; data aggregation is the process of collecting the sensed data and running simple processing activities to eliminate redundancy. Data fusion is the process of collecting sensed data to expose environmental events. Till now, no rigid and comprehensive framework can unify the available different techniques and algorithms proposed by the research community (Essaadi et al., 2017). Furthermore, there is terminology confusion and no standard definition that explains the scope of data fusion (Nakamura, Loureiro, & Frery, 2007); for example, Akyildiz & Vuran (2010, p. 201) state that in some situations data aggregation is referred to as data fusion. Therefore, we notice in our Table 2-5 that 27% of the discussed solutions do not support any data processing activities, 67% of the solutions support simple data aggregation tasks, and only 7% of the solutions support both data aggregation and fusion, which correlates with the work presented by Tei et al. (2015). Tei et al. (2015) try to offer simple data processing activities such as max, min and present them as data fusion tasks.

Concerning **target platforms**, it is evident that 60% of the proposed solutions and defined languages target TinyOS, due to its importance in the WSN domain. Our percentage appears comparable to the results found by Essaadi et al. (2017). Finally, most of the proposed solutions and modelling languages are **evaluated** using either case study, user study or both.

2.5 Formulated Design Requirements

According to the principal findings already discussed, we have developed a set of design requirements to design and build our proposed solution.

DR1. **Goal**: considering that our research focuses on reducing complexity for beginner developers, network analysis and optimisation are outside that scope, and **code generation** is the primary goal of our proposed solution.

DR2. **Developer Types**: **beginner developers** are the main target to define a new development abstraction according to thesis objectives (**O2**)

DR3. **Strategy:** According to the discussed facts, using the **MDD** strategy to define and build a new application development is a reasonable choice. Regarding the number of required layers, most discussed solutions and proposed languages divide their offered activities into multiple layers. The first layer enables the user to model the required application logic functionality, while the other layers are related to network functionalities and performance analysis. However, expanding the number of modelling layers leads to usability problems. Therefore, defining one modelling layer that enables users both to model the application logic and network functionalities will be the right solution.

DR4. **Language Type**: Using embedded or external DSL are offered, and possible choices, but using **external DSL** seems the more suitable solution, because by choosing internal DSL, defining new high-level vocabularies for the necessary commands and tasks will be limited to the host languages compiler rules and reserved words.

DR5. **Modelling Scope**: Provide a solution that offers **node**, **group**, and **network** modelling.

DR6. **Data Processing**: According to the mentioned limitations concerning data fusion, **data aggregation** tasks can be offered.

DR7. **Concrete Syntax**: offering **text** and **visual** modelling facilities appear an attractive factor for beginner developers to use the proposed solution.

DR8. **Target Platform**: to build a promising solution, it should be based on an affordable and usable platform; **TinyOS** seems the perfect platform.

DR9. Evaluation Method: using case study and user study to evaluate the proposed solution.

Table 2-6: Summary of the Concluded Design Requirements

	Category	Required Features
DR1	Goal	Code Generation
DR2	Developer Types	Beginner Developers
DR3	Development Strategy	Model-Driven Development (MDD)
DR4	Language Type	External DSL
DR5	Modelling Scope	Node, Group, Network
DR6	Data Processing	Data Aggregation
DR7	DSL Concrete Syntax	Text and Visual
DR8	Target Platform	TinyOS
DR9	Evaluation Method	User Study & Case Study

2.6 Chapter Summary

In this chapter, we have provided a background review related to WSN technology, by explaining its history, discussing the most popular hardware sensor nodes, and introducing TinyOS, one of the most frequently used operating systems in WSN. This was followed by explaining how the complexity problem can be solved using Software Engineering practices, represented by MDD and DSL. Next, the three main approaches used to define WSN development abstractions were outlined: Embedded Operating System, Customised Platform and MDD, as well as surveying the most influential WSN software development initiatives proposed by the research community for each approach. These initiatives were then analysed in terms of a set of criteria to find their pros and cons. Finally, a set of requirements based on the analysis findings were identified as design requirements for our proposed solution.

Chapter 3: SenNet Meta-Model

3.1 Introduction

To introduce an abstraction development to facilitate the development process for WSN domain, first we have to map the main tasks and activities in the application-level with the required network elements in the network-level, by defining the basic components and concepts for these two domains and identifying the relationship between them. If the developer needs to send a message, they need to define the message text (the data that will be included in the message), the message structure (message properties such as message size), and finally, the communication means that will be used to send this message. So, from a developer's point of view, they need to send a message (application-level), which in turn reflects the necessity of defining the message text, structure and the communication means from the network-level; in other words, finding the right methodology they should follow to implement the necessary application.

This chapter explains how a SenNet meta-model is used to link and mapped the application-level that represents the developer's point of view with the network-level activities. This meta-model was developed according to the general principles of ISO/IEC29182-SNRA. Many researchers' findings have been considered throughout the course of developing this meta-model, including Tei et al. (2015); Shimizu et al. (2014); Dantas et al. (2013); Rodrigues et al. (2011); and Akbal-Delibas, Boonma, & Suzuki (2009). This was followed by discussing the type of applications supported by this meta-model.

3.2 Modelling and Meta-Modelling

Modelling is a representation of a system or domain according to specific criteria or perspective. The system can be represented using a set of different models, each one capturing a specific aspect. Because the model is the representation of a system in reality, it cannot represent all aspects of reality, which allows developers to deal with the systems in a simplified, safer and cheaper manner (Rothenberg, 1989).

Meta-modelling represents the deep "structures" of models (Goeken & Alter, 2009), and is defined as "*the structure, semantics and constraints for a family of models*" (Mellor, Scott,

Uhl, & Weise, 2004, p. 14). Meta-Modelling also defines the relationship between different components of a system. Rothenberg (1989) states that meta-modelling is the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modelling a predefined class of problems. Accordingly, meta-modelling can be used as a method to link more than one aspect of a system logically.

To solve a specific reality problem, or present a specific object in the universe, then we start with modelling this problem or object as the first step to abstract them. For deep analysis and modelling of any part of the world (that might be a model), we need three main aspects: (1) The way of modelling, which stands for the language/notations used to model and represent this model; (2) The way of working, which represents the procedures that apply to model this model; and (3) The way of thinking, which represent the guidelines, rules, directions and principles that used in the modelling process (Goeken & Alter, 2009).

Many modelling languages and techniques have been developed to enable developers to model their systems and domains, such as Unified Modelling Language (UML), which is used as a universal technique for object-oriented systems. In the last decade, researchers have started to focus on how to transfer domain knowledge into models, given that using UML to model diverse domains is not an adequate method. Accordingly, many researchers propose using meta-modelling as a modelling technique to model wide spectrum objects and domains in a stepwise method (Kurpjuweit & Winter, 2007). Some other researchers believe that the first step in any research activity should start with describing a meta-model in an epistemological way (Moody, 2005; Schuette & Rotthowe, 1998).

After reviewing the difference between modelling and meta-modelling, and showing the advantages of using meta-modelling, one more important question should be answered: Can a meta-model express a development methodology or process?

Curtis, Kellner, & Over (1992) expressed their concerns about the inadequacy of most techniques used to represent and model software development lifecycles. Later, Henderson-Sellers & Bulthuis (1998) proposed using meta-modelling to create more rigorous methodology/process models, and they called meta-modelling "*methodology modelling*", after which many researchers advocated this proposal and believed in using meta-modelling as a method to model development methodology (Gonzalez-Perez, McBride, & Henderson-Sellers, 2005).

Finally, by considering our research first objective **O1:** to define a new model that logically links the application thinking scenarios with the real network elements and domain constraints, considerate is necessary to determine the method that will be followed to develop the required application, as this will help the developer to develop the required components related to network elements to achieve the required application. According to the literature, the best method to represent this methodology is using meta-modelling.

3.3 SenNet Meta-Model

In SenNet, the meta-model represents the rules that the WSN applications should follow. Figure 3-1 shows the meta-model that reflects the logical linking between the application level (developer's point of view) and the network elements. The SenNet meta-model is divided into Application Configuration and Network Configuration.

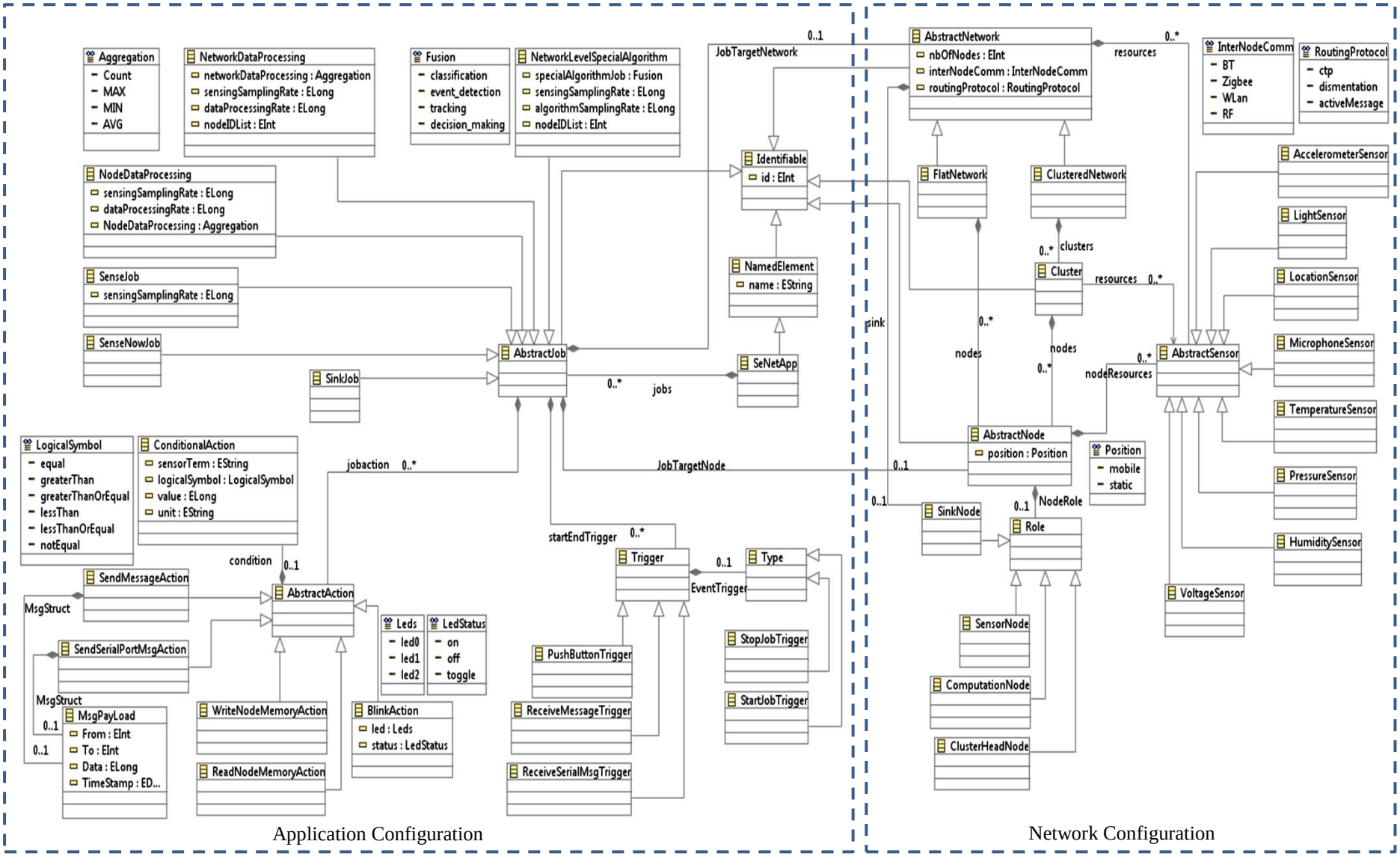

Figure 3-1: WSN Meta-Model

3.3.1 Application Configuration in SenNet Mata-Model

The Application Configuration includes high-level application settings, such as the number of jobs an application performs and the job type (event-driven or periodic). A job can be sense-forward, event-trigger, or data processing (node or network-level data aggregation). Node-Level data aggregation is designed for data-processing operations and activities that each node can apply to its local data. Network-Level data aggregation is designed for special node roles, where data processing algorithms are applied to other nodes' data.

- **AbstractJob:** is the general template that includes all job types that developers could use; these jobs can be summarised as below:
    1. **SenseJob:** this job type is the most frequently used job by the developers as it stands for the sensing and monitoring application job. This job is a continuous job; the developer should set the sampling frequency to configure the time to initiate the job periodically. The job type enables the developers to build periodic sensing and sense-forward application types.
    2. **SenseNowJob:** this job is a single sensing job that will do the sensing operation once, where data will be ready for user manipulation. The difference between SenseJob and SenseNowjob is that SenseJob uses Split-Phase operations (Gay et al., 2003; Levis, 2006). SenseNowJob uses blocking / synchronous, where the CPU will start a loop, waiting until the reading process is complete. In contrast, the split-phase operation is non-blocking / asynchronous, then the CPU will fire the read sensor value command, and will not wait until the operation is complete, but will start processing another command. When the reading process is complete in the split-phase operation, then the hardware will initiate an interrupt, which is called call-back, where the CPU will process the reading value; Figure 3-2 shows the difference between synchronous/asynchronous. Usually, split-phase operations are used with:
        A. Hardware that takes a long time to complete its task.
        B. Reading samples that need to be sensed periodically.
        C. In Split-Phase operations that enable the node to start more than one operation at the same time, and they execute in parallel.

Blocking	Split-phase

state = WAITING;
op1();
sleep(500);
op2();
state = RUNNING
	state = WAITING;
	op1();
	call Timer.startOneShot(500);
	
	event void Timer.fired() {
	  op2();
	  state = RUNNING;
	}

Figure 3-2: The Difference Between Synchronous / Asynchronous Programming Models ("Getting Started with TinyOS," n.d.)

3. **NodeDataProcessing:** is the job type that is used to apply all data processing on the node-level, which means it enables the sensor node to aggregate its data that getting from its sensors. This sub-job type is equipped with a set of predefined algorithms related to data processing, which can be found in Aggregation enumeration, such as finding the minimum, largest, average and count values.
4. **NetworkDataProcessing:** this kind of job uses specific predefined algorithms to apply them on sensed data obtained from other nodes, such as finding the minimum, largest, average and count values; and these algorithms can be found under aggregation enumeration. According to (Essaadi et al., 2017), this type of functions is more attracted to developers than usual fusion algorithms, as they are easy to model and program.
5. **NetworkLevelSpecialAlgorithm:** this type of job is used to apply a particular type of algorithms such as object identification, tracking, decision making or any other special algorithms. The current version of SenNet meta-model is developed to be a generic meta-model. Accordingly, it is not practical to include specific domain algorithms in SenNet, and it will be the role of other developers to customise this job towards a particular domain, such as human health or identification purposes and including special algorithms belonging to the required domains.

- **Trigger:** this is a trigger that can be used by developers to add some intelligence to the network to start or end a specific job according to a specific trigger. These triggers can be summarised either by receiving a message or pressing the push button.
- **Action:** this represents the actions that the developers could configure their applications to use after finishing the sensing job or any other job type. There are many action types such as sending a message, writing the sensed data to the node memory or blinking a led. In

addition, the SenNet meta-model offers ConditionalAction, which means that the action will be done according to a specific condition.

3.3.2 Network Configuration in SenNet Mata-Model

The Network Configuration includes the configuration parameters for the network, cluster (a group of nodes) and the sensor node. It also organises the relationship between a single node on one side and a cluster (group of nodes) or network from the other side. Furthermore, it is used to configure the type of communication technology and routing protocols, the position of the sensor node if it is static, or mobile sensor node. It can be divided into three main parts or entities ***Network***, ***Sensor Node*** and ***Sensors***. One of the advantages of the SenNet meta-model is managing the relationships between the network elements (network, sensor node and sensor).

- **Network:** SenNet meta-model divides the network into two types, FlatNetwork and ClusterNetwork, where each network type has its sensor node types.
- **Node:** SenNet meta-model provides many node roles to choose according to the network type. For FlatNetwork, there are ***SinkNode***, ***ComputationNode*** and ***SensorNode***, where ComputationNode is a special role node devised to implement complex network-level data processing algorithms, so its role in the network is to collect data from its neighbouring nodes and implement the required data processing algorithms; moreover, the developer can assign any sensing job to this node role, so ComputationNode can be considered as a cluster head node in the cluster network. On the other side, ClusterNetwork node types are ***SinkNode***, ***SensorNode*** and the ***ClusterHeadNode***.
- **Sensors:** The available types of sensors that could be used within the network, cluster (a group of nodes) and sensor nodes, such as humidity, temperature and pressure sensors.

3.4 ISO/IEC 29182 – Sensor Network Reference Architecture

The ISO/IEC29182-SNRA reference architecture standard is one of the main resources for WSN Knowledge. This reference architecture defines three domains that WSN systems will interact with: sensing, network and service domain. Sensing domain deals with the real environment and how the sensor nodes will extract the needed information from the environment, while the network domain is related to the wireless sensor network and how the network elements are related to and communicate with each other. Finally, the service domain is related to the defined tasks and jobs available for developer's use as needed. In terms of these

three domains, ISO/IEC29182-SNRA standard defined many entities that play a significant role in the WSN application. Table 3-1 shows the main functional entities listed in ISO/IEC29182-4: 2013, Sensor Network Reference Architecture (SNRA) - Entity Model (ISO/IEC, 2013). Table 3-2 shows the main explained entities that defined in ISO/IEC29182-SNRA and how they modelled within the SenNet meta-model.

Table 3-1: ISO/IEC29182-4 Main Functional Entities (ISO/IEC, 2013)

FUNCTIONAL ENTITIES	DESCRIPTIONS
DATA ACQUISITION	Sense and capture data from the environment for applications.
DATA STORAGE	Store sensor data, control instructions, and management data.
DATA PROCESSING	Use data/signal processing algorithms to extract requested or useful information from sensor data and metadata. The information extraction algorithms include collaborative information processing (e.g. data fusion, feature extraction, data aggregation and data presentation).
DATA COMMUNICATION	Transmit and receive data among sensor nodes and sensor network gateway through a communication protocol stack and communication support functions. Examples of data transmitted and received are temperature, humidity, time synchronisation and location data.
NETWORK MANAGEMENT	Manage the network topology, routing table, configuration information, performance and reconfigure network information.
FEEDBACK & CONTROL	Trigger control instruction on actuators according to user's feedback depending on the application requirement. Whether feedback control is needed or not depends on the application requirement.
DATA AGGREGATION	Data aggregation assembles the similar data from multiple sources (e.g. sensors, processors, databases). It also may assemble the data chronologically when needed.
DATA COMMUNICATION	Transmit and receive data among devices using existing or emerging data transmission technologies.
BUSINESS MANAGEMENT	Manage the business procedure, business rules, business operations and statistical analysis of business data. Whether business management is necessary depends on application requirement

Table 3-2: SenNet Meta-Model Mapping to ISO/IEC29182 Functional Entities

ISO/IEC 29182 Functional Entities	**SenNet Meta-Model**
Data acquisition	AbstractJob
Data storage	• ReadNodeMemoryAction • WriteNodeMemoryAction
Data processing	• NodeDataProcessing • NetworkDataProcessing • NetworkLevelSpecialAlgorithm
Data communication	• SendMessageAction • ReceiveMessageTrigger
Network management	AbstractNetwork
Feedback & control	• Action • Trigger
Data Aggregation (Node-Level)	NodeDataProcessing
Data Aggregation (Network-Level)	NetworkDataProcessing
Data communication	AbstractNetwork • interNodeComm (parameter) • routingProtocol (parameter)

Business Management	• AbstractJob • Action • Trigger

The ISO/IEC29182-3:2014, Sensor Network Reference Architecture (SNRA) - Reference architecture views (ISO/IEC, 2014a) illustrates the general sequence of a sensor network's operations and activities required to accomplish a specific task, as shown below in Figure 3-3 which summarises the sequence of operations. It also describes the general sensor network system functionality and the flow of data inside the system. Figure 3-4, which shows the data produced and consumed by each entity, summarises the general system functionality.

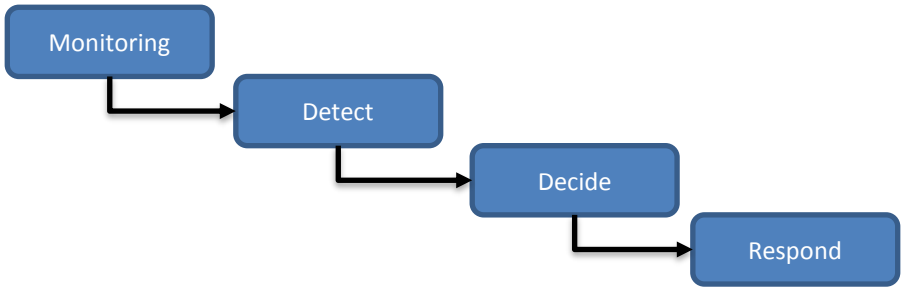

Figure 3-3: Physical Operational Activity Model

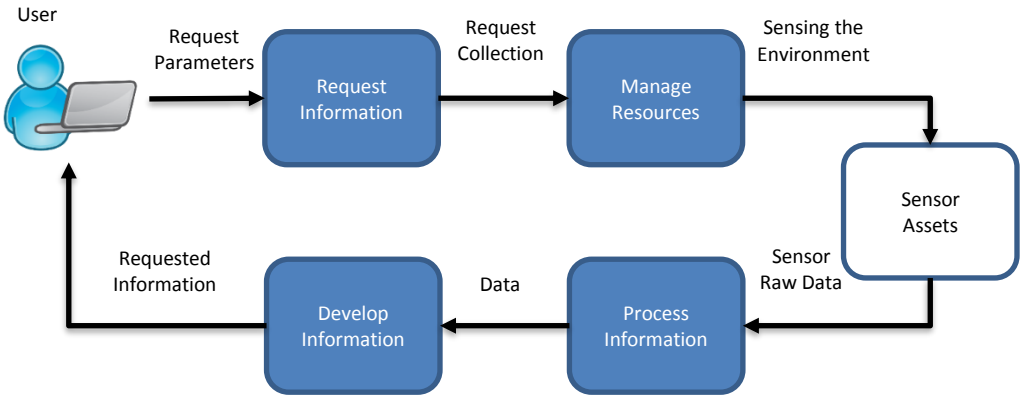

Figure 3-4: General Sensor Network System Functionality Diagram

3.5 Application Types Supported by the SenNet Meta-Model

To find the application types that are supported by the SenNet meta-model, Oppermann, Boano, et al. (2014) propose an interesting WSN application taxonomy: the main criteria used to categorise application types are application **Goal** and **Sampling**. Table 3-3 shows the different application types that are supported by the SenNet meta-model according to Oppermann's taxonomy criteria goal and sampling.

- **Goal**: General WSN function is to sense the environment and forward the sensed data to a central location, where the application logic will be applied and useful information extracted, and this job is called ***sense-forward*** or ***sense-and-forward***. In some other cases, it is useful to put some logic inside the network and enable the network to become an intelligent network, according to specific logic rules. This case of application is very useful as it will reduce the amount of transferred raw sensed data to the sink node, which implies that the amount of communication will be reduced. Finally, according to these logic rules and the getting sensed data, the network will react, this job type is called ***sense-react***. Regarding SenNet meta-model, then our meta-model supports a sense-forward application using SenseJob type, besides using SendMessageAction which is one of the action types offered by the SenNet meta-model. On the other side, regarding sense-and-react, due to the fact that this research scope is limited to WSN monitoring application, our SenNet meta-model does not support actuation activities; however, simple react activities are supported by our SenNet meta-model, such as LED blinking or sending a message to a specific node. Thus, the SenNet meta-model supports simple sense-react application scenarios.
- **Sampling**: many application scenarios can apply to WSN, but in terms of sampling these applications can categorise using either ***periodic*** or ***event-trigger*** sampling. Regarding periodic sampling, the SenNet meta-model supports periodic sampling or sensing using SenseJob which supports periodic sampling after the sampling frequency has been set by the developer. SenNet meta-model also supports event-trigger applications. SenNet meta-model offers two event-trigger types:
    1. **Developer Event-Trigger:** some researchers call this type software event-trigger, which means that the developer performs some actions to trigger the start or ending of a specific job or a set of activities. This trigger can be sending a message, or pressing a push button. This type of job can be done through SenNet meta-model using any AbstractJob type, as well as Trigger.
    2. **Environmental Event-Trigger:** this means some events have occurred in the environment that could be utilised to trigger the WSN to do some actions. SenNet meta-model enables this type of event-trigger using AbstractJob and AbstractAction with the aid of ConditionalAction. That enables the developer to set any job type to do either one action or a set of actions according to specific environmental conditions, such as while the network is running a periodic temperature sensing operation, then if the sensed temperature reaches 40 C°, then

the node or the set of nodes will send a message to the Sink, blink a led, or save the reading to the node memory.

Table 3-3: Mapping SenNet meta-model to Taxonomy Criteria (Oppermann, Boano, et al., 2014)

Criteria		SenNet Meta-Model	
		Supported or Not	How
Goal	Sense-Forward	✓	SenseJob + SendMessageAction
	Sense-React	✓	SenseJob + AbstractAction
Sampling	Periodic	✓	SenseJob
	Event-triggered	✓	AbstractJob + Trigger
			AbstractJob + AbstractAction + ConditionalAction

3.6 Chapter Summary

In this chapter, we have described the SenNet meta-model, which forms a base for SenNet language. One of the definitive sources for WSN technology is the ISO/IEC 29182 standard; therefore, the SenNet meta-model was defined in accordance with this standard. The SenNet meta-model was defined to map the application-level to the required network-level. Therefore, this meta-model can be divided logically into two parts: Application configuration and Network Configuration. The first part represents the main tasks and activities that can be used by developers to build their required applications, while the Network Configuration represents the network components and operations that need to be applied to implement the required application-level tasks and activities.

WSN applications can be divided into in many categories, such as Sense-Forward, Sense-React, Periodic, and Event-Triggered applications. SenNet meta-model supports developing all these application types, as presented in section **3.5**, which discussed how these applications could be developed using our meta-model.

The SenNet meta-model was developed using Eclipse Modelling Framework (EMF) (Steinberg, Budinsky, Paternostro, & Merks, 2008). EMF is one of the core technologies of the Eclipse universe, which is a Meta Object Facility (MOF) that is used to define meta-models (Stephan & Cordy, 2012). Ecore is the model name used to represent any model in EMF (Steinberg et al., 2008).

Chapter 4: SenNet Language - Internal View

4.1 Introduction

This chapter focuses on how the SenNet meta-model language has been developed, as well as its internal architecture and functionality, including the technologies and development techniques utilised to develop SenNet's components, such as Eclipse, Eclipse Modelling Framework (EMF), Xtext, Xtend, and Guice. Accordingly, this chapter first discusses the SenNet internal view, then explaining CPC and CGC components. Finally, a simple and basic plan is outlined for updating and adding new algorithms and activities to SenNet's functionality.

4.2 SenNet Internal View

SenNet is a DSL that translates its source code into source code of another existing language (nesC), this type of language is called source-to-source transformation DSL type (Kosar et al., 2008). Implementing a source-to-source transformation DSL is not a trivial task. Accordingly, SenNet was implemented using two transformation stages, the first stage is text-to-model transformation, and the second stage is model-to-text transformation.

SenNet includes CPC and CGC components; the CPC component is responsible for providing the necessary commands library and IDE functionalities. In addition, it implements all the required source code analysis to produce the AST model. So, this component represents the text-to-model (T2M) transformation stage. The CGC component is responsible for generating the necessary nesC source code for each sensor node defined in the SenNet application, which represents the model-to-text (M2T) transformation stage.

SenNet was developed using the Eclipse development environment, which is becoming a widely used platform to build DSLs and modelling tools (Kolovos, García-Domínguez, Rose, & Paige, 2017). All SenNet components and their internal processes are controlled and wired to each other via a runtime module using Guice[9], which is a lightweight Dependency Injection (DI) framework from Google. Dependency Injection (DI) is one of the inversion control techniques that transfer the responsibility for the creation and linking of SenNet components

[9] Google Guice: https://github.com/google/guice

and sub-components to an externally configurable framework that would reduce the coupling in the SenNet environment, which offers great flexibility to control the interaction and linking between SenNet components and their internal processes. Figure 4-1 shows the general implementation architecture for SenNet. In the same way, Figure 4-2 illustrates an abstracted high-level SenNet functionality model diagram.

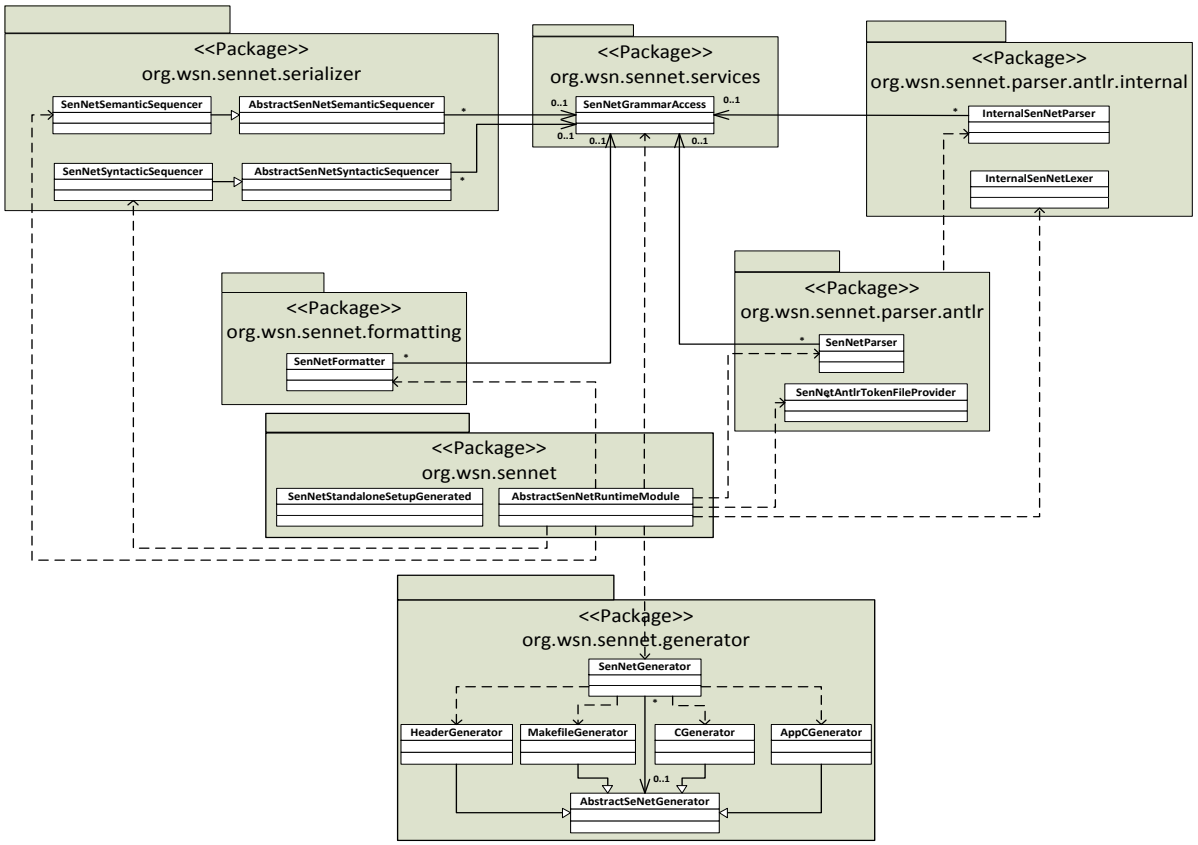

Figure 4-1: SenNet Implementation Architecture

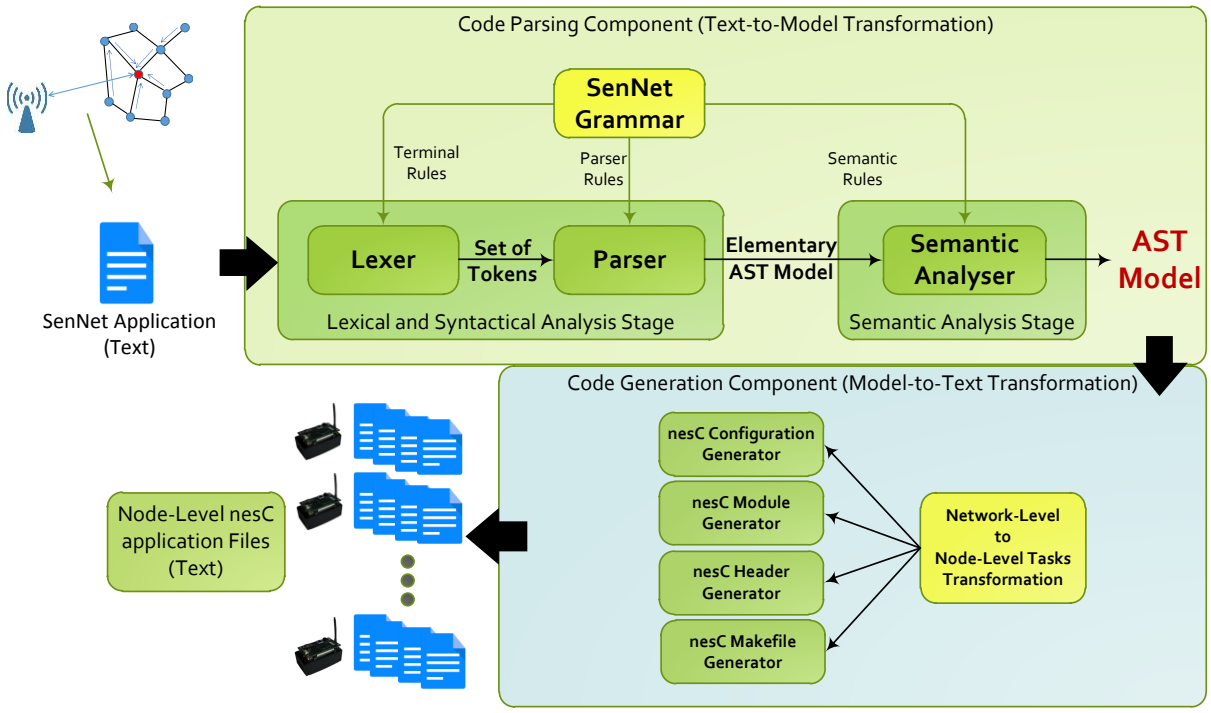

Figure 4-2: High-Level SenNet Functionality Model

4.3 Code Parsing Component (CPC)

The CPC component's role is not limited to providing developers with IDE functionalities, where all the compilation and code analysis activities are done by this component. The code analysis is done in two stages; the first stage is lexical and syntactical analysis, while the second stage is semantic analysis.

The CPC component was developed using Xtext (Eysholdt & Behrens, 2010), which supports Text-to-Model (T2M) transformation. The CPC component was integrated via Xtext with the Eclipse Modelling Framework (EMF) (Steinberg et al., 2008) to generate the tree-based editor view. Listing 4-1 shows how the SenNet tree-based editor is initialised using the extension to contribute the functionality provided by Eclipse. An extension to the working mechanism is a particular Eclipse plug-in (EMF) feature that offers an extension point, which allows other plug-ins (Xtext) to use or customise part of its functionality. An extension point is a contract that includes a combination of XML and Java interfaces that all extensions made by other plug-ins should conform to.

Listing 4-1: SenNet Tree-Based Editor View Initialisation

```
<extension point="org.eclipse.ui.editors">
   <editor
         id="org.wsn.sennet.presentation.SennetEditorID"
         name="%_UI_SennetEditor_label"
         icon="icons/full/obj16/SennetModelFile.gif"
         extensions="sennettree"
         class="org.wsn.sennet.presentation.SennetEditor"
        contributorClass="org.wsn.sennet.presentation.SennetActionBarContri
    butor">
   </editor>
</extension>
```

The source code analysis run by the CPC is done through using a context-free grammar (CFG), which is used in most of the modern languages used to specify the grammatical structure of programming languages (Aho, Lam, Sethi, & Ullman, 2007; Fowler, 2010). CFG is called context-free because it is hard to restrict rules to a certain context, due to the lack of the context-sensitive recognition algorithms (Parr, 2007; Voelter et al., 2013). Instead, syntactic and semantic rules are used to achieve this purpose. Extended Backus-Naur Form (EBNF) (ISO/IEC, 1996) is a textual representation used to build this type of grammar. EBNF is a DSL that was developed for the purpose of creating rules (Schmitt, Kuckuk, Kostler, Hannig, & Teich, 2014): each rule had a name and a list of alternative meanings. Listing 4-2 shows a simple representation of EBNF rules. CFG rules are parsed using ANother Tool for Language Recognition (ANTLR) parser, that is offered by Xtext as a back-end parser. Using ANTLR and EBNF techniques enables programmers and developers to define their programming languages grammar in one grammar file (Bettini, 2013). SenNet grammar rules can be found in APPENDIX D.

Listing 4-2: EBNF Sample Rule

```
expression
: INT
| expression '*' expression
| expression '+' expression
;
```

4.3.1 Lexical and Syntactical Analysis Stage

To run and execute a program written in a specific programming language, we have to make sure that the program respects the syntax of that language. To this end, the Lexical analysis

stage is related to analysing the source code and producing the AST model, and this process is done using lexical and syntactical analysis steps.

- **Lexing Step:** this step involves breaking the program into small chunks called tokens. Lexer divides the SenNet application into small tokens. Tokens are created according to specific rules called lexical or terminal rules; Listing 4-3 shows an extracted sample of SenNet terminal rules.

Listing 4-3: Sample of SenNet Grammar Showing Terminal Rules

```
terminal ID : '^'?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
terminal INT returns ecore::EInt: ('0'..'9')+;
terminal STRING : '"' ( '\\' . /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ |
!('\\'|'"') )* '"' | "'" ( '\\' . /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ |
!('\\'|"'") )* "'";
terminal ML_COMMENT : '/*' -> '*/';
terminal SL_COMMENT : '//' !('\n'|'\r')* ('\r'? '\n')?;
terminal WS : (' '|'\t'|'\r'|'\n')+;
```

- **Parsing Step:** The second analysis step is syntactical analysis. After generating the sequence of tokens from the input program, we have to confirm that the sequence of tokens forms a valid statement in the language and respects its syntactical structure, in addition to formulating a tree that is called the Node Model or Abstract Syntax Tree (AST). Tokens are processed and analysed according to the parser or EObject (Eclipse Object) rules, because they work as a kind of building plan for the creation of the EObject that will form the AST in memory with the aid of EMF resources. AST represents the abstract syntactic structure of the SenNet application. After finishing the parsing step, a representation of the parsed program is already built and stored in memory, where further semantic analysis can be applied without parsing the same SenNet application text over and over again. Listing 4-4 shows an extracted sample of SenNet parser rules.

Listing 4-4: Sample of SenNet Grammar Showing Parser Rules

```
AbstractNode: SensorNode | ClusterHeadNode | SinkNode | ComputationNode;

AbstractNetwork: FlatNetwork | ClusteredNetwork;

AbstractStartEndingJobTrigger: ReceiveMessageTrigger | ReceiveSerialMsgTrigger
| StartJobTrigger_Impl | StopJobTrigger_Impl | PushButtonTrigger;

AbstractAction: ReadNodeMemoryAction | WriteNodeMemoryAction |
SendMessageAction | BlinkAction | SendSerialPortMsgAction;
```

4.3.2 Semantic Analysis Stage

This stage is related to all logical analysis that is done on the AST model. Many researchers advise language developers to distribute the required code analysis on different levels, this technique provides much better error messages and more precisely detected problems (Bettini, 2013), and would be better than processing the same program text over and over again. Thus, all logical conditions and constraints that cannot be processed in the lexical and syntactical analysis can be handled at this stage to ensure the AST model complies logically with the semantic rules. For instance, Listing 4-5 shows an extraction sample of SenNet grammar that includes semantic rules related to SenseJob job type, that highlight whether SenseJob is initiated, then define the internal parameters, such as id, jobaction, and start/EndTrigger are optional, except that the sensingSamplingRate parameter should be defined.

Listing 4-5: Sample of SenNet Grammar Showing Semantic Rules

```
'SenseJob'
      '{'
            ('id' id=EInt)?

            ('sensingSamplingRate' sensingSamplingRate=ELong)

            ('JobTargetNode' JobTargetNode=AbstractNode)?

            ('JobTargetNetwork' JobTargetNetwork=AbstractNetwork)?

            ('start/EndTrigger' '{'
      start/EndTrigger+=AbstractStartEndingJobTrigger ( ","
      start/EndTrigger+=AbstractStartEndingJobTrigger)* '}' )?

            ('jobaction' '{' jobaction+=AbstractAction ( ","
      jobaction+=AbstractAction)* '}' )?

      '}';
```

4.4 Code Generation Component (CGC)

The CGC component's main function is to generate the necessary nesC code. This component was developed according to the fact that each nesC program is divided into static and dynamic code. The static code represents the program structure and components to be used, while the dynamic code represents the developer's preferences in building the program. Consequently, Figure 4-3 shows how a sample nesC program can be divided into the static and dynamic code, where the configuration and declaration parts can be considered as the static part while the implementation part can be considered as the dynamic part.

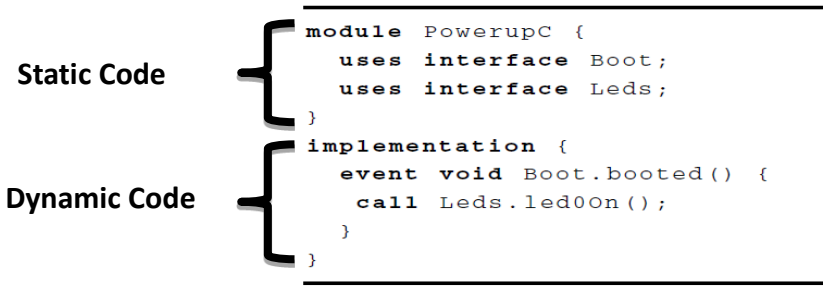

Figure 4-3: General nesC Module Program Structure

The CGC component analyses the application AST that is generated by the CPC and translates all the general network activities into a single-node nesC format. The final nesC code for each sensor node is produced by injecting the generated application code into pre-defined templates. The pre-defined templates plus the injected application code represent the final sensor node program in nesC format. More over to the advantages of using TinyOS as a foundation for SenNet that will be described in below section, the nesC pre-processor has no dynamic linking, so it omits all un-accessed variables, librarries and unused code (Levis & Gay, 2009) to reduce the program size in memory and decrease the C compilation time.

The CGC component was developed using Xtend (Klatt, 2007) which supports Model-to-Text (M2T) transformation (Birken, 2014). To generate code the CGC component should be capable of accessing the AST model that was formed by the CPC, besides having the ability to create new textual files that include a specific format. The CGC component uses an IGenerator class that belongs to the runtime module, which helps CGC achieve these functionalities through the doGenerate method, and transfer these capabilities to other generators included in this component.

4.4.1 Why Use TinyOS as a Foundation for SenNet?

It has been noticed that the most used operating systems in the WSN applications domain are TinyOS (Hill et al., 2000) and Contiki (Dunkels et al., 2004) because of their technical support and maturity. According to Delamo et al. (2015), 85% of the WSN projects were using TinyOS. TinyOS is used in many commercial products such as Zolertia[10], Cisco's smart grid systems[11] (formerly Arch Rock), and People Power[12], and it has approximately 25,000 downloads per

[10] http://zolertia.io/
[11] http://www.cisco.com/c/en/us/solutions/industries/energy/external-utilities-smart-grid.html
[12] http://www.peoplepowerco.com/

year. The last TinyOS core version reached more than 80,000 Lines of Code with approximately 39 contributors, which reflects the maturity and continuous support, which are a highly important factor for any software success story (Levis, 2012).

The second version of TinyOS that was released in November 2006 presents some improvements over the earlier version. The most important improvement in this version is a Hardware Abstraction Architecture (HAA) (Handziski et al., 2005), which is a three-layer architecture:

- Hardware Independent Layer (HIL), which is the top layer that provides access to the underlying hardware, such as radio, storage, and timer, in a platform-independent way, enabling the developer to access the hardware using independent interfaces. Accordingly, the HIL layer presents a general interface for operations that are available on multiple platforms.
- Hardware Adaptation Layer (HAL), which provides a high-level abstraction of the underlying hardware to simplify the use of the complex hardware. So, HAL is platform-specific, but developers should use hardware-independent interfaces when possible (Levis & Gay, 2009).
- Hardware Presentation Layer (HPL), which is the lowest level of the architecture that sits directly above the hardware. This layer's purpose is to abstract the irrelevant differences between similar hardware and present the hardware components as a friendly nesC interface.

The advantages of this architecture can be shown in a simple application called AntiTheft[13] which is an application that uses a light sensor component (PhotoC) and an accelerometer component (AccelXStreamC) that belongs to a particular sensor board (MTS310) for a particular platform family (MicaZ). However, this application uses independent hardware interfaces to access these two components; these interfaces are Read and ReadStream. Consequently, this application can be ported to work with any other sensor nodes that include equivalent sensors.

In summary, applications that use HIL interfaces, or HAL hardware-independent interfaces are easy to port to different sensor nodes that meet the application's hardware requirements. In

[13] AntiTheft Example Application is one of the application examples that offered by TinyOS team: https://github.com/tinyos/tinyos-main/tree/master/apps/AntiTheft

contrast, applications that use HAL or HPL via hardware-dependent interfaces will be complex to port to different sensor nodes. Currently, TinyOS supports a wide range of sensor node platforms, such as Mica, Telos, Imote2, and IntelMote2; the updated list of supported platforms can be found on the official TinyOS website[14].

The current programming language used to program a sensor node is nesC (Gay et al., 2003), which is a component-based programming language that is included with the TinyOS operating system. The developer should develop the required files using nesC language commands, where these files are compiled using TinyOS operating system to produce the necessary files in C language format, and then these files will be converted into a binary format, as depicted in Figure 4-4. Finally, the required application binary code is ready for deployment to the actual sensor node hardware. The developer should define and program more than one file, for example configuration and module files, to implement any application scenario; those files include the main component definitions that will be used in the application and the code logic respectively, in addition to the makefile, which is used for headless software.

Using TinyOS as the foundation of the SenNet means that SenNet can be used with a wide range of sensor node types and platforms. Moreover, it will utilise the existing components and functions that are included with TinyOS, so there is no need to build them from scratch.

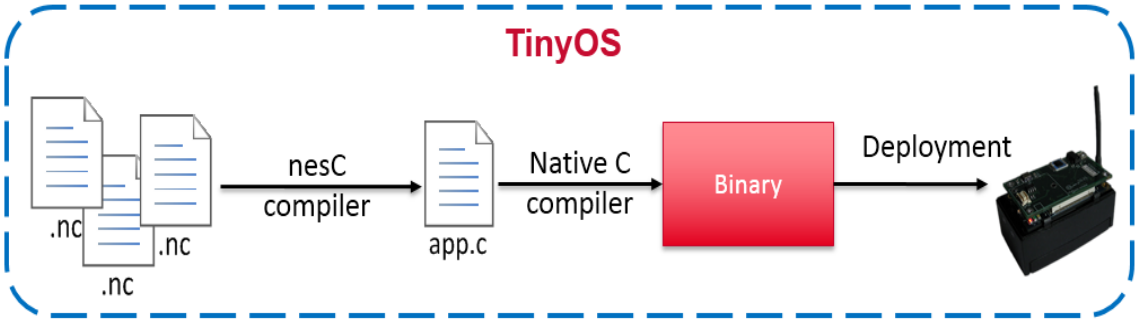

Figure 4-4: Sensor Node Programming Conventional Process

[14] Official TinyOS webpage that list the updated supported hardware platform families: http://tinyos.stanford.edu/tinyos-wiki/index.php/Platform_Hardware

4.4.2 Code Generation Process

The CGC implements the code generation functionality in two steps, the first step is the network to node activity transformation, and the second step is triggering the single node generators to start to generate code.

- **Network to Node-Level Tasks Transformation Step:** this step involves accessing the generated AST model, which considers the final output of the CPC component and makes the necessary analysis to find the number of jobs, and the type of network elements that these jobs are assigned to, before deciding the final number of sensor nodes, and naming the files using appropriate naming convention; Listing 4-6 illustrates how names identified for each node defined in SenNet application. For example, the SenNet application defines a certain job that is assigned to a sensor node, and the developer specifies the node-id. Then all generated files are named according to this rule (`nodeName = app.name + jobTargetNode.id`), where app.name is the application name specified by the developer at the beginning of SenNet application. Likewise, if the SenNet application includes a job assigned to a network, then the generated files are named according to this rule (`nodeName = app.name + i`), where `i` is a counter that starts from 2, as 1 is reserved for the sink node.

Listing 4-6: Pseudocode of Generated Nodes-Naming Process

```
# NetworkStartingNodeID = 2
if (jobTargetNode != null) then
        nodeName = Application_Name + Node_ID
else
        if (jobTargetNetwork != null) then
                if (node defined within the network as Sink type) then
                        nodeName = Application_Name + 1
                else
                        maximumNodeId = NetworkStartingNodeID + Network.nbOfNodes
                        for (i = NetworkStartingNodeID; i < maximumNodeId; i++) do
                                nodeName = Application_Name + i
                        end-for
                end-if

        end-if

end-if
```

- **Generating the Final nesC Step:** this step is started after finishing the previous one, where the number of sensor nodes has been identified, and their file names also specified. More than one generator may be triggered in this step, according to the application logic embedded in the SenNet application, as shown in Figure 4-5; for example, if the SenNet application includes a message sending facility, then HeaderGenerator.xtend should be triggered to generate AMsg.h that will hold the message structure that the sensor node should follow. Figure 4-6 shows a sample algorithm to implement SenseJob job type using the CGenerator generator. Listing 4-7 and Figure 4-7 shows how SenNet generated code for node and network-level data processing job type. APPENDIX E includes four generators implemented in Xtend language:
    1. **AppCGenerator.xtend** that generates the nesC configuration files
    2. **CGenerator.xtend** that generates the nesC module files
    3. **MakefileGenerator.xtend** that generates the nesC Makefile
    4. **HeaderGenerator.xtend** that generates the AMsg.h files

Table 4-1 illustrates how SenNet sample commands and attributes are mapped into nesC code format for the configuration and module files.

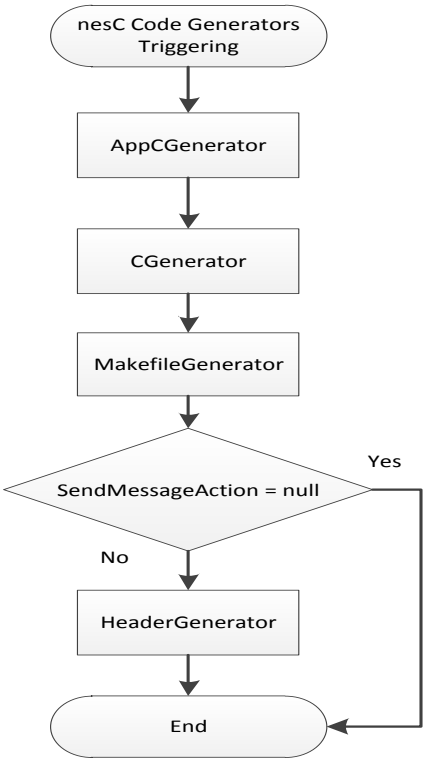

Figure 4-5: Code Generators Triggering Process

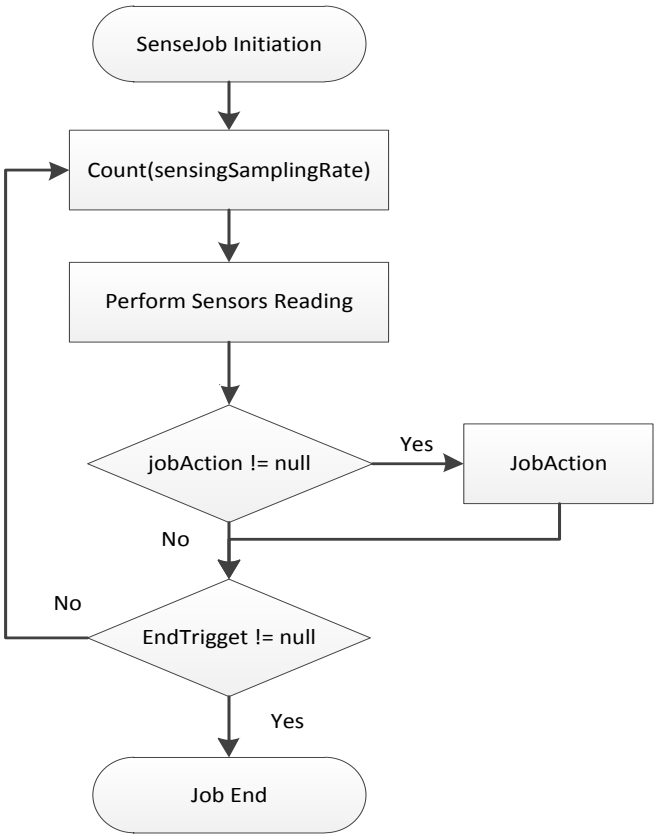

Figure 4-6: The CGenerator Algorithm to Implement SenseJob Job Type

Listing 4-7: Pseudocode of the Generated nesC code for the NodeDataProcessing

```
********** NodeDataProcessing**********
# dataProcessingRate
# sensingSamplingRate
# Aggregation = {Avg, Min, Max, Count}
# Action = {ReadNodeMemoryAction, WriteNodeMemoryAction, SendMessageAction, BlinkAction}
for (i = dataProcessingRate; i < dataProcessingRate; i++) do
        for (j = sensingSamplingRate; j < sensingSamplingRate; j++) do
                Perform sensor reading
        end-for
        perform Aggregation

        if (jobAction != null) then
                perform Action
        end-if
end-for
```

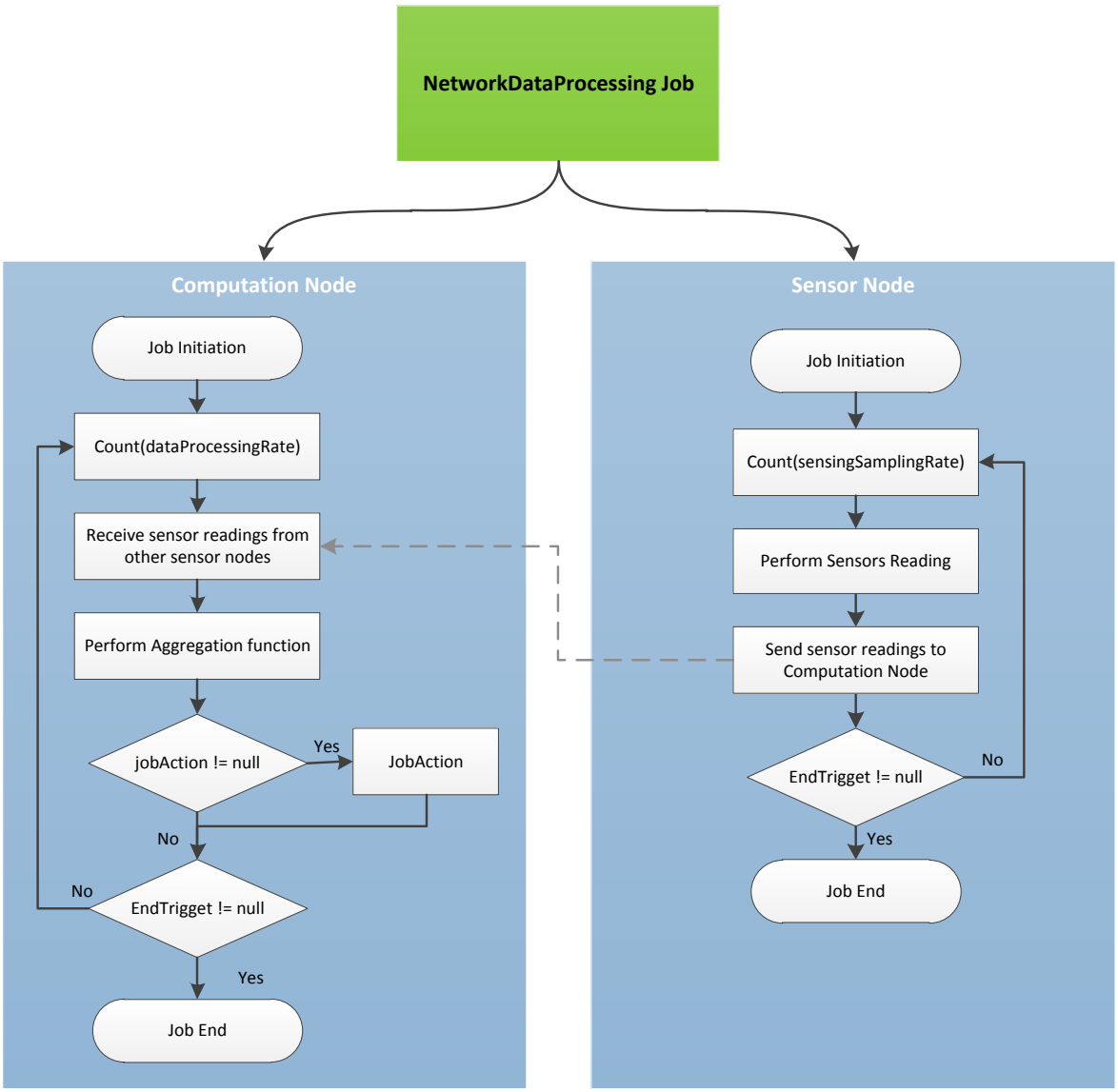

Figure 4-7: The CGenerator Algorithm to Implement NetworkDataProcessing Job

Table 4-1: Sample of SenNet to nesC Code Mapping

SenNet Class				Related nesC code	
Abstract Class	**Sub-Class**	**Attribute**	**User input**	**XXXAppC.nc**	**XXXC.nc**

SenNetAPP		name	XXX	configuration XXXAppC
				{
				}
				implementation {
				components XXXC;
				components MainC;
				XXXC.Boot -> MainC;
				}
					module XXXC
					{
					uses {
					interface Boot;
					interface SplitControl;
					}
					}
					Implementation {
					event void Boot.booted() {
					}
					}

AbstractJob	SenseJob	SensingSamplingRate	Z	components new TimerMilliC();
				XXXC.Timer -> TimerMilliC;
					#include "Timer.h"
					interface Read<uint16_t>;
					interface Timer<TMilli>;
					call Timer.startPeriodic(Z);
					call SplitControl.start();
					Boot.booted ()
					event void Timer.fired()
					{
					}
					event void Read.readDone(error_t result, uint16_t data)
					{
					}
					event void SplitControl.startDone(error_t error){
					if (error != SUCCESS)
					{
					call SplitControl.start();
					}
					event void SplitControl.stopDone(error_t error){
					}

AbstractAction	SendMessageAction	SenMessageTo	Y	components ActiveMessageC;
				components new AMSenderC(AM_RADIO);
				components new AMReceiverC(AM_RADIO);
				XXXC.Packet -> AMSenderC;
				XXXC.AMPacket -> AMSenderC;
				XXXC.AMSend -> AMSenderC;
				XXXC.SplitControl -> ActiveMessageC;
				XXXC.Receive -> AMReceiverC;
					#include "AMsg.h"
					interface Read<uint16_t>;
					interface Packet;
					interface AMPacket;
					interface AMSend;
					interface Receive;
					bool radioBusy;
					message_t messagePacket;
					event void AMSend.sendDone(message_t *msg, error_t error) {
					if (msg == & messagePacket) {

					radioBusy = FALSE;
					}
					}
					event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len){
					return msg;
					}
					
					if (radioBusy == FALSE) {
					ActiveMessage_t* msg = call
					Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
					msg -> NodeID = TOS_NODE_ID;
					msg -> TData = data;
					if (call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t))) {
					radioBusy = TRUE;
					}
					}

	BlinkAction			components LedsC; XXXC.Leds -> LedsC;	interface Leds;

		Led	Led1		call Leds.led0Toggle();
					call Leds.led1Toggle();
					callLeds.led2Toggle();
					call Leds.led0On();
					call Leds.led1On();
					callLeds.led2On();
					call Leds.led0Off();
					call Leds.led1Off();
					callLeds.led2Off();

	ConditionalAction	logicalSymbol	Equal		if (data = 400){ }
		value	40		
AbstractNode	SensorNode			We have to generate only one file for 1 node	We have to generate only one file for 1 node
AbstractNetwork	AbstractNetwork	nbOfNodes	M	We have to generate M number of files, in case the user assign M = 3, then we have to generate 3 files of XXXAppC.nc	We have to generate M number of files, in case the user put M = 3, then we have to generate 3 files of XXXC.nc
Abstract Sensor	TemperatureSensor			components new SensirionSht11C() as TempHumS; XXXC.Read -> TempHumS.Temperature;	
	Humidity			components new SensirionSht11C() as TempHumS; XXXC.Read -> TempHumS.Humidity;	
	Light			components new Taos2550C() as lightS; XXXC.Read -> lightS.VisibleLight;	
	Pressure			components new Intersema5534C() as PressureS; XXXC.Read -> PressureS.Intersema;	
	Voltage			components new VoltageC() as VoltageS; XXXC.Read -> VoltageS.Read;	

4.5 SenNet Extensibility and Update Process

One of the objectives of the SenNet is that it should have the ability to be extended and updated in a simple manner. Figure 4-8 shows how the extension and updating process is done. Experts are needed to update the grammar, and update the CGC component generators, if they need to add new tasks and algorithms. While, for updating the existing tasks and commands, the experts should update the CGC component generators only.

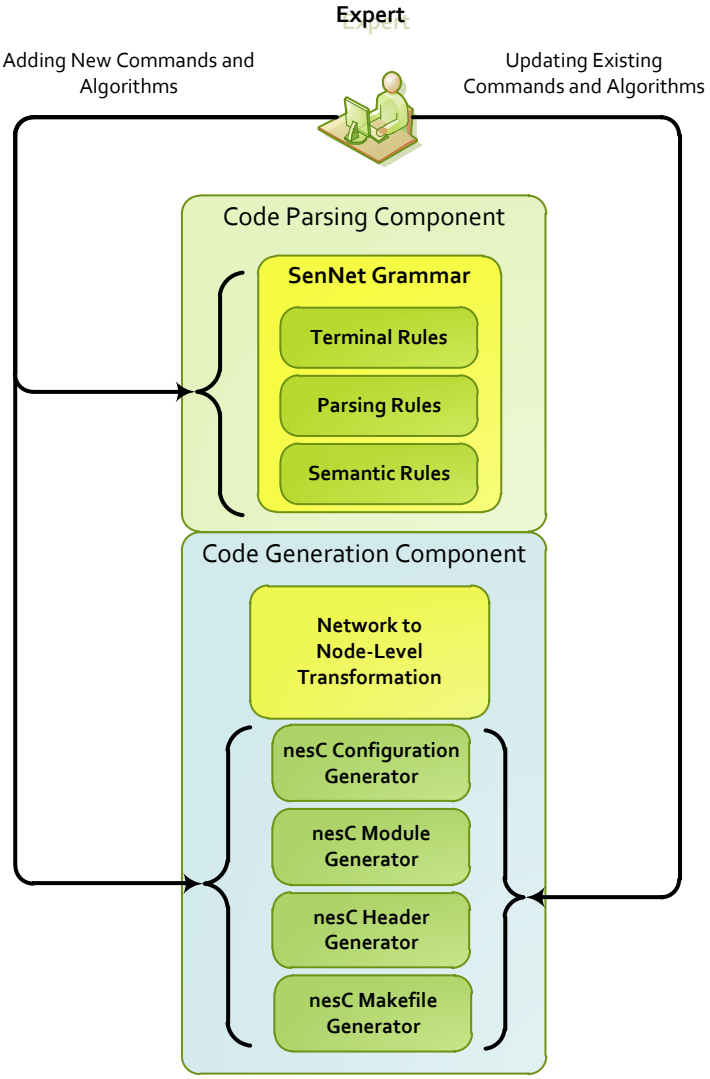

Figure 4-8: SenNet Updating and Expanding Process

4.6 Chapter Summary

In this chapter, we have described the internal view – SenNet's internal working mechanism – by explaining how the CPC component parses and analyses the SenNet application code to produce the Abstract Syntax Tree (AST), which helps the CGC component to generate the required nesC code for the required application. The CGC component should first process the AST by converting the network-level tasks and activities to node-level tasks and activities, before triggering four different generators to generate the required nesC code files. Finally, the SenNet extensibility and updating process was explained.

Chapter 5: SenNet Language - Programming View

5.1 Introduction

A DSL is a programming language that is developed to offer a notation for a specific domain based on the concepts and features of that domain without the need for detailed knowledge of that domain (Kosar, Martinez López, Barrientos, & Mernik, 2008). The objective of proposing the SenNet language is to offer a new abstracted programming language that hides the complexity of low-level programming that is related to the WSN domain from the developer's point of view. This chapter will firstly deal with a general overview of the SenNet language, followed by describing its main components. Next, it will identify the steps that should be followed to develop SenNet applications successfully, and explain the general SenNet application structure and commands.

5.2 SenNet Language

SenNet[15] (**Sen**sor **Net**work) is an open-source domain-specific language (DSL) that is designed for Wireless Sensor Network (WSN) application development; it helps developers to focus on the application logic rather than programming complexity or technology low-level details. SenNet has been designed to work with the TinyOS embedded operating system. Figure 5-1 shows SenNet's general working mechanism and how it interrelates to TinyOS.

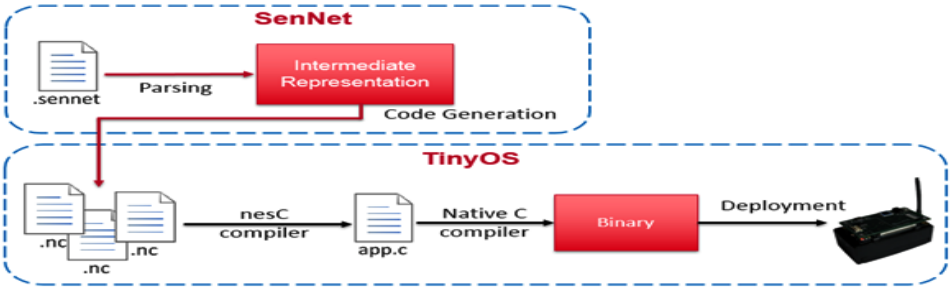

Figure 5-1: General WSN Application Development Steps Using SenNet

[15] SenNet Official Website: http://lzd987.edu.csesalford.com/index.html

Figure 5-2 and 5-3 explain the general application development stages and code generation using SenNet. The developer should use SenNet abstracted commands to express the necessary application requirements. These commands are first parsed and converted into an intermediate representation that encompasses the application logic model, and is called Abstract Syntax Tree (AST). Then the code generation stage produces the right nesC code for each of the sensor nodes that comprise the application. SenNet gives developers the ability to program the required application using a node-level and network-level programming approach.

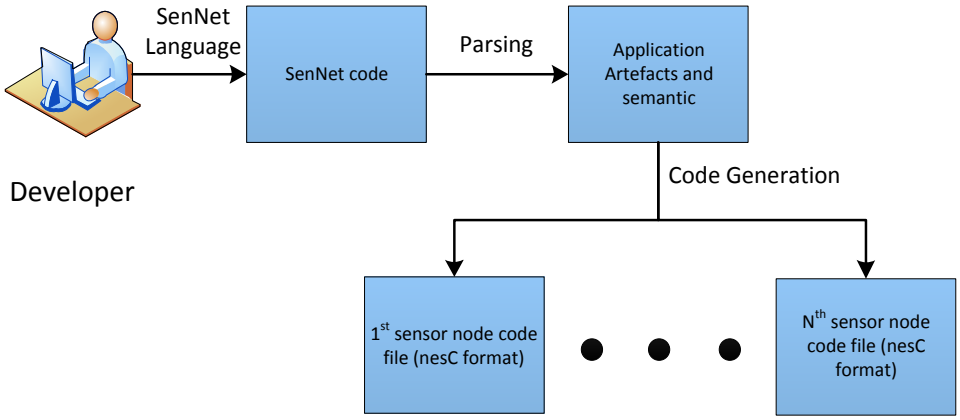

Figure 5-2: SenNet Working Mechanism

SenNet provides developers with many features that are depicted in Figure 5-3.

- **Open Source:** which means that SenNet is free to use, distribute and modify, besides being multi-platform and easy to deploy.
- **Macro-Programming technique:** where a sequence of programming instructions is offered to the developers as a single instruction, as many of the SenNet commands are built according to macro-programming techniques.
- **Multi Editor Views:** SenNet offers two types of editor view, the first editor view is a text-based editor which is used in most programming language editors, while the second editor view is tree-based, which is similar to visual programming.
- **Multi-Programming Abstraction-Level:** Multi-Abstraction level is provided by SenNet, as it enables the developer to develop node or network-level application, in the network-level application, SenNet gives developers the ability to model their application logic in terms of a sensor node, group of nodes, or network-level.

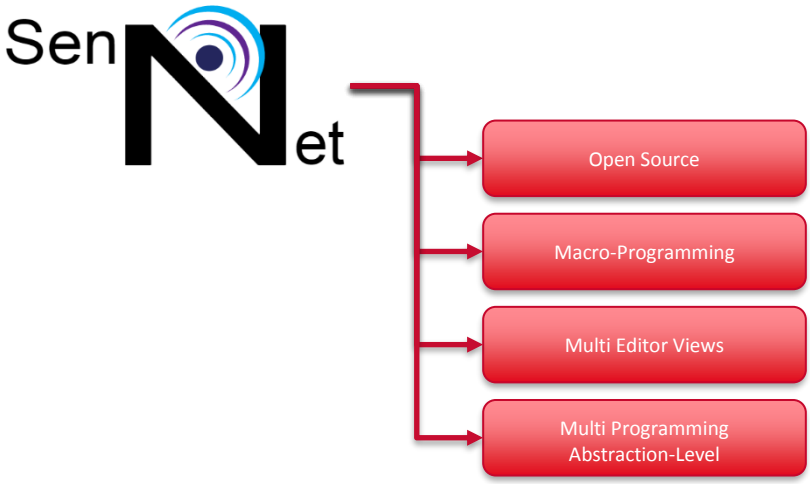

Figure 5-3: SenNet's Main Features

5.3 SenNet Components

SenNet includes two main components, Code Parsing Component (CPC), and Code Generation Component (CGC). Detailed descriptions of these two components are listed below.

5.3.1 Code Parsing Component (CPC)

From a developer's point of view, the CPC component provides two different editor views, as shown in Figure 5-4. The first editor view is text-based, while the second view is a tree-based editor view. Additionally, SenNet editor views offer a number of Integrated Development Environment (IDE) functionalities to be used by a developer to facilitate the developing process, such as immediate feedback, incremental syntax check auto-completion and suggested corrections, as shown in Figure 5-5, which shows how the IDE helps the developer in identifying errors and highlighting solutions. These IDE functions simplify the application programming task and shorten the SenNet learning process

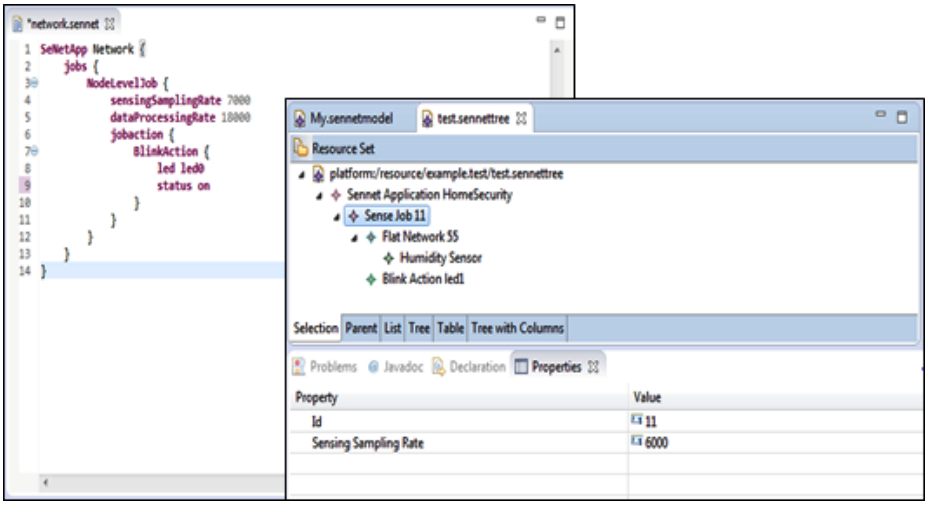

Figure 5-4: SenNet Editor Views

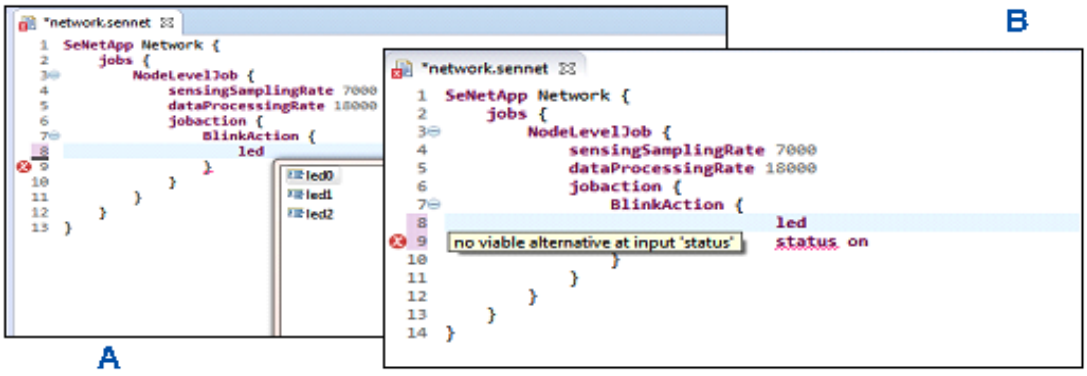

Figure 5-5: SenNet IDE Features (A) IDE Language Helper, (B) IDE Error Message Example

5.3.2 Code Generation Component (CGC)

The CGC component's main function is to generate the necessary nesC code as the final output of the SenNet application, a complete set of nesC code files is generated for each sensor node configured in the SenNet application, as shown in Figure 5-6, which includes a simple SenNet application for a small network that includes three nodes, and how the nesC files related to these three nodes are generated.

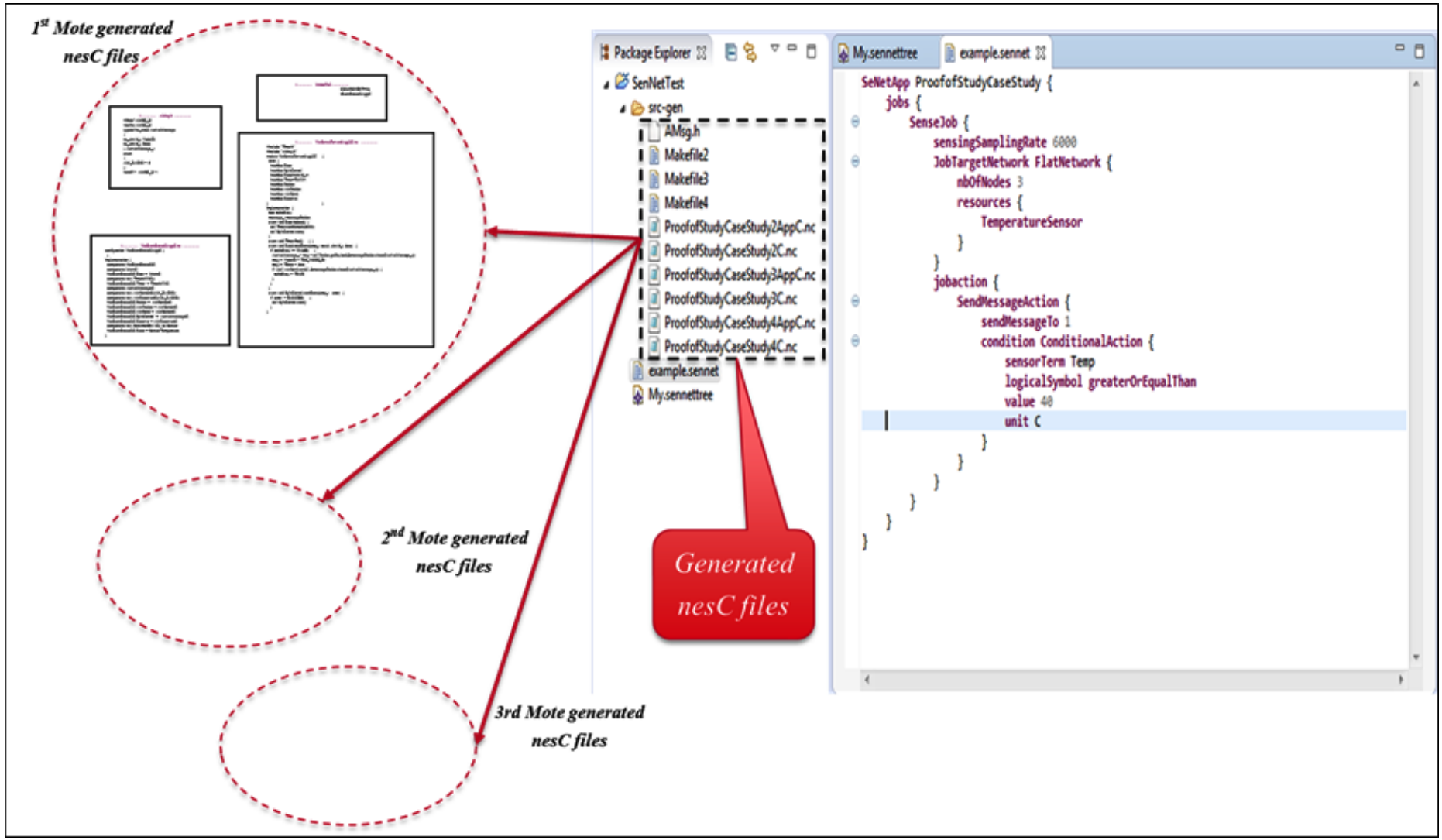

Figure 5-6: nesC Generated Files for SenNet Application

5.4 SenNet Application General Syntax and Development Steps

The general SenNet application structure can be shown in Listing 5-1, which starts with the SenNetApp command that is considered the starting point of the SenNet application, and then the *Application_Name* which is the application name that should be set by the developer. This is followed by the Jobs command that represents the main container that will include all different job types that the developer could define in the SenNet application; id is an optional parameter that can be set by the developer to assign an application id. After that, the developer would start defining the required jobs, as SenNet applications can include one job or more according to the application requirements.

Listing 5-1: General SenNet Application Syntax

```
SenNetApp Application_Name
{
      [id app_id]
      Jobs
      {
          Job1 [, Job2, Job3, ….]
      }
}
```

SenNet language is built and initialised according to the SenNet meta-model (Salman & Al-Yasiri, 2016a), which represents the rules that the developer should follow when using SenNet. The SenNet meta-model has been developed in accordance with ISO/IEC29182: SNRA (ISO/IEC, 2014a), which is considered to be a definitive source of WSN knowledge. Accordingly, SenNet job programming steps are designed to conform to the activity model defined by ISO/IEC29182: SNRA, which is illustrated in Figure 3-1 in the previous chapter; Figure 5-7 below explains each job structure within the SenNet application.

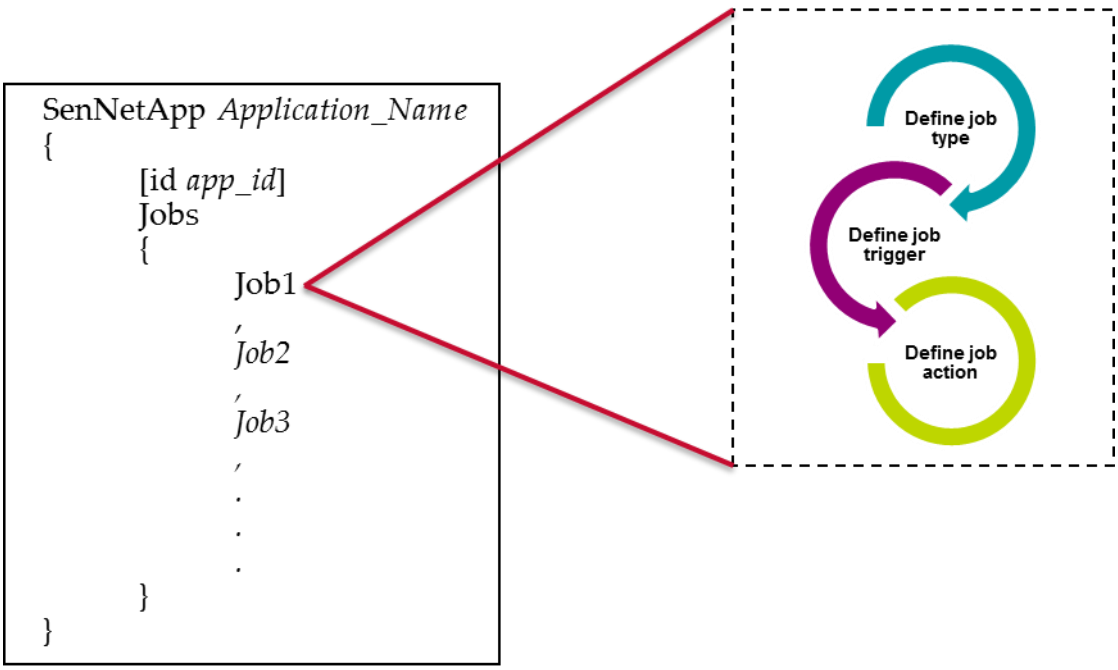

Figure 5-7: General SenNet Application and Job Structure

To program a specific job using SenNet, developers should follow the following steps shown in Figure 5-8:

- Define the required jobs to be performed by the network elements, which could be one job or more.
- Configure the network elements that jobs will apply to; network elements could be a sensor node or a network, and network elements configuration may include network name, the number of sensor nodes, declaring the group of nodes and assigning nodes to these groups.
- Assign the resources, where 'resources' represents the sensors that will be utilised throughout the job, such as sensor type.
- Define the type of triggers that could be used to start or finish the job; this operation is optional for the developers to use.
- Define the job action, which represents the action that will be performed by the network elements after finishing the job, such as sending a message to the sink, or blinking a led; this operation is optional for the developers to use.

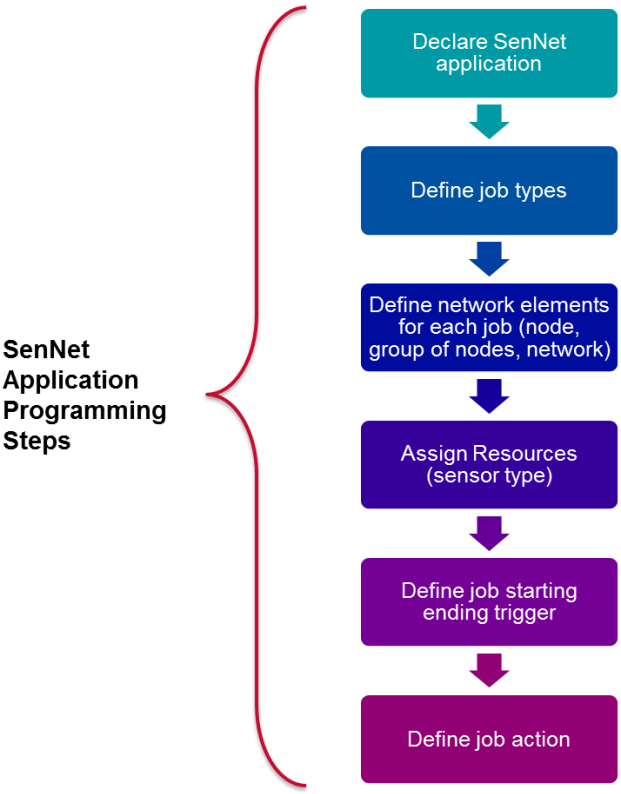

Figure 5-8: SenNet Job Development Steps

5.4.1 Job Types

Developers can configure many types of tasks such as sense-forward, event-triggered, or data processing. Developers can develop these tasks by selecting the right StartEndingJobTrigger and Action commands. The current version of SenNet includes a set of job types such as *SenseJob*, *SenseNowJob*, *NodeDataProcessing* and *NetworkDataprocessing*. Some of these tasks relate to a direct sensing job, while the others are related to data processing. Figure 5-9 represents the SenNet meta-model part that is related to job types.

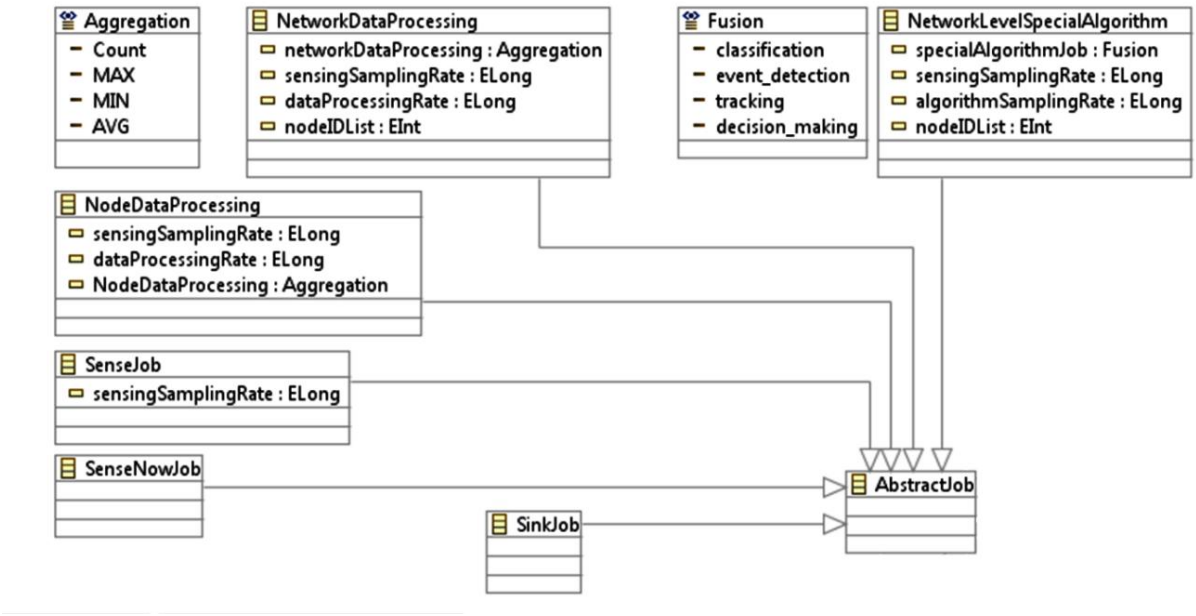

Figure 5-9: Job Types SenNet Meta-Model

- **SenseJob** is a job type that is offered for regular periodic sensing jobs described in Listing 5-2; *id* parameter is optional for the developer to set. The *sensingSamplingRate* parameter is the sampling frequency that is set by the developer according to the application requirements; this parameter should be set in millisecond format, for example 1 second = 1000 milliseconds. Furthermore, the developer should define the network element that will do the job. Developers have two options to choose either node-level by selecting *JobTargetNode* or network-level by selecting *JobTargetNetwork*. The *start/EndTrigger* is optional for the developers to configure if they need the job to be started or ended by a specific trigger, for example, initiating the job if the push button is pressed or by receiving a message from the Sink. The *jobAction* is optional for the developer to use, so it enables the developer to configure an action after the sensing job is finished, such as blinking a LED or forwarding the results to the Sink.

Listing 5-2: SenseJob Command General Syntax

```
Job SenseJob
{
        [id Job_id]
        sensingSamplingRate Frequesncy_in_Millisecond_Format
        NetworkElements JobTargetNode | JobTargetNetwork
        [start/EndTrigger]
        [jobAction]
}
```

- **SenseNowJob** command, which is described in Listing 5-3, is the command that is used by the developer for a one-off sensing job; the differences in internal structure and working mechanism between SenseJob and SenseNowJob have been described in section **3.4.1**. The *id* is an optional parameter to be set by the developer. Moreover, the developer should set it, as this job will be done through node or network-level, *start/EndTrigger* and *jobAction* are optional to the developer to set according to the application preferences.

Listing 5-3: SenseNowJob Command General Syntax

```
Job SenseNowJob
{
      [id Job_id]
      NetworkElements JobTargetNode | JobTargetNetwork
      [start/EndTrigger]
      [jobAction]
}
```

- **NodeDataProcessing** is a node-level data processing job command that any sensor node can apply to its local sensed data. All sensor nodes defined within this job will run a specific aggregation function to process their sensed data, which is described in Listing 5-4. Firstly, the developer should choose the necessary aggregation function from a set of offered functions through the *nodeDataProcessing* parameter. An illustration of the current functions that are available are maximum, minimum, and average. Then the developer should set up two parameters:

    1. ***dataProcessingRate***: this parameter represents the time that the sensor node runs the aggregation function each time. For example, a developer could configure a sensor node to sense the environment temperature every one second, and run max aggregation function every 30 seconds to find the maximum temperature data.

2. ***sensingSamplingRate:*** this parameter represents the time-frequency that sensor node should run to initiate the sensing operation. For example, a developer can configure a sensor node to sense the environment temperature every one second.

After setting the above two parameters, the developer should set the network elements that will run this job. In addition, the developer can set the jobAction and start/EndTrigger if needed, according to the application requirements.

Listing 5-4: NodeDataProcessing Command General Syntax

```
Job NodeLevelJob
{
    [id Job_id]
    nodeDataProcessing max | min | avg | count …
    sensingSamplingRate Frequency_in_Millisecond_Format
    dataProcessingRate Frequency_in_Millisecond_Format
    NetworkElements: JobTargetNode
    [start/EndTrigger]
    [jobAction]
}
```

- **NetworkDataProcessing** job type command is expressed in Listing 5-5. This command is used for network-level data aggregation, which means configuring a special network element to process other nodes' data. This type of job should be assigned to a special network element such as Sink, ClusterHead or ComputationNode. The developer should firstly set the aggregation function required, such as maximum, minimum, average, and count, before setting the *sensingSamplingRate* and *dataProcessingRate*, followed by assigning the network elements that will do the job. Finally, the *start/EndTrigger* and *jobAction* are set according to the application requirements.

Listing 5-5: NetworkDataProcessing Command General Syntax

```
Job NetworkDataProcessing
{
    [id Job_id]
    networkDataProcessing max | min | avg | count …
    sensingSamplingRate Frequency_in_Millisecond_Format
    dataProcessingRate Frequency_in_Millisecond_Format
    [nodeIDList]
    JobTargetNode ComputationNode | Sink | ClusterHead
    [start/EndTrigger]
    [jobAction]
}
```

5.4.2 Network Elements

Figure 5-10 represents the network elements of the SenNet meta-model, which is the part that gives developers the flexibility to configure the required network elements (node or network-level) that will do the job, whether at node-level or network-level.

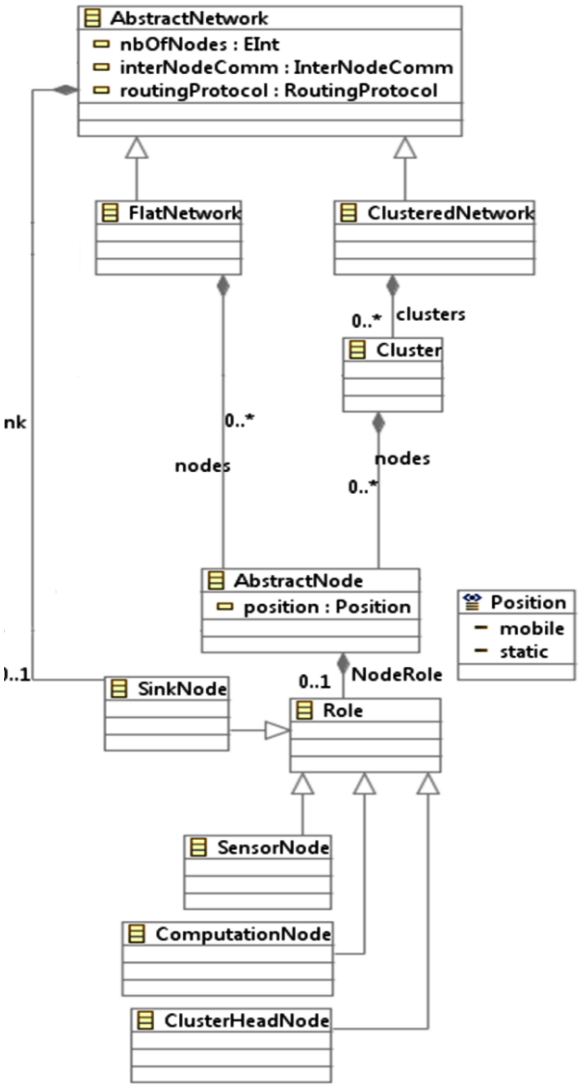

Figure 5-10: Network Elements SenNet Meta-Model

- **JobTargetNode** is the command that can be used by the developer to define a node-level network element; the general command syntax is illustrated in Listing 5-6. SenNet offers some node roles to be used by the developer, such as *SinkNode*, *ComputationNode*, *SensorNode*, and *ClusterHeadNode*.
    1. *SesnorNode* is a regular sensor node with regular sensor node capabilities. This node type can be used within a flat network and clustered network as a cluster node.

2. *SinkNode* is the central node, where all sensor nodes in the network send their sensed data.
3. *ClusterHeadNode* this type of node is used within clustered networks only.
4. *ComputationNode* is a node configuration role that can be assigned to any node in the FlatNetwork to do more activities related to data processing; this role can be considered as a ClusterHeadNode in FlatNetwork.

After choosing the required node role, the developer can optionally set the node id, and set the necessary resources that will be utilised through the sense or processing job, which means deciding which sensor to use, or the developer can choose more than one sensor.

Listing 5-6: JobTargetNode Command General Syntax

```
JobTargetNode [SensorNode | ComputationNode | SinkNode | ClusterHeadNode] {
    [id Node_ID ]
    [nodeResources { AccelerometerSensor | LightSensor | LocationSensor | MicrophoneSensor |
    TemperatureSensor | PressureSensor | HumiditySensor | VoltageSensor } ]
}
```

- **JobTargetNetwork** is the command that developers can use to define the network-level elements; the general command syntax is expressed in Listing 5-7. Developers should first set the *nbOfNodes* parameter with the number of nodes that require to be configured. The developer can optionally set the *RoutingProtocol*, the current SenNet version support Collection Tree Protocol (CTP), Dissemination Protocol, and Active Message Protocol, as these are the official communication protocols that are currently adopted and supported by TinyOS. If the developer does not set the *RoutingProtocol*, then SenNet will use Active Message Protocol by default. Finally, developers should assign the necessary resources, meaning the sensor type that is necessary to achieve the job successfully.

Listing 5-7: JobTargerNetwork Command General Syntax

```
JobTargetNetwork [FlatNetwork | ClusterNetwork] {
    [id Node_ID ]
    nbOfNodes Number-of-Nodes
    [routingProtocol RoutingProtocol ]
    [Resources { AccelerometerSensor | LightSensor | LocationSensor | MicrophoneSensor |
    TemperatureSensor | PressureSensor | HumiditySensor | VoltageSensor } ]
}
```

5.4.3 Job Start/Ending Trigger

Figure 5-11 shows the SenNet meta-model that is related to start/end trigger, and the **start/EndTrigger** command syntax is described in Listing 5-8. This command is used by developers to configure the trigger that will start the job or end it. Firstly, the developer should set this command either for starting a job or ending it, then choose the trigger type; SenNet offers the developers three trigger types for this case, *PushButtonTrigger*, *ReceiveSerialMsgTrigger*, and *ReceiveMessageTrigger*. A simple case study to use this command, such as pressing the push button of the sensor node, will be the trigger to initiate (start) the sensing job for a node.

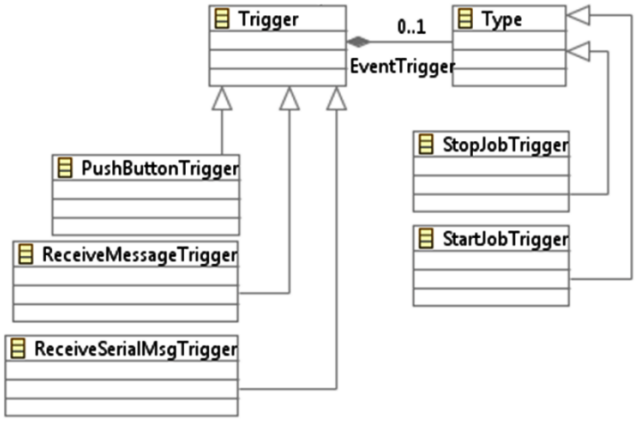

Figure 5-11: StartEndJob SenNet Meta-Model

Listing 5-8: start/EndTrigger Command General Syntax

```
Start/EndTrigget {
      [StartJobTrigger: {ReceiveMessageTrigger | ReceiveSerialMsgTrigger | PushButtonTrigger} ]
      [StopJobTrigger: {ReceiveMessageTrigger | ReceiveSerialMsgTrigger | PushButtonTrigger} ]
}
```

5.4.4 Job Action

The **jobAction** is a command option available to the developer to configure the network elements' behaviour after completing the job. Figure 5-12 represents the SenNet job action meta-model, and Listing 5-9 describes the general *jobAction* command syntax. The developer has many options to choose as job action, such as send message action, and blink action. In addition, the developer can configure the action to be done according to a specific condition,

for example, sending a message to the sink (node_id = 1) if the temperature is greater than or equal to 20 C°. The general conditional action command syntax is illustrated in Listing 5-10.

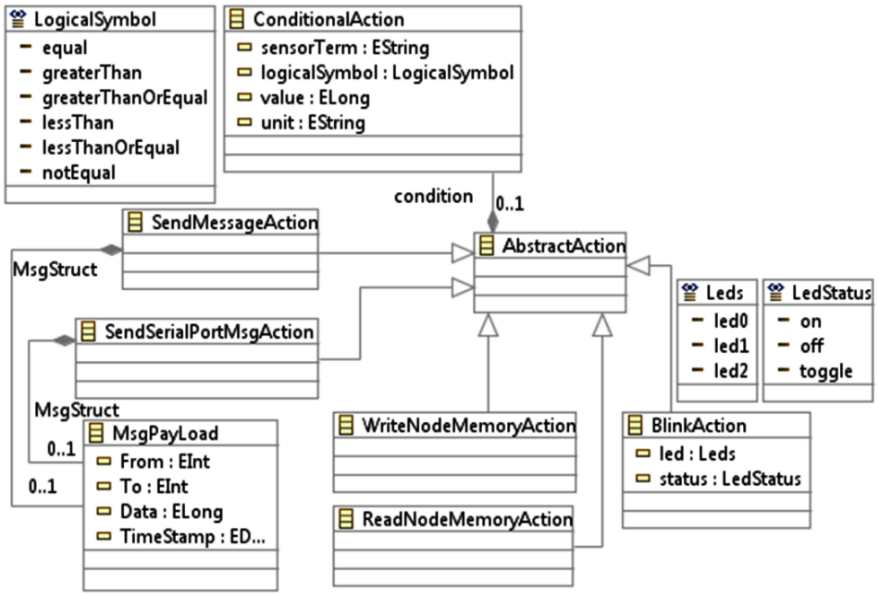

Figure 5-12: JobAction SenNet Meta-Model

Listing 5-9: jobAction Command General Syntax

```
jobAction {
    [ReadNodeMemoryAction | WriteNodeMemoryAction | SendMessageAction | BlinkAction |
    SendSerialPortMsgAction] {
    [sendMessageTo Node_ID]
    [led led0 | led1 | led2]
    [status on | off | toggle]
    [condition conditionalAction]
    }
}
```

Listing 5-10: Conditional Action Command Syntax

```
ConditionalAction
{
    LogicalSymbol Equal | greaterThan | greaterThanOrEqual | lessThan | lessThanOrEqual |
    notEqual
    Value Node_ID
}
```

5.5 Chapter Summary

In this chapter, we have described the SenNet language from the developer's point of view, starting by explaining the importance of using TinyOS as a foundation for SenNet, because TinyOS enables SenNet to be used by developers to develop WSN applications for a broad range of sensor node types.

SenNet includes two main components: Code Parsing Component (CPC) and Code Generation Component (CGC). CPC is responsible for providing developers with the required IDE functionalities, such as error messages and language helper; while CGC is responsible for generating the required TinyOS application. SenNet application general syntax and development steps are discussed in section **5.4**.

Chapter 6: SenNet Evaluation

6.1 Introduction

This chapter will discuss the validation and evaluation phase to assess SenNet language functionality. Firstly, to prove SenNet usability and code functionality, it will describe a SenNet application developed for a simple and representative case study and validate the final generated nesC code. Next, an evaluation plan is outlined using Goal/Question/Metric (GQM) approach (Basili, 1992), which is considered one of the popular measurement approaches to ensure the validity and integrity of the measurement results. GQM consider a *de facto* measurement framework. Where many companies and organisations have employed, such as Philips and NASA (Hussain & Ferneley, 2008).

Therefore, a set of questions are identified based on the evaluation goals that represented by usability and functional suitability. These questions should be answered through the evaluation phase. A set of possible metrics is outlined that can be useful to answer the mentioned questions. These metrics were calculated using multiple methods such as conducting a user survey, implementing a real business case studies, and others. Finally, we outline a summary of the final metric results, that lead to the final answers for the proposed questions.

6.2 Proof of Concept Case Study

WSN supports a vast range of real-world applications, such as habitat monitoring (Mainwaring et al., 2002), active volcano monitoring (Werner-Allen et al., 2006) and beaches monitoring (Alkandari, Alnasheet, Alabduljader, & Moein, 2012). For case study purposes, using a small but representative scenario can be enough to demonstrate the power and functionality of SenNet. So as a representative case study, a simple home temperature monitoring scenario has been chosen (Salman & Al-Yasiri, 2016a). This simple scenario can be described as follows:

- Temperature monitoring for three home locations
- The monitoring process should be every 1 minute.
- If the sensed temperature in any of these three places is greater than 40 C°, then a message should be sent to the base station.

Developing this scenario using SenNet is a simple task. Monitoring three different locations means three sensor nodes are needed to do the same monitoring job. Accordingly, Listing 6-1 represents the SenNet application developed for this simple scenario, which first defines a sensing job with a sampling rate at 1 minute = 60000 milliseconds. This sensing job is assigned to a network that includes three sensor nodes (network-level programming concept). Next, conditional job action is defined, which instructs that if the sensing temperature becomes greater than 40 C°, then each node in the network will send a message to the sink node (sink node ID = 1) using ActiveMessage Protocol (AM). Figure 6-1 and 6-2, show the development of this application using SenNet editor views.

After completing the programming process, the required nesC single-node files are generated for three sensor nodes, as shown in Figure 6-1. The generated files are titles with the same application name that was configured at the beginning of the SenNet application "ProofofStudyCaseStudy", and numbered automatically with 2, 3, and 4 that represent the node-id (number 1 is reserved for the sink node). The last file is called AMsg.h, which includes the message structure that should be followed by all sensor nodes when sending a message. Listing 6-2, 6-3, 6-4, 6-5 shows sample of the generated nesC code. Below is a list of the generated files:

- ProofofStudyCaseStudy2AppC.nc
- ProofofStudyCaseStudy2C.nc
- Makefile2
- ProofofStudyCaseStudy3AppC.nc
- ProofofStudyCaseStudy3C.nc
- Makefile3
- ProofofStudyCaseStudy4AppC.nc
- ProofofStudyCaseStudy4C.nc
- Makefile4
- AMsg.h

Listing 6-1: SenNet Application for Case Study Scenario

```
**********************************************************************
SeNetApp ProofofStudyCaseStudy {
	jobs {
		SenseJob {
			sensingSamplingRate 60000
			JobTargetNetwork FlatNetwork {
				nbOfNodes 3
				resources {
					TemperatureSensor
				}
			}
			jobaction {
				SendMessageAction {
					sendMessageTo 1
					condition ConditionalAction {
						sensorTerm Temp
						logicalSymbol greaterOrEqualThan
						value 40
						unit C
					}
				}
			}
		}
	}
}
**********************************************************************
```

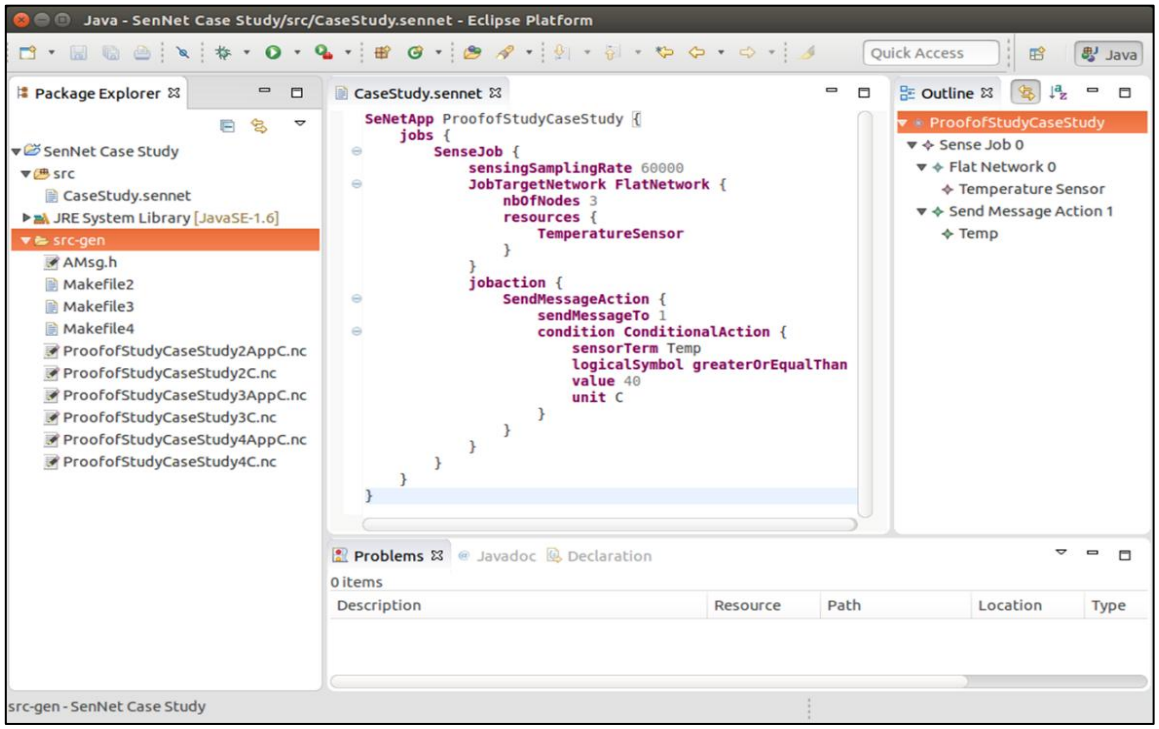

Figure 6-1: Case Study Development Using Text-Based Editor View

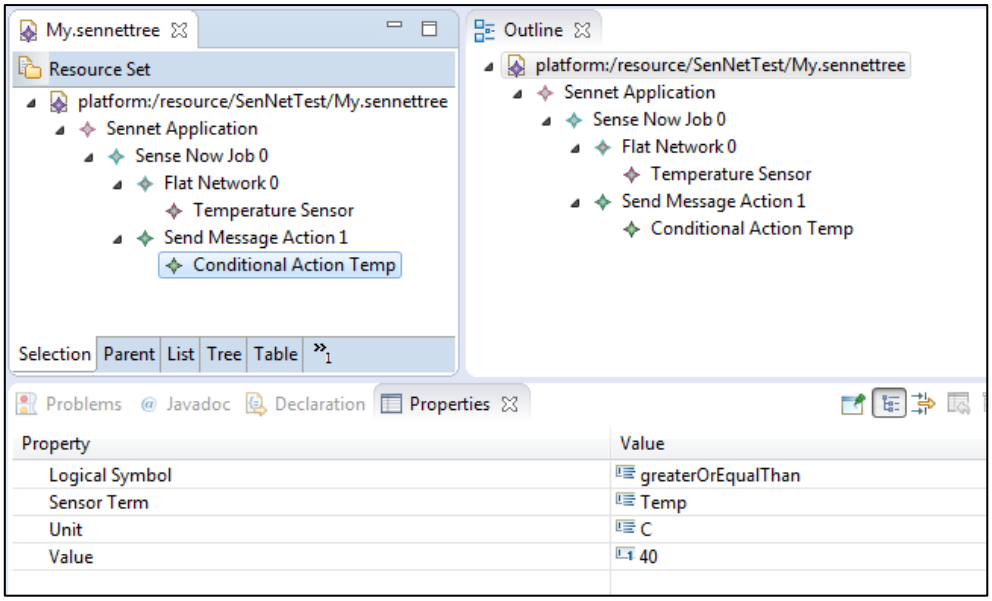

Figure 6-2: Case Study Development Using Tree-Based Editor View

To prove the generated code functionality, two steps were implemented. The first step was to compile the generated files using TinyOS to find if there were any errors or warnings. Figure 6-3 illustrates the TinyOS component graph, which shows that all required components are linked correctly and without any errors.

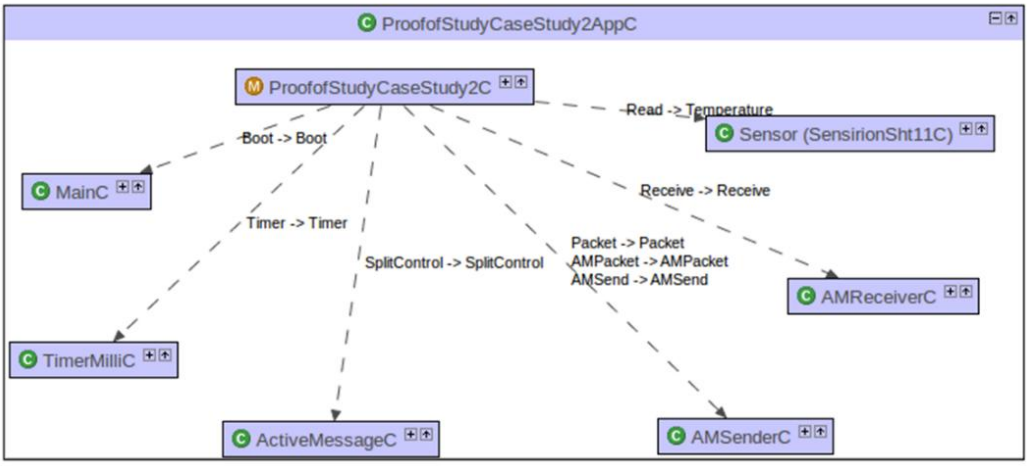

Figure 6-3: TinyOS Component Graph for A Sample of The Generated nesC Files

The second step was to deploy these files into three real sensor nodes. Three IRIS XM2110 sensor nodes (motes) were equipped with an MTS-400 Sensor board. The MIB520 programming board was used to deploy these files to the mentioned sensor nodes using make command; the deployment process was done successfully, and each sensor node worked according to the deployed application.

Figure 6-4: The Three IRIS-XM2110 Used to Validate the Generated Files Code

Listing 6-2: Sample of the nesC Configuration Files

```
**************************************************************************
configuration ProofofStudyCaseStudy2AppC
{
}
implementation {
  components ProofofStudyCaseStudy2C;
  components MainC;
  ProofofStudyCaseStudy2C.Boot -> MainC;
  components new TimerMilliC();
  ProofofStudyCaseStudy2C.Timer -> TimerMilliC;
  components ActiveMessageC;
  components new AMSenderC(AM_RADIO);
  components new AMReceiverC(AM_RADIO);
  ProofofStudyCaseStudy2C.Packet -> AMSenderC;
  ProofofStudyCaseStudy2C.AMPacket -> AMSenderC;
  ProofofStudyCaseStudy2C.AMSend -> AMSenderC;
  ProofofStudyCaseStudy2C.SplitControl -> ActiveMessageC;
  ProofofStudyCaseStudy2C.Receive -> AMReceiverC;
  components new SensirionSht11C() as Sensor;
  ProofofStudyCaseStudy2C.Read -> Sensor.Temperature;
}
**************************************************************************
```

Listing 6-3: Sample of the nesC Module Files

```
**************************************************************************
   #include "Timer.h"
   #include "AMsg.h"
   module ProofofStudyCaseStudy2C
   {
     uses {
       interface Boot;
       interface SplitControl;
       interface Read<uint16_t>;
       interface Timer<TMilli>;
       interface Packet;
       interface AMPacket;
       interface AMSend;
       interface Receive;
     }
   }
```

```
  implementation {
    bool radioBusy;
    message_t messagePacket;
    event void Boot.booted()
    {
      call Timer.startPeriodic(60000);
      call SplitControl.start();
    }
    event void Timer.fired()
    {
      call Read.read() ;
    }
    event void Read.readDone(error_t result, uint16_t data)
    {
      if (data  >= 40)
      {
        if (radioBusy == FALSE)
        {
          ActiveMessage_t* msg = call
Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
          msg -> NodeID = TOS_NODE_ID;
          msg -> TData = data;
          if ( call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t)))
          {
            radioBusy = TRUE;
          }
        }
      }
    }
    event void SplitControl.startDone(error_t error)
    {
      if (error != SUCCESS)
      {
        call SplitControl.start();
      }
    }
    event void SplitControl.stopDone(error_t error)
    {
    }
    event void AMSend.sendDone(message_t *msg, error_t error)
    {
      if (msg == & messagePacket)
      {
        radioBusy = FALSE;
      }
    }
    event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len)
    {
      return msg;
    }
}
**********************************************************************
```

Listing 6-4: AMsg.h Header File

```
**********************************************************************
  #ifndef AMSG_H
  #define AMSG_H
  typedef nx_struct ActiveMessage
  {
  nx_uint16_t NodeID;
```

```
   nx_uint16_t Data;
   } ActiveMessage_t;
   enum
   {
   AM_RADIO = 6
   };
#endif /* AMSG_H */
**********************************************************************
```

Listing 6-5: Sample of the nesC Makefile

```
**********************************************************************
COMPONENT=ProofofStudyCaseStudy2AppC
Include $(MAKERULES)
**********************************************************************
```

6.3 SenNet Evaluation Plan

Every programming language has its strengths and weak points. However, the reason for choosing a particular programming language may be based on factors that are not related to technical features and characteristics of this language (Naiditch, 1999).

Accordingly, we have tried to evaluate SenNet in terms of functionality that is related to the aim that is designed for, and in terms of language usability that focuses on user perception. Therefore, GQM approach has been used to prepare an evaluation plan for that purpose. The goals of the evaluation plan can be summarised as follows:

- **Usability:** this goal factor is focused on the language usability from a developer's viewpoint, especially the following two aspects:
    1. Can SenNet help developers to develop WSN applications and minimise development efforts?
    2. Can Developers learn and use SenNet?
- **Functional Suitability:** this goal is focused on the WSN domain, and whether SenNet can be used to express WSN domain scenarios and concepts.

According to the above-defined goals, three questions are identified; Questions 1 and 2 are assessing the Usability goal, while Question 3 is trying to determine whether SenNet is functionally suitable to develop WSN applications.

Q1. Can SenNet hide programming complexity and facilitate the development process?
Q2. Can developers learn and use SenNet language easily and rapidly?
Q3. Is SenNet suitable to explain WSN concepts and cover most domain scenarios?

Figure 6-5 and Table 6-1 outline the evaluation plan used to assess SenNet language.

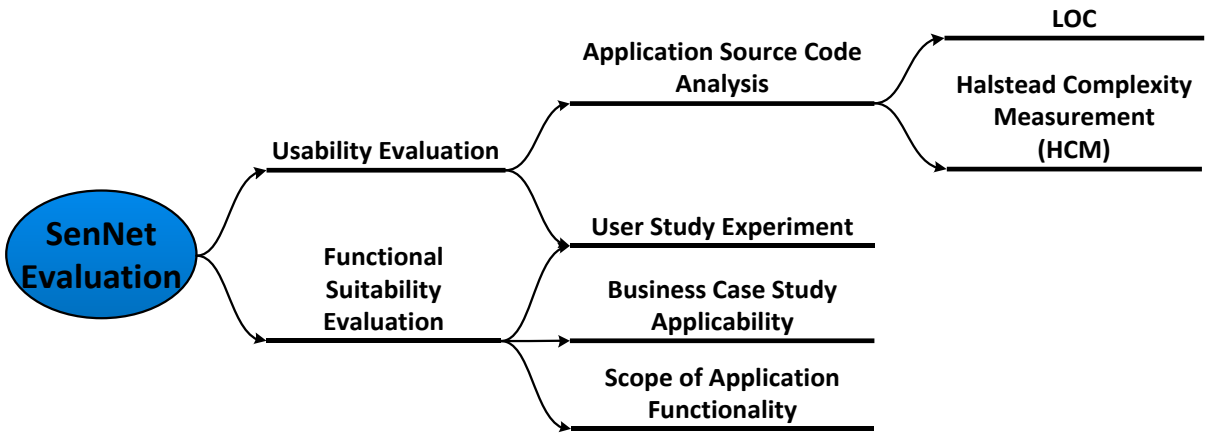

Figure 6-5: SenNet Evaluation Plan

Table 6-1: General SenNet Evaluation Plan

Evaluation Goals	Evaluation Questions	Evaluation Method	Metric
Usability	Q1: Can SenNet hide programming complexity and facilitate the development process?	Application Source Code Analysis	• LOC • HCM a. Code Complexity b. Programming effort and time c. No. of vocabulary items used d. Program Length & Volume
		User Study Experiment	• Effectiveness • Efficiency • Programming Technique
	Q2: Can developers learn and use SenNet language easily and rapidly?	User Study Experiment	• Likeability • Learnability • Maintaining Existing Applications • Mind to program mapping
Functional Suitability	Q3: Is SenNet suitable to explain WSN concepts and cover most domain scenarios?	Scope of Application Functionality	The scope of Application Functionality (L. Bai et al., 2009).
		Business Case Study Applicability	• Light Monitoring in Tunnels (Ceriotti et al., 2011) • Temperature Monitoring for Smart Homes (Rodrigues, Delicato, et al., 2015)
		User Study Experiment	• Appropriateness

6.3.1 Languages under Test

SenNet is evaluated against other languages in terms of specific factors and metrics. The languages that will be used in the evaluation will be chosen according to the criteria that suggested by (Bai, Dick, Dinda, et al., 2011):

- Focus on languages that are designed to develop WSN applications.
- It was designed to support applications that carry out periodic data sampling and transmission, whereas some proposed languages offer continuous sensing activities only, or do not support data transmission.
- Compilers, toolchain, and language documentations are publicly available.

6.4 Scope of Application Functionality

Wireless sensor network technology can be used with a broad range of real-world applications, and many classifications have been used to classify and categorise WSN applications in terms of application field, such as environmental monitoring and human health monitoring. In contrast, some other classifications are made in terms of the functionalities required from the sensor platform, such as the ability to initiate data transmission and data aggregation. Bai et al. (2009) identified eight properties that represent the main application tasks and proposed a new classification for Wireless Sensor and Actuation Network (WSAN) according to these tasks. The classification properties are mobility, data transmission, actuation support, interactivity support, data interpretation, data aggregation, heterogeneity, and sampling.

To assess SenNet capability to cover the domain functionalities and application tasks, then we used Bai classification:

- **Sampling:** SenNet language provides both periodic and event driven jobs where the user can configure the sampling rate. In addition, SenNet enables the target sensor to take one read sample.
- **Data transmission:** All SenNet job types have the ability to transmit data using Active Message (AM), Call Tree Protocol (CTP) or Dissemination Protocol, which are the main TinyOS routing protocols.
- **Actuation support:** The current version of SenNet targets sensing and monitoring jobs and data transmission within the network. The actuation behaviour that is available in this version is LED blinking.
- **Interactivity support:** SenNet empowers developers to design jobs that can be triggered by the users such as receiving a message from the Sink node, or bottom triggering, whether these triggers are used to start or end a job.

- **Data interpretation support:** The current version of SenNet supports data interpretations, where jobs or specific responses can be designed according to a specific data interpretation.
- **Data aggregation support:** SenNet supports node-level and network-level data processing. The current SenNet data processing jobs are limited to simple processing algorithms, but more algorithms can be added.
- **Heterogeneity:** SenNet applications are TinyOS dependent and Hardware Abstraction Architecture (HAA) is one of the TinyOS advantages. Most SenNet tasks and activities are developed using Hardware Independent Layer (HIL). Of course, not all special components can be presented using HIL so further customisation may be needed by the developer.
- **Mobility support:** SenNet language structure and produced semantics support mobile nodes, but not functionality; therefore, localisation and mobile supported routing algorithms should be developed.

In conclusion, SenNet supports 6 out of 8 of WSN application tasks, which means that SenNet can be used to design applications that cover **75%** of the Wireless Sensor and Actuation Network (WSAN) application functionalities. This in turn answers **Q3** in the evaluation plan, that SenNet is appropriate to develop WSN monitoring applications.

Table 6-2: SenNet Assessment According to Bai Classification

Application Task	SenNet
Sampling	✓
Data transmission	✓
Actuation support	x
Interactivity support	✓
Data interpretation support	✓
Data aggregation support	✓
Heterogeneity	✓
Mobility support	x

6.5 Application Source Code Analysis

The source code analysis provides a way to analyse code without having to run the code. The Line of Codes (LOC) and Halstead Complexity Measurement (HCM) are basic and fundamental methods to measure code complexity (Yu & Zhou, 2010). The LOC metric is the count of physical source code without considering the language type; one of the main advantages of the LOC metric is that it can give some suggestions on how easily the code can

be written and estimate the efforts needed by developers to program a specific scenario using two different languages. In the same way, HCM is used to analyse the structure and the characteristics of the program. It calculates code complexity from data stream perspective (operator and operand) and ignores the complexity of the control flow (branch and jumps), which is considered a perfect method to assess WSN applications because they do not usually include many control flow commands. Moreover, both LOC and HCM share some advantages, such as that both methods are easy to calculate, and they are programming language independent (Keshavarz, 2011; Yu & Zhou, 2010).

Many metrics can be extracted using HCM, these metrics and their calculation formulas are (Yu & Zhou, 2010):

- n1 = the number of distinct operators
- n2 = the number of distinct operands
- N1 = the total number of operators
- N2 = the total number of operands from these numbers
- Software length: N = N1+ N2
- Software vocabulary: n = n1 + n2
- Volume: V = N * log2(n)
- Difficulty: D = (n1/ 2) * (N2/n2)
- Programming Effort: E = V * D
- Programming Time: T = E/18

All HCM calculation formula depends on two parameters (operator and operands). The operator is a symbol, character or a set of characters that help users to command the computer to do a specific job, while operands participate in the job.

Four application scenarios have been chosen for this evaluation method. The four applications categorised into single-node based applications and network-based applications are listed below:

- Node-Level Applications
    1. Empty application (Elsts et al., 2013).
    2. Simple Sense-Forward application (Elsts et al., 2013).
- Network-Level Application

1. Sense-Forward.
2. Event Based job.

Within the context of SenNet, the generated nesC code can be considered as an optimal and efficient code to program the required functionalities because of the below reasons:

1. SenNet was designed to produce nesC code based on the best practices that gained from the learning materials and application examples that provided by the nesC official website.
2. The nesC language is an imperative and procedural programming language, for instance, to send a message to the base station, then the developer should first define the message structure and include the required data. Next, checking the radio communication if busy or not, followed by sending the message, the final step is to check if the sending process has been sent properly and without errors. So, the complexity in using nesC is because of the detailed programming steps that required to program the low-level hardware components. Thus, it can be noticed that different developers produce semi-identical solutions to develop a specific functionality.

Therefore, we will compare the SenNet source code results with the results that gained from the generated nesC code.

6.5.1 Node-Level Applications

6.5.1.1 Empty Application

This application sample is an empty application with no user logic, to show the difference between SenNet and nesC structurally. Listing 6-7 and 6-8 shows the necessary implementation code for this scenario using SenNet and nesC respectively, while Table 6-3 shows this scenario LOC and HCM statistics.

Listing 6-6: Empty Application - SenNet Version

```
SeNetApp Empty {

}
```

Listing 6-7: Empty Application - nesC Version

```
********** EmptyAppC.nc ************
configuration SenseAppC {
}
implementation {
}
************ EmptyC.nc *************
module EmptyC {
}
implementation {
}
************ Makefile *************
COMPONENT=EmptyAppC
Include $(MAKERULES)
```

Configuration File

Module File

Table 6-3: Empty Application LOC & HCM Statistics

Empty Application							
LOC and Files Statistics				**HCM Statistics**			
	Files Name	LOC	no. of application files	n1	N1	n2	N2
nesC	EmptyC.nc	2	3	10	14	3	3
	EmptyAppC.nc	2					
	Makefile	2					
	nesC Total	**6**					
SenNet	**Empty.sennet**	**1**	1	2	2	1	1

The detailed HCM operators and operands for this scenario are follows:

- SenNet
    1. Operators (n1 = 2, N1 = 2): SenNetApp, {}.
    2. Operands (n2 = 1, N2 = 1): Empty.
- nesC
    1. Operators (n1= 10, N1=14): Configuration, {}, {}, {}, {}, Implementation, implementation, Module, =, (), Component, $, MAKERULES, include.
    2. Operands (n2=3, N2=3): SenseAppc, EmptyC, EmptyAppC.

6.5.1.2 Sense-Forward Application

This is a simple SF scenario, where a sensor node senses the environment temperature periodically, then sends the sensed data over the network to the Sink node, with a sampling rate of 1 minute (Elsts et al., 2013). Figure 6-6 shows this scenario's implementation using SenNet; the complete SenNet application and the generated nesC code for this scenario can be found in APPENDIX F.1. Table 6-4 shows the basic scenario implementation statistics for nesC and SenNet.

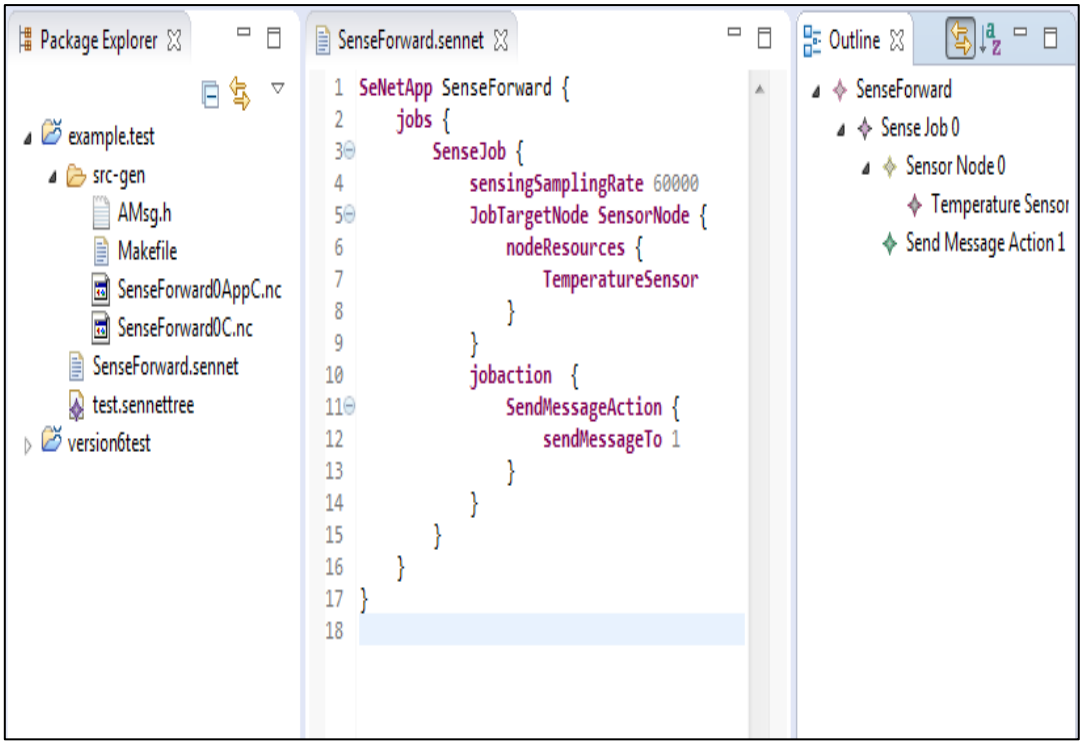

Figure 6-6: Screen Snapshot for SF Scenario Using SenNet

Table 6-4: SenseForward Application Scenario Statistics

SenseForward Application Scenario							
LOC and Files Statistics				HCM Statistics			
	File Names	LOC	no. of application files	n1	N1	n2	N2
nesC	SenseForward0AppC.nc	17	4	50	218	42	111
	SenseForward0C.nc	29					
	Makefile	2					
	AMsg.h	9					
	nesC Total	**57**					
SenNet	**SingleNodeSenseForwardApp.sennet**	**10**	1	12	18	3	3

6.5.2 Network-Level Applications

6.5.2.1 Sense-Forward Application

In this small network, which includes three sensor nodes, a sensing temperature job is assigned to each node which sends the results to the sink with a sampling rate of 1 minute. Figure 6-7 shows this scenario's implementation using SenNet. Moreover, the complete SenNet application and the generated nesC code for this scenario can be found in APPENDIX F.2. Table 6-5 shows the basic scenario implementation statistics for nesC and SenNet.

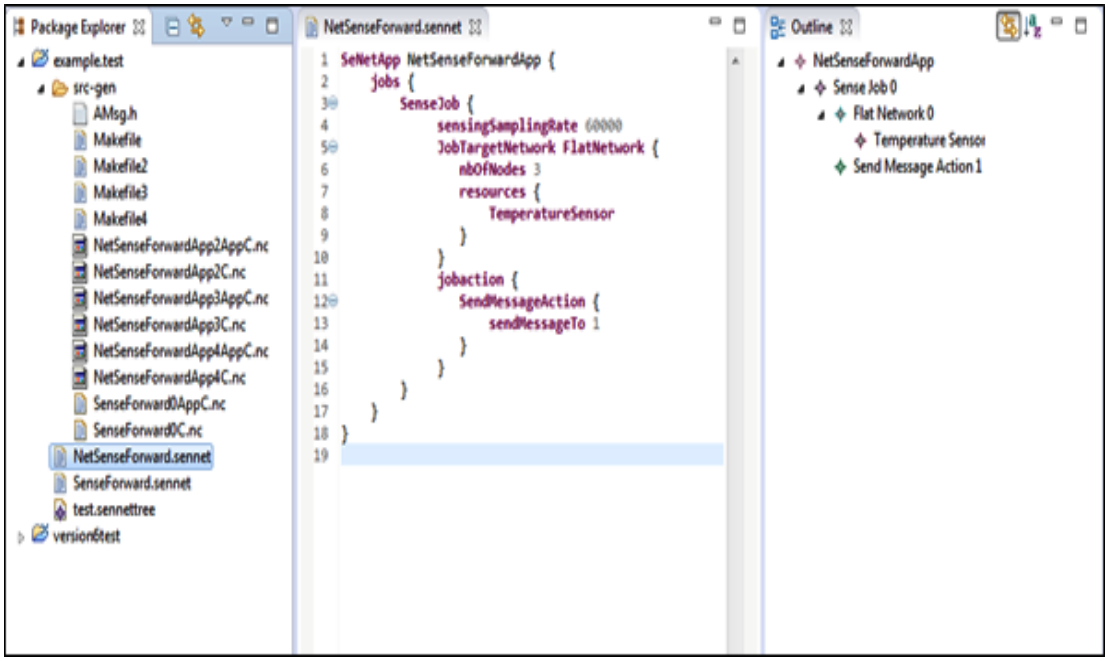

Figure 6-7: Screen Snapshot for Network-Level SenseForward Scenario Using SenNet

Table 6-5: Network-Level SenseForward Application Scenario Statistics

Net SenseForward Application Statistics							
LOC and Files Statistics				**HCM Statistics**			
	File Names	LOC	no. of application files	n1	N1	n2	N2
nesC	NetSenseForwardApp2AppC.nc	17	10	140	645	133	311
	NetSenseForwardApp3AppC.nc	17					
	NetSenseForwardApp4AppC.nc	17					
	NetSenseForwardApp2C.nc	35					
	NetSenseForwardApp3C.nc	35					
	NetSenseForwardApp4C.nc	35					
	Makefile2	2					
	Makefile3	2					
	Makefile4	2					
	AMsg.h	9					
	nesC Total	**171**					
SenNet	**NetSenseForwardApp.sennet**	**11**	1	13	19	4	4

6.5.2.2 Event-Based Application

In a group of 5 sensor nodes, those sensing temperature and sending a notification message to the sink if Temp > 40 C°. Figure 6-8 shows this scenario's implementation using SenNet, and the complete SenNet application and the generated nesC code for this scenario can be found in APPENDIX F.3. Table 6-6 shows the basic scenario implementation statistics for nesC and SenNet.

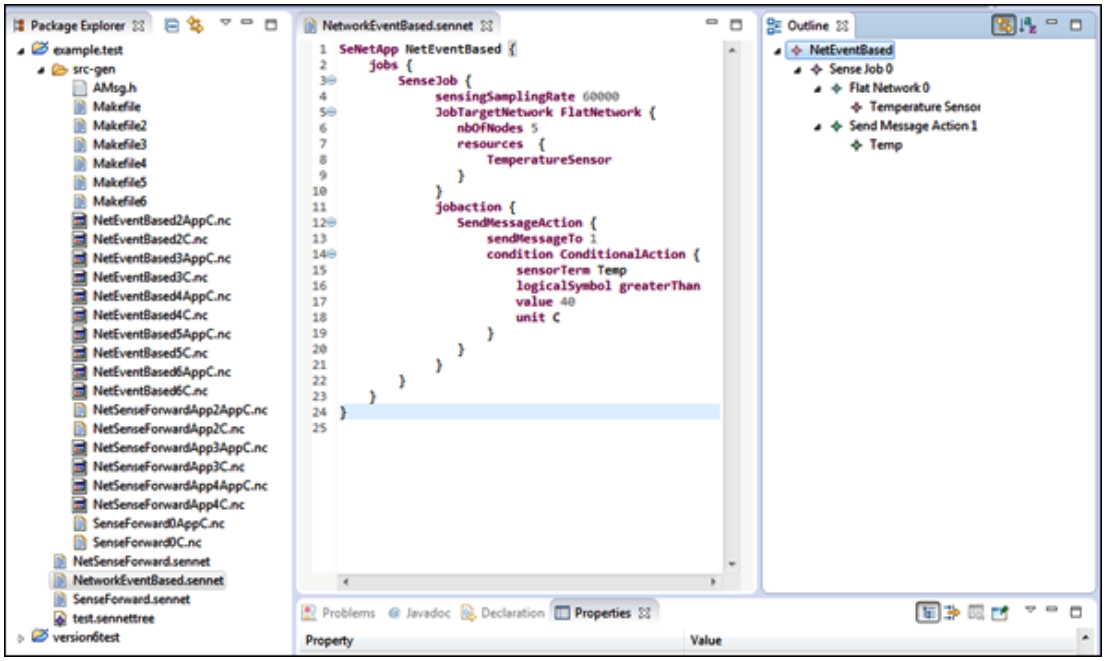

Figure 6-8: Screen Snapshot for Network-Level EventBased Scenario Using SenNet

Table 6-6: Network-Level EventBased Application Scenario Statistics

Network Event-Based Application Statistics							
LOC and Files Statistics				**HCM Statistics**			
	File Names	LOC	no. of application files	n1	N1	n2	N2
nesC	NetEventBased2AppC.nc	17	16	246	1133	217	478
	NetEventBased3AppC.nc	17					
	NetEventBased4AppC.nc	17					
	NetEventBased5AppC.nc	17					
	NetEventBased6AppC.nc	17					
	NetEventBased2C.nc	36					
	NetEventBased3C.nc	36					
	NetEventBased4C.nc	36					
	NetEventBased5C.nc	36					
	NetEventBased6C.nc	36					
	Makefile2	2					
	Makefile3	2					
	Makefile4	2					
	Makefile5	2					
	Makefile6	2					
	AMsg.h	9					
	nesC Total	**284**					
SenNet	**NetSenseForwardApp.sennet**	**16**	1	15	22	12	12

6.5.3 Results Summary

After developing the aforementioned scenarios and analysing the SenNet and nesC code in terms of LOC and HCM methods, the final results can be summarised below.

Regarding counting the code lines, Table 6-7 and Figure 6-10 show a significant improvement in the number of lines of code required when developing an application using SenNet. They also show the real power of SenNet when used to develop a network-level application, especially when the network includes a high number of sensor nodes. As a conclusion, SenNet saves on average **88.4%** of the LOC that developers should write in nesC code.

Table 6-7: LOC Final Statistics

Application Type		nesC		SenNet		
		LOC	files	LOC	files	saving
Node-Level Application	Empty	6	3	1	1	83.3%
	Sense Forward	57	4	10	1	82.5%
Network-Level Application	Sense Forward	171	10	11	1	93.6%
	Event Based	284	16	16	1	94.4%
Average LOC Saving						**88.4%**

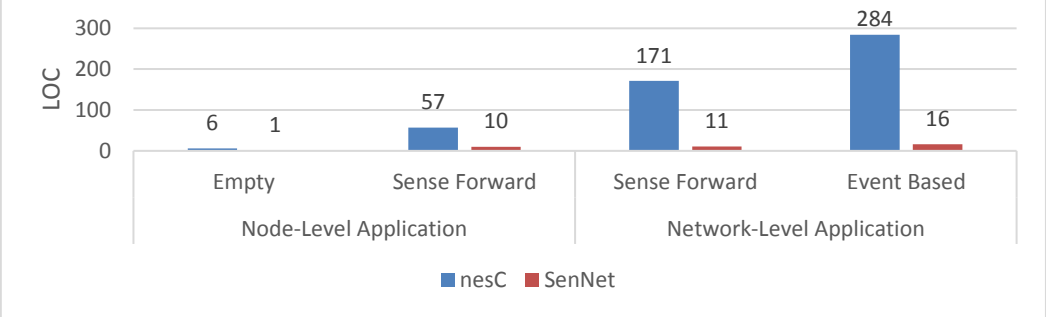

Figure 6-9: nesC and SenNet LOC Statistics

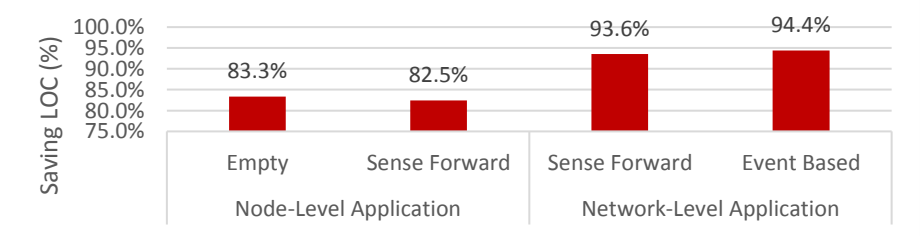

Figure 6-10: SenNet Saving in LOC

One the other hand, **Error! Reference source not found.** shows the detailed and final results for HCM parameters calculated for SenNet and nesC. Table 6-9 indicates that SenNet saves **87.14%**, **92.86%** and **96.64%** in terms of required vocabularies used, application length, and code difficulty respectively. Figure 6-11 illustrated that a SenNet saves **99.89%** of the programming efforts needed to develop a WSN application using nesC.

Table 6-8: HCM Statistics

	Node-Level		Network-Level		
	Empty	Sense-Forward	Sense-Forward	Event-Trigger	
nesC	17	329	956	1611	**Application Length - N**
	13	92	273	463	**Application Vocabulary - n**
	63	2146	7737	14265	**Volume - V**
	8	1388	10885	29397	**Difficulty - D**
	472	2977924	84213716	419353878	**Programming Effort - E**
	26	165440	4678540	23297438	**Programming Time – T (Sec)**
SenNet	3	21	23	34	**Application Length - N**
	3	15	17	27	**Application Vocabulary - n**
	5	82	94	162	**Volume - V**
	1	9	13	45	**Difficulty - D**
	2	738	1222	7275	**Programming Effort - E**
	0	41	68	404	**Programming Time – T (Sec)**

Table 6-9: SenNet Saving in Terms of HCM Formulas

	Node-Level		Network-Level		Average Saving
	Empty	Sense-Forward	Sense-Forward	Event-Trigger	
Saving in Application Length	82.35%	93.62%	97.59%	97.89%	**92.86%**
Saving in Application Vocabulary	76.92%	83.70%	93.77%	94.17%	**87.14%**
Saving in Application Volume	92.06%	96.18%	98.79%	98.86%	**96.47%**
Saving in Application Code Difficulty	87.50%	99.35%	99.88%	99.85%	**96.64%**
Saving in Application Programming Effort	99.58%	99.98%	100.00%	100.00%	**99.89%**
Saving in Application Programming Time	100.00%	98.09%	99.12%	97.17%	**98.59%**

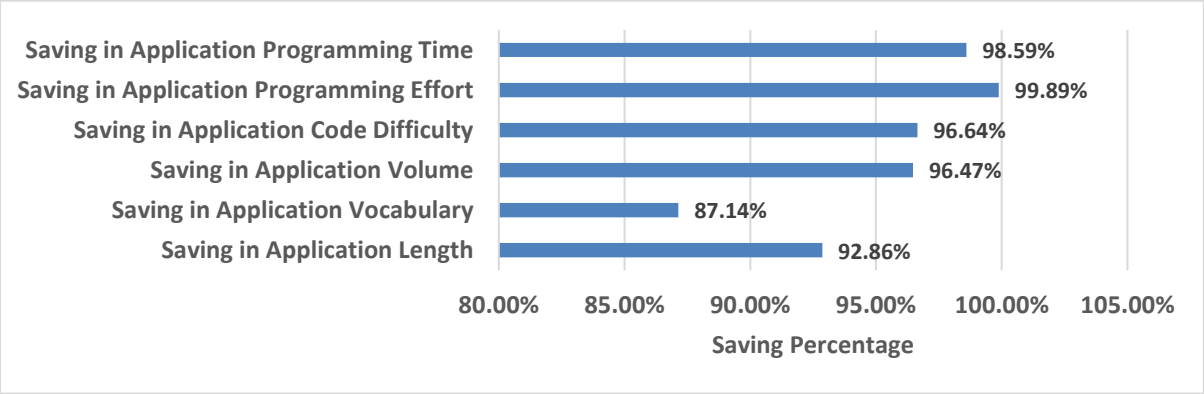

Figure 6-11: SenNet Saving Percentages According to HCM Formulas

To sum up, results from this evaluation method showed a significant advantage in using SenNet, because it reduces the programming vocabularies, effort and time. That implies SenNet helps in hiding development complexity, which can be considered an answer to **Q1** in the evaluation plan.

6.6 Business Case Study Applicability

WSN can be used with a broad range of applications, so to prove SenNet is applicable and useful in developing WSN applications, two real business case study scenarios have been developed to show the potential power of SenNet and how SenNet can be used to implement many types of application.

The first scenario can be summarised as defining different jobs for multiple sensor nodes, while the second scenario is defining a single job for a network of sensor nodes, as well as using node-level aggregation function for data processing in both scenarios. The chosen scenarios have been developed and tested completely except for the actuation parts, as the current version of SenNet is focusing on the sensing activities and tasks. The hardware used to test the generated code functionality for these scenarios is IRIS-XM2110 motes (ATmega1281 low-power microcontroller) equipped with MTS-400 and MDA-300 sensor board.

- **Scenario 1**: Light monitoring in tunnels (Ceriotti et al., 2011)
- **Scenario 2**: Temperature Monitoring for Smart Homes (Rodrigues, Delicato, et al., 2015)

Implementing these scenarios has proved SenNet is appropriate for developing WSN applications. That provides an answer for **Q3** in the evaluation plan.

6.6.1 Scenario 1 - Light Monitoring in Tunnels

A wireless sensor and actuator network is deployed in a traffic tunnel. The sensor devices sample light values, aggregate them, calculate the average over a 5-second period, and send it to the network, where they are intercepted by the tunnel control infrastructure, which in turn controls the artificial light level. The sampling frequency is location-dependent; interior nodes are sampled less often than motes in entrance and exit zones (Ceriotti et al., 2011).

This scenario has been implemented using SenNet as below:

A group of sensor nodes (3 motes) are programmed to sense light samples. The sensor devices sample light values, aggregate them, calculate the average over a 5-second period, and send the result to the sink. The SenNet application developed for this scenario can be seen in Listing 6-8 and Figure 6-12. The sensor nodes were programmed as follows: firstly, define a three NodeDataProcessing, because we have three different jobs, each applied to a different node. Then define the main job parameters, such as the necessary aggregation function which is avg; the dataProcessing rate which represents the timer will be used to fire the average calculation of the sensed data, as well as defining the sensing sampling frequency. Finally, the main node resources are defined that will be used to do the sensing job, which in our scenario is a light sensor. Figure 6-13 shows a sample screenshot for the TinyOS component graph checker that illustrates how the generated nesC files are error free and all required components are linked together correctly.

The first sensor node will sample the light every 0.5 minutes (sampling rate = 30000 milliseconds, as SenNet uses the TMillisec interface provided by TinyOS), then calculate the average samples every 5 minutes (sampling rate = 300000 milliseconds) and send the results to the sink. The second sensor node will sample the light every 1 minute (sampling rate = 60000 milliseconds), calculate the average samples every 5 minutes (sampling rate = 300000 milliseconds) and send the results to the sink. Finally, the third sensor node will sample the light every 2 minutes (sampling rate = 120000 milliseconds), calculate the average samples every 5 minutes (sampling rate = 300000 milliseconds) and send the results to the sink. The three nodes will send the messages to the sink node (Sink node ID = 1) using Active Message Protocol (AM) with the average results of the light sensed results. Finally, after completing the programming process of this scenario, a complete TinyOS application files are generated. The AMsg.h is a header file that will be used by the nodes to construct the sending message to the sink, and LMT10AppC.nc, LMT10C.nc and Makefile10 are the nesC files generated for the sensor node with node-ID = 10 that is configured in the SenNet application. A sample of the nesC generated files can be shown in APPENDIX G.1.

Listing 6-8: SenNet Application for Scenario-1

```
**********************************************************************
SeNetApp LMT {
	jobs {
		NodeDataProcessing {
			id 100
```

```
        nodeDataProcessing avg
        sensingSamplingRate 30000
        dataProcessingRate 300000
        JobTargetNode SensorNode {
                id 10
                nodeResources {
                        LightSensor
                }
        }
        jobaction {
                SendMessageAction {
                        sendMessageTo 1
                }
        }
},
NodeDataProcessing {
        id 200
        nodeDataProcessing avg
        sensingSamplingRate 60000
        dataProcessingRate 300000
        JobTargetNode SensorNode {
                id 20
                nodeResources {
                        LightSensor
                }
        }
        jobaction {
                SendMessageAction {
                        sendMessageTo 1
                }
        }
},
NodeDataProcessing {
        id 300
        nodeDataProcessing avg
        sensingSamplingRate 120000
        dataProcessingRate 300000
        JobTargetNode SensorNode {
                id 30
                nodeResources {
```

```
                        LightSensor
                    }
                }
                jobaction {
                    SendMessageAction {
                        sendMessageTo 1
                    }
                }
            }
        }
}
**********************************************************************
```

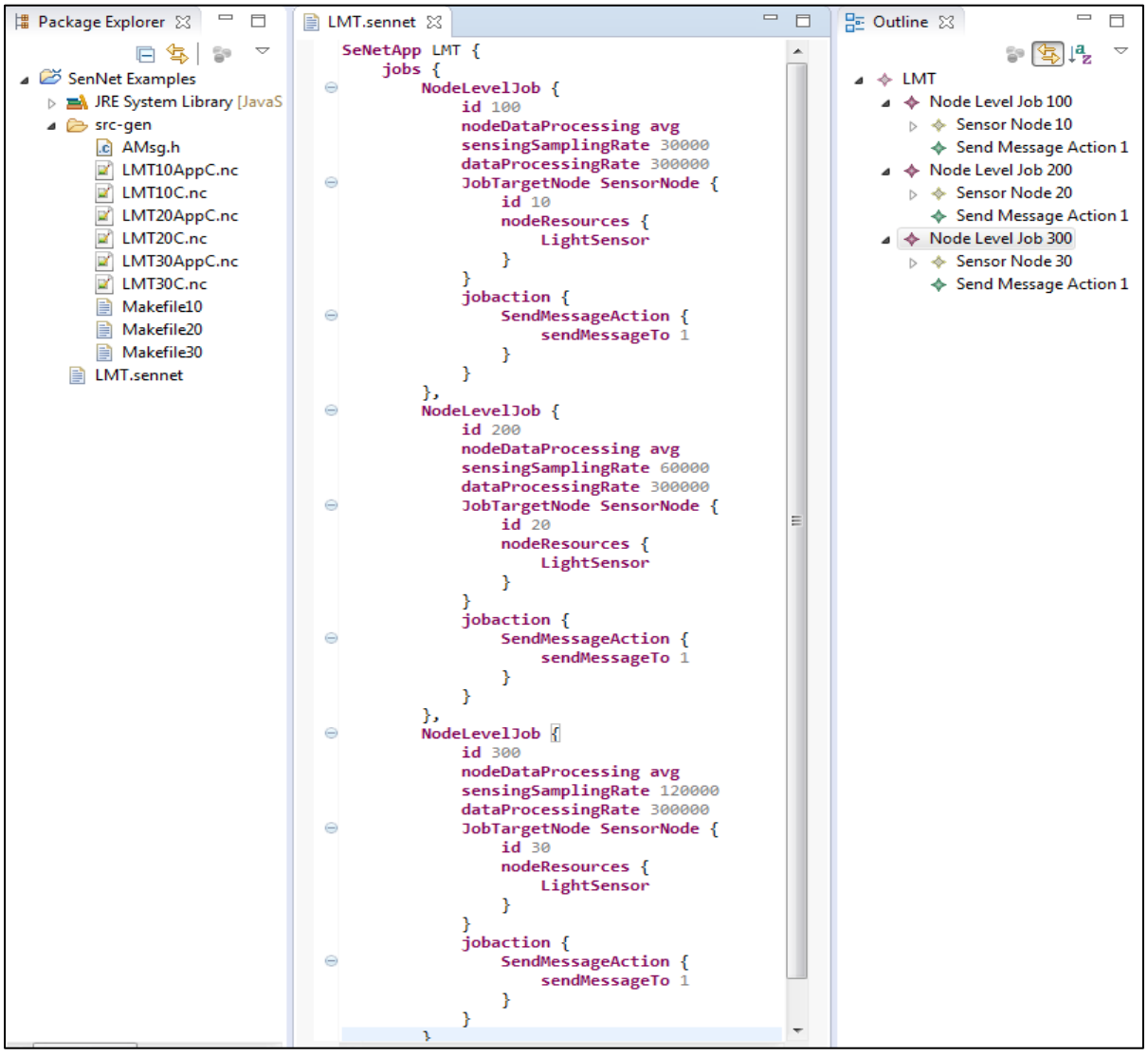

Figure 6-12: Sample Screenshot for the SenNet Editor for Scenario-1

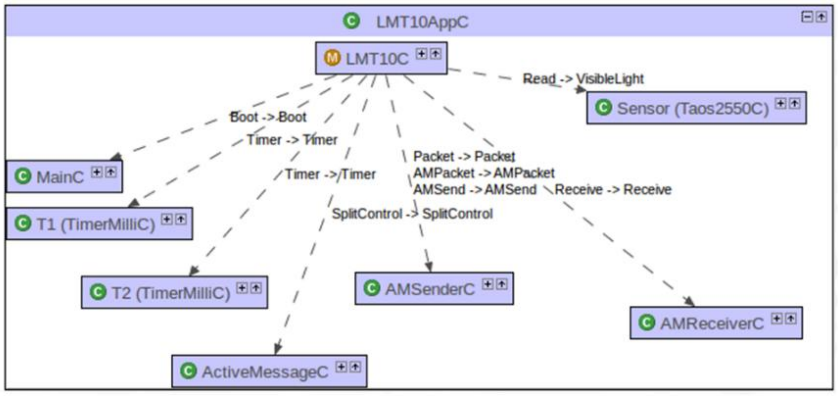

Figure 6-13: Sample Screenshot for the TinyOS Component Graph for Scenario-1

6.6.2 Scenario 2 - Temperature Monitoring for Smart Homes

Monitoring the temperature of a house instrumented by a wireless sensor network: collect room temperature data, process such data through an aggregate function so that only the highest among 10 samples is considered and finally, send only that sample to the sink node. The environment control is done through actuators connected to the air conditioner. The application has to keep temperature at 20C° at all times by controlling the air conditioner (Rodrigues, Delicato, et al., 2015).
This scenario has been implemented using SenNet as below:

A group of sensor nodes will be programmed to collect room temperature data, process the data through an aggregate function so that only the highest among 10 samples is considered, and finally send only that sample to the sink node. The whole scenario will be implemented in the lab.

Two sensor nodes are programmed to collect room temperature data every 1 minute. The sensed data is processed through an aggregation function so that only the highest data samples will be considered within a time window of 10 minutes and finally, a message is sent to the sink node if the temperature becomes greater than 20C°.

The SenNet application developed for this scenario can be seen in Listing 6-9 and Figure 6-14. The two sensor jobs will perform the same function, so the best way to develop a SenNet application for this scenario is by defining one job that will be applied to a network that includes two motes. So, firstly set a NetworkDataProcessing job command that will use the following configuration parameters:

- Apply max aggregation function as the main function that will be the main role of the sensor nodes included in the network.
- Define the sensing sampling rate that will be applied to the sensors included in the sensor nodes, which will be every 1 minute (1 minute = 60000 Milliseconds).
- Define the dataProcessingRate, which is the timer that will be responsible for initiating the aggregation function, which in our scenario is every 10 minutes, equal to 600000 Milliseconds.
- Define the network parameters and how many nodes will be included in this network.
- Define the resources that will be used by each sensor node inside the network to do the sensing job.
- Define the next job after completing the sensing and calculating the aggregation function, which is sending the final results to the sink node (sink node ID =1), if the final results are greater than 20 C°.

After completing the programming process of this scenario, a complete nesC single–node files are generated. The AMsg.h is a header file that will be used by the nodes to construct the sending message to the sink. In addition, TMSH2AppC.nc, TMSH2C.nc and Makefile2 are the nesC files that are generated for one of the two sensor nodes. According to the SenNet application, network-ID is then specified as 10, but we do not specify the nodes-ID for each sensor node, so SenNet has assigned node-ID 2 and 3 respectively, as node-ID = 1 is reserved for the sink node.

Figure 6-15 shows a sample screenshot for the TinyOS component graph checker that illustrates how the generated nesC files are error free and all required components are linked together correctly. Besides, sample of the nesC generated files can be shown in APPENDIX G.2.

Listing 6-9: SenNet Application for Scenario-2

```
**********************************************************************
SeNetApp TMSH {
       jobs {
              NetworkDataProcessing {
                     id 100
                     networkDataProcessing max
                     sensingSamplingRate 60000
                     dataProcessingRate 600000
                      JobTargetNetwork FlatNetwork {
                            id 10
                            nbOfNodes 2
```

```
                        resources {
                                TemperatureSensor
                        }
                }
                jobaction {
                        SendMessageAction {
                                sendMessageTo 1
                                condition ConditionalAction {
                                        sensorTerm Temp
                                        logicalSymbol greaterThan
                                        value 20
                                        unit C
                                }
                        }
                }
        }
    }
}
**********************************************************************
```

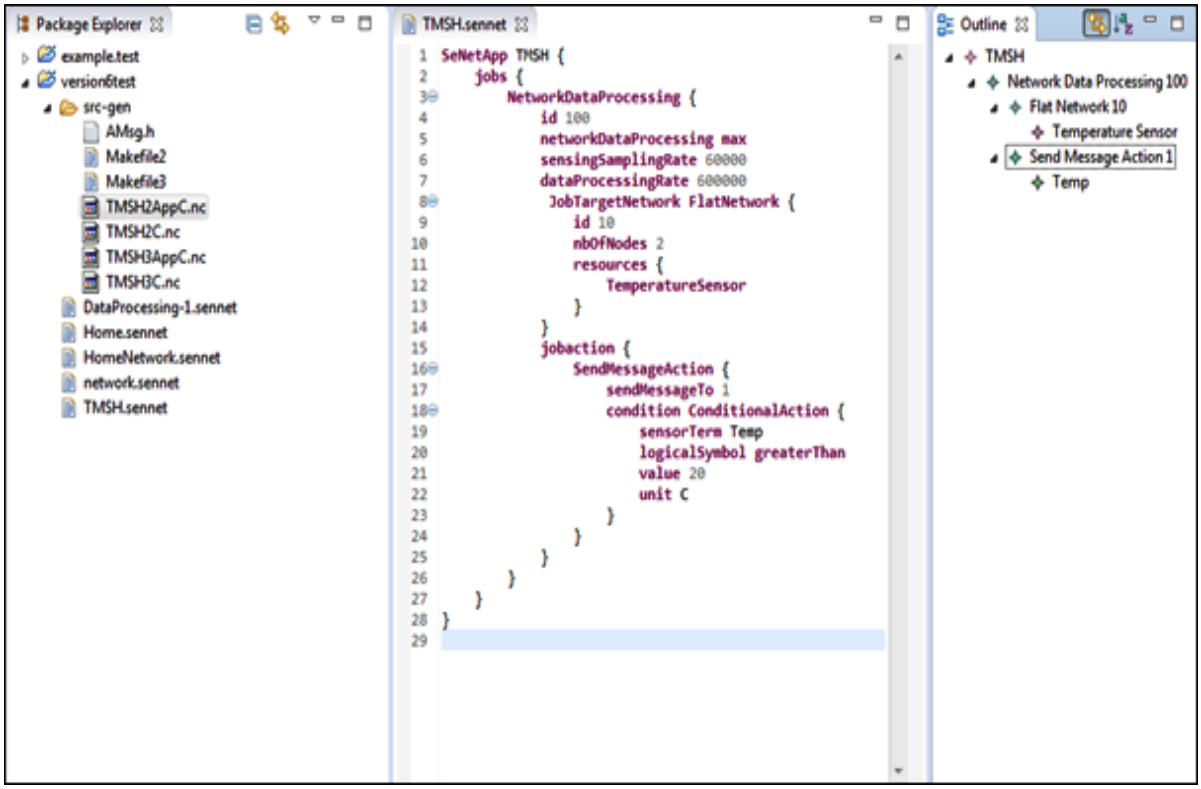

Figure 6-14: Sample Screenshot for the SenNet Editor for Scenario-2

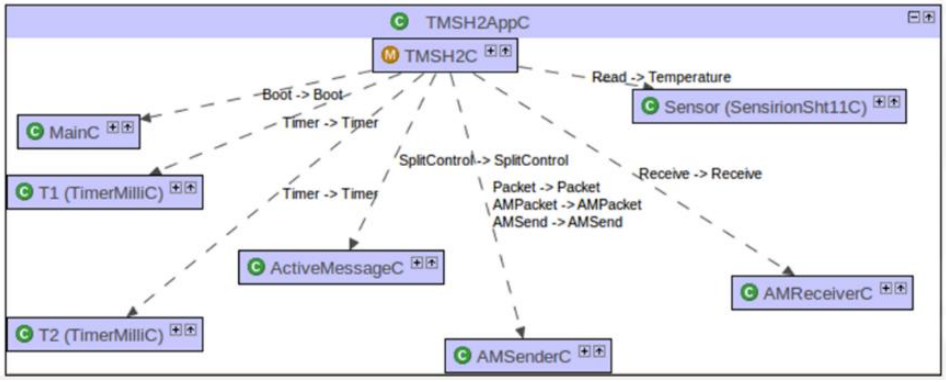

Figure 6-15: Sample Screenshot for the TinyOS Component Graph for Scenario-2

6.7 User Study Experiment

Proof of concepts scenarios, HCM and LOC measurement results show that SenNet is a language that can be used to develop WSN applications effectively, besides reducing the effort required to develop a WSN application. However, two aspects have not been addressed; the first aspect is developer's usability and how average developers (i.e. novice developers with respect to WSN programming) can use SenNet and whether they can deliver successful applications using it. The second aspect is related to the WSN domain, the question of whether SenNet is useful to represent WSN domain concepts and scenarios; we have therefore used controlled experiment as an evaluation technique to evaluate SenNet and answer these two aspects. Thus, a small-scale controlled experiment was performed to assess SenNet and nesC in terms of usability and functional suitability for first-time users. Guidelines for controlled experiment reported (Jedlitschka, Ciolkowski, & Pfahl, 2008) and Goal/Question/Metric (GQM) approach (Koziolek, 2008) have been adopted to ensure the correctness and integrity of this experiment's results. Ethical approval has been obtained to conduct a user study for this research, which can be found in APPENDIX H.1.

6.7.1 Goals/Questions/Metrics

This section discussed how the goals, questions and metrics are identified for this user study experiment according to the GQM approach, taking in consideration the Jedlitschka et al. (2008) guidelines.

- **Goals:** The experiment goals were identified according to research objective (**O3**), using a pre-defined format provided by Jedlitschka et al. (2008). Accordingly, the experiment goals can be identified as:

  G1. To analyse the aspects of **Usability** when using SenNet for the purpose of WSN application development.

  G2. To analyse the aspects of **Functional Suitability** when using SenNet for the purpose of WSN application development.

  Usability of a DSL is the degree to which a DSL can be used by specified users to achieve specified goals (Brooke, 2013; Kahraman & Bilgen, 2015; Lárusdóttir & Ármannsdóttir, 2005). The standard ISO-9241-11 (1998 as cited in Bevan, 2009, p. 2) defined Usability as *"the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use"*.

  Functional Suitability is the degree to which the DSL is fully developed (Kahraman & Bilgen, 2015), which means that all required functionalities are offered in the DSL. ISO/IEC 25010: 2011 define functional Suitability as *"The degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions"* (ISO/IEC, 2011, p. 10).

- **Questions:** Regarding questions then a number of questions have been formulated, as well as adapting some questions to fit our experiment goals from the Framework for Qualitative Assessment of DSLs (FQAD) (Kahraman & Bilgen, 2015), the System Usability Scale (SUS) (Brooke, 1996), besides some other questions proposed by Elsts (2013), Bai (2011), and Tullis & Albert (2013) to evaluate DSL. The experiment questions are divided into two parts: pre-experiment and post-experiment questions.

  1. **The Pre-Experiment questions:** these questions include nine questions used to generate a participant profile, such as programming skills, and technology interest, but with no personal information.

  2. **The Post-experiment questions:** these twenty-four questions focus on SenNet and nesC skills that participants gained in the experiment and the tasks they were asked to program. The twenty-four questions can be divided into four groups.

A. The first group is Introductory Questions (IQ), this group includes questions Q1-Q2 that focus on the SenNet and nesC presentations and tutorials that were given to the participants to assess their clarity.

B. The second group includes questions Q3-Q17 that related to the first experiment goal (G1), targeting usability, which implicitly measures effectiveness, efficiency, and many other metrics.

C. The third group includes questions Q18-Q23 that related to the second experiment goal (G2) and they implicitly measure appropriateness criteria.

D. The last group includes one question (Q24), which is an open question that gives participants the opportunity to add comments if they need to.

A complete list of pre- and post-experiment questions can be found in APPENDIX H.2 and H.3.

- **Metrics:** The metrics identified for this controlled experiment can be classified into two groups; the first group is measured directly from participants' feedback to specific questions, while the second group is measured according to participants' activity throughout the experiment, such as how much time participant X consumed to complete task Y. The general metrics identified for our controlled experiment can be summarised as:

M11. This metric is specified to evaluate SenNet **Effectiveness**, where effectiveness is the accuracy and completeness with which users achieve specific goals (Atoum & Bong, 2015; Productivity Commission, 2013). The purpose of M11 is to measure whether or not SenNet enables developers to develop a successful WSN application. This metric is measured by checking whether participants could complete their tasks or not, besides evaluating participants' responses to Question 3, which asks whether they consider that SenNet can help them to develop a WSN application.

M12. This metric is specified to evaluate SenNet **Efficiency**, where efficiency is the accuracy and completeness of tasks in terms of the resources consumed, such as time or mouse clicks (Atoum & Bong, 2015; Hussain & Ferneley, 2008; Productivity Commission, 2013). The purpose of this metric is to evaluate the resources consumed to complete a

set of tasks using SenNet and nesC (Fenton & Bieman, 2014). This metric was measured using the below measurements:

M121. Evaluating participant's feedback to Questions 4-8, which ask the participants whether they consider that SenNet will reduce the development time and the Number of activities for task achievement. They are also asked about SenNet and nesC and which programming language they consider needs less technology background and programming skills. Which reflects participants experience in terms of SenNet efficiency.

M122. Time-Saving for Task Accomplishment: which calculates the percentage of time saved by using SenNet to accomplish a task compared to nesC, this measurement is calculated for each task using Equation 6-1.

$$\frac{\text{nesC Time} - \text{SenNet Time}}{\text{nesC Time}} \text{ x } 100\%$$

Equation 6-1: Percentage of SenNet Time Saving for Each Task

M123. Total Time Saved for Tasks Accomplishment: which calculates the percentage of total time saved by using SenNet to accomplish all tasks compared to nesC, this measurement is calculated for the set of tasks using Equation 6-2.

$$\frac{\text{nesC Total Time} - \text{SenNet Total Time}}{\text{nesC Time}} \text{ x } 100\%$$

Equation 6-2: Percentage of SenNet Time Saving For All Tasks

M13. This metric is specified to evaluate SenNet **Likeability** (Kahraman & Bilgen, 2015), the goal of this metric is to assess the participant's opinion on acceptability and comfort (Brooke, 2013) using SenNet. This metric is measured by assessing the participant's feedback for Questions 9-12. These questions ask the participants their opinion about SenNet and nesC and which language they consider as a friendly language.

M14. This metric is specified to evaluate SenNet **Learnability** (Kahraman & Bilgen, 2015); the goal of this metric is to evaluate which programming language (SenNet or nesC) is easier to use and learn from the participant's perspectives. Questions 13 and 14 are used to measure this metric.

M15. This metric is specified to evaluate the **Programming Technique** preferred by the participants; this metric will measure whether participants prefer to use network-level or node-level programming technique. Participants' feedback on Question 15 is used to measure this metric.

M16. This metric is specified to evaluate SenNet's **Ability to Maintain Existing Application** (Kahraman & Bilgen, 2015), the goal of this metric is to evaluate in which programming language (SenNet or nesC) it will be easier to maintain or change an existing application from the participant's perspective. Participants' feedback on Question 16 is used to measure this metric. Participants will gain the ability to answer this question after examining Task-3, which will be discussed later.

M17. This Metric is specified to evaluate **Mind to program mapping**, the goal of this metric to evaluate how easy it is to map the application logic that is required by the developer into an application code, or what is the relationship between the program and what the programmer has in mind (Kahraman & Bilgen, 2015). Participant's feedback to Question 17 is used to measure this metric.

M21. This metric is specified to evaluate SenNet **Appropriateness**, which reflects whether all concepts and scenarios can be expressed using the DSL (Kahraman & Bilgen, 2015). The goal of this metric is to reflect whether participants consider that SenNet is appropriate to express WSN concepts and scenarios, and whether SenNet includes conflicting elements. The participant's feedback to Questions 18-23 is used to reflect this metric.

Table 6-10 and Figure 6-16 illustrated the relationship between goals, questions and metrics.

Table 6-10: Goal/Question/Metric Mapping for the User Study

Goal	Questions	Metric	
-	Q1-Q2	Introductory Questions (IQ)	
Usability	Q3	M11 effectiveness	
	Q4-Q8	M121 Efficiency - UX[16]	M12 Efficiency
	-	M122 Time-Saving for Task Accomplishment	
	-	M123 Total Time Saved for Tasks Accomplishment	
	Q9-12	M13 Likeability, user perception	
	Q13-Q14	M14 Learnability	
	Q15	M15 Programming Technique	
	Q16	M16 Maintaining existing applications	
	Q17	M17 Mind to program mapping	
Functional Suitability	Q18-Q23	M21 Appropriateness	

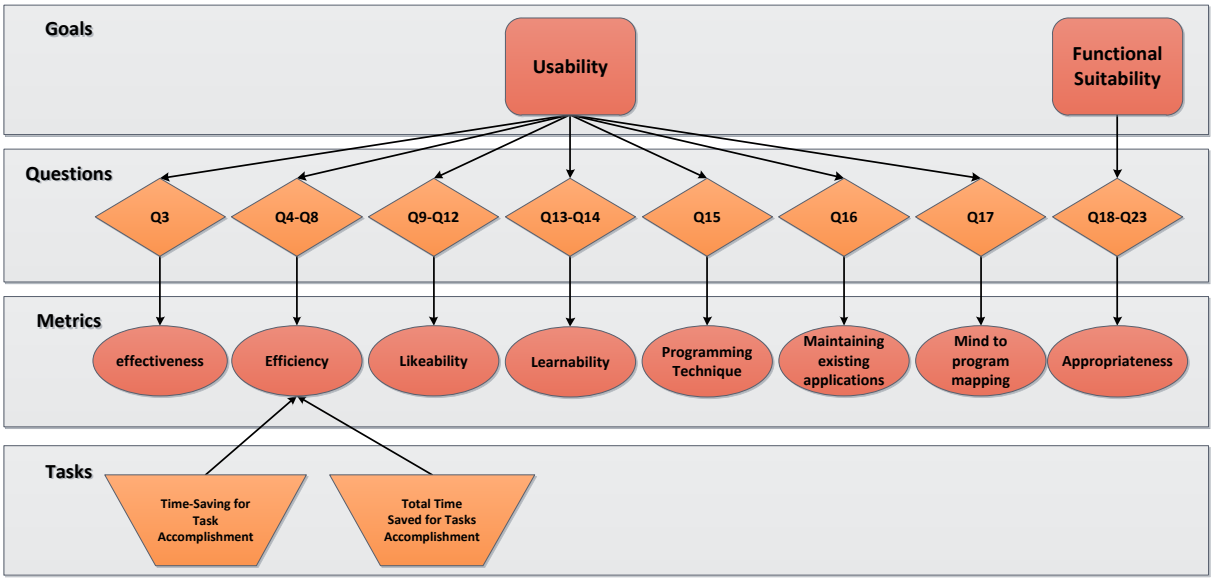

Figure 6-16: Goals/Questions/Metrics Mapping for the User Study

6.7.2 Question Rating Scales and Metrics Benchmarking

Regarding questions rating according to the literature review, User Experience (UX) can be used with many scaling rates such as (3, 4, 5 and 7). Odd or even number of values to be adopted is another thing that many researchers argue about. Interesting guidelines have been proposed by Tullis & Albert (2013) regarding rating scale:

- Using multiple scales will ensure obtaining more reliable data from the user.

[16] UX: User Experience

- Using an odd number of values to enable the user to be neutral, which is natural behaviour in real world situations.
- The total number of points: some researchers believe that more points are always better, but according to the literature (Cox, 1980; Friedman & Friedman, 1986) any more than nine points will rarely provide useful additional information. Furthermore, five and seven points are the highest number of scaling values used in real-world user experience questionnaires. Besides, Finstad (2010) established an interesting study comparing five and seven versions of the same set of rating scales, where five-point scales are more likely than seven points in the usability inventories.

After taking the above guidelines into consideration, the questions feedback rating was designed in two groups according to the purpose and nature of the questions; Table 6-13 shows the different question types.

- **Type-1 (SenNet General Evaluation):** This group of questions concerned evaluating SenNet behaviour in general, such as: "did you think SenNet is useful in developing WSN application?". Five-scale Likert feedback was used for this type of questions to measure participants' satisfaction with SenNet. Table 6-11 shows sample 5-scale questions and the value assigned to each scale.

Table 6-11: Type-1 Questions Sample

	Likert Scale	Value
Were the SenNet and nesC tutorials and presentation given easy to understand?	☐ Very Easy	5
	☐ Easy	4
	☐ Neutral	3
	☐ Difficult	2
	☐ Very Difficult	1

- **Type-2 (SenNet vs. nesC Evaluation):** This group of questions asked the participants feedback about a specific aspect, whether SenNet or nesC is more preferable. A three-scale Likert feedback was used for this type of question. Table 6-12 shows sample 3-scale questions and the value assigned to each scale.

Table 6-12: Type-2 Questions Sample

	Scale	Value
If you needed to develop a WSN application, which language toolkit would you prefer to use?	☐ SenNet	3
	☐ Undecided	2
	☐ nesC	1

Table 6-13: User Survey Question Types

Goal	Questions	Questions Type	Metric	
-	Q1-Q2	Type-1	Introductory Questions (IQ)	
Usability	Q3	Type-1	M11 effectiveness	
	Q4, Q5, Q8	Type-1	M121 Efficiency - UX	M12 Efficiency
	Q6, Q7	Type-2		
	-	-	M122 Time-Saving for Task Accomplishment (T1, T2, and T3)	
	-	-	M123 Total Time Saved for Tasks Accomplishment	
	Q9-12	Type-2	M13 Likeability, user perception	
	Q13-Q14	Type-2	M14 Learnability	
	Q15	Type-2	M15 Programming Technique	
	Q16	Type-2	M16 Maintaining existing applications	
	Q17	Type-2	M17 Mind to program mapping	
Functional Suitability	Q18-Q23	Type-1	M21 Appropriateness	

Regarding Metrics measurement, all the metrics are measured according to participants' feedback after understanding the presentations and practising the required tasks so that participants will answer the questions according to their experience; except that M122 (Time-Saving for Task Accomplishment) and M123 (Total Time Saved for Tasks Accomplishment) are measured according to the participant's tasks results. Table II shows the metrics required, besides the results range and the minimum required results. Some tasks have a direct impact on metric measurement; Table 6-14 presents the available metrics and their measurement methods. Concerning measurement methods, there are three methods to measure a metric. The first method is calculating the mean value for the participants' feedback using 5-scale rating feedback; this method is called 5-Scale User Feedback (5SUF). The second method is calculating the mean value of the participants' feedback using 3-scale rating feedback; this method is called 3-Scale User Feedback (3SUF). The last method is to measure the participant's tasks implementation statistically, using Equations 6-1 and 6-2, this method is known as Participant's Tasks Implementation Results (PTIR).

Table 6-14: Metrics Measurement Methods and Possible Values

Goals	Metrics		Measurement Method	Range	Baseline Value
Usability	M11 effectiveness		5SUF[17]	1 - 5	3
	M12 Efficiency	M121 SenNet Efficiency-UX	5SUF	1- 5	3
			3SUF[18]	1 - 3	2
		M122 Time-Saving for Task Accomplishment (T1, T2, and T3)	PTIR[19]	0% - 100%	50%
		M123 Total Time Saved for Tasks Accomplishment	PTIR	0% - 100%	50%
	M13 Likeability		3SUF	1 - 3	2
	M14 Learnability		3SUF	1 - 3	2
	M15 Programming Technique		3SUF	1 - 3	2
	M16 Maintaining existing applications		3SUF	1 - 3	2
	M17 Mind to program mapping		3SUF	1- 3	2
Functional Suitability	M21 Appropriateness		5SUF	1 - 5	3

6.7.3 Experiment Tasks

Studies related to WSN programming languages have been done previously by Miller et al. (2009) and Elsts et al. (2013), who assessed their new proposed languages BASIC and SEAL respectively. For that purpose, we selected some tasks to use in our user study experiment. Tasks T1, T2 and T3 were given to the participants to program using nesC and SenNet, while T4 was programmed by the participants using SenNet only, using network-level programming techniques which are not supported by nesC.

T1. Program a sensor node to Blink (toggle) a LED.

T2. Program a sensor node to sense the temperature with a sampling rate of 1 minute and turn LED1 on.

T3. To "describe the code functionality", snippets of SenNet and nesC code are provided to participants. Participants are asked to explain the code functionality. The SenNet snippets code handed to the participant is a sensor node configured to monitor room temperature every 3 minutes, then aggregate the sensed data every 15 minutes to compute the maximum temperature results, then if the resulting figure is greater than 20 C°, the sensor node has to send a message to the sink. In addition, the nesC snippets code handed to the participants is a sensor node configured to sense light every 1 minute, and then every 10 minutes, the

[17] 5SUF: 5-Scale User Feedback Mean Value.
[18] 3SUF: 3-ScaleUser Feedback Mean Value.
[19] PTIR: Participant's Tasks Implementation Results.

sensor node computes the average sensed data, and sends the results to the sink. Sample of SenNet and nesC snippets code can be found in APPENDIX H.4.

T4. Program 2 different sensing jobs, the first job applied to a single node, and the second job applied to a network of three sensor nodes.

The participants were given enough time to complete each task. The tasks, presentations, and material provided were designed such that no example code in the tutorial could be easily transformed into an exercise solution, but they had to describe all the required information. In this way, the set of exercises was non-trivial and forced the participants to apply the language primitives learned in the tutorial.

There is a direct relationship between the type of tasks and their accomplishment with the metrics measured for this experiment, as shown in Table 6-15. For example, the accomplishment of all tasks helps in measuring **M11** that is related to effectiveness in ensuring that all goals are met regardless of the resources utilised to accomplish these aims. The tasks T1, T2 and T3 help in calculating **M122** (Time-Saving for Task Accomplishment) and **M123** (Total Time Saved for Tasks Accomplishment). Likewise, T4 assists in measuring **M15** that assesses participants' preferences for working with node-level or network-level development abstraction. Finally, T3 helps in measuring **M16** that deals with whether SenNet or nesC is better for changing or editing existing applications.

Table 6-15: The Correlation between Goals, Questions, Metrics and Tasks

Goal	Questions	Metrics	Tasks
G1	Q3	M11	T1, T2, T3 and T4
	Q4-Q8	M121	
		M122	T1, T2 and T3
		M123	T1, T2 and T3
	Q9-12	M13	
	Q13-Q14	M14	
	Q15	M15	T4
	Q16	M16	T3
	Q17	M17	
G2	Q18-Q23	M21	

6.7.4 Experiment Participants

SenNet has been developed targeting beginner developers as discussed previously in Chapter 1. Fifteen participants were invited to participate in the user study, recruited from a population of current PhD students at the University of Salford, specifically targeting persons with

intermediate to good programming skills, but without any experience in WSN programming. Fifteen is considered a rational number, especially considering previous user studies done by Tei et al. (2015), Gordon et al. (2010), Rodrigues, Batista, et al. (2015), Cuenca, Bergh, Luyten, & Coninx (2005), and Elsts et al. (2013) to assess and evaluate newly developed DSL's, where they used 4, 5, 10, 12, and 18 participants respectively.

According to participants' answers to the pre-experiment questions, a complete participants profile has been constructed:

- **Study Background and Technology Interest:** The recruited participants' study background varied between Computer Engineering and Sciences, Electrical, Electronics and Communication Engineering, as illustrated in Figure 6-17. That was considered the perfect sample to deal with WSN technology. Some were also involved in different technologies such as robotics, cloud computing and mobile networks; some participants were involved in two or more technologies, such as P-5, who was involved in WSN routing protocols and cloud computing but had no experience in WSN programming. In terms of current participant's technology interest, we categorised them into five main groups (Networking, Acoustic, Image processing, Intelligent Power Management, and Control and Robotics) as described in **Error! Reference source not found.** and Figure 6-18.

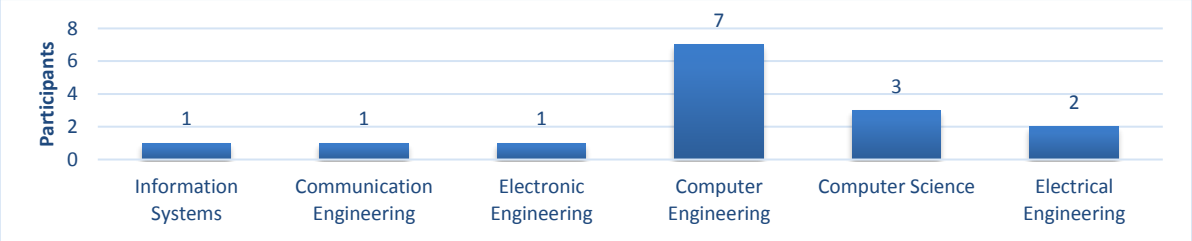

Figure 6-17: Participants' Study Background

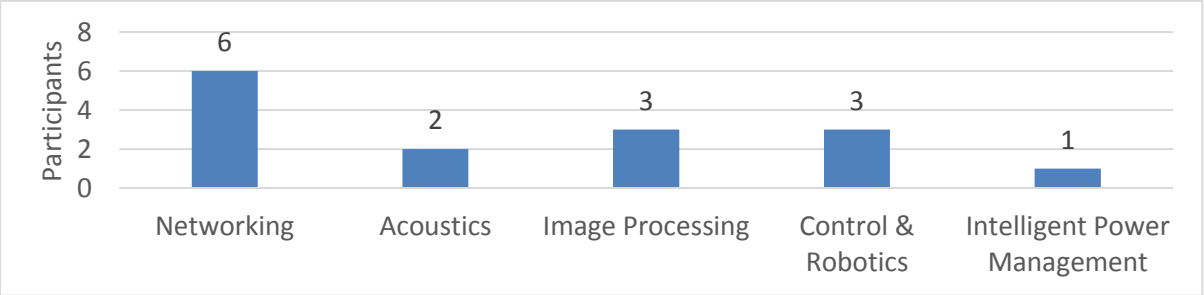

Figure 6-18: Participants' Current Technology Field of Interest

Table 6-16: Participants' Categorised Groups According to Technology Interest

	Technology Interest	General Technology
P-01	IoT	Networking
P-02	IPTV & Mobile Networks	Networking
P-03	Mobile Networks	Networking
P-04	Cloud Computing	Networking
P-05	Cloud Computing & WSN	Networking
P-06	Speech Recognition	Acoustic
P-07	Intelligent Power Management	Intelligent Power Management
P-08	Medical Image Processing	Image Processing
P-09	Medical Image Processing	Image Processing
P-10	Control & Robotics	Control & Robotics
P-11	Control & Robotics	Control & Robotics
P-12	Control & Robotics	Control & Robotics
P-13	Cloud Computing & Mobile Networks	Networking
P-14	Speech Recognition	Acoustic
P-15	Medical Image Processing	Image Processing

- **Programming Skills:** Participants were skilled in different General Programming Languages (GPL) such as C++, PHP, and Visual Basic, as illustrated in Figure 6-19. In addition, 73% of the participants were currently using DSL and GPL, while only 27% of the participants were using GPL only, as described in Figure 6-20.

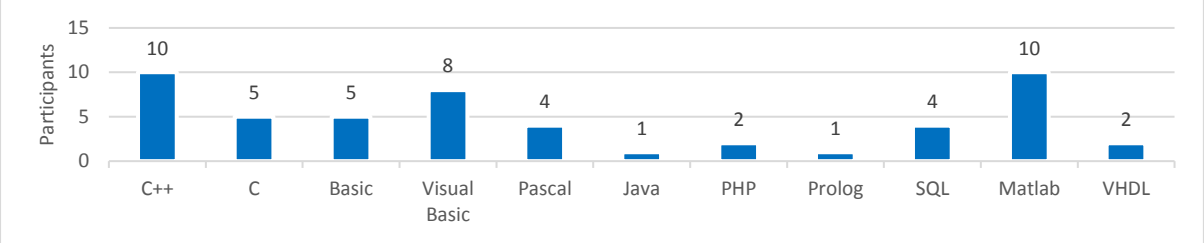

Figure 6-19: Participants' Programming Experience

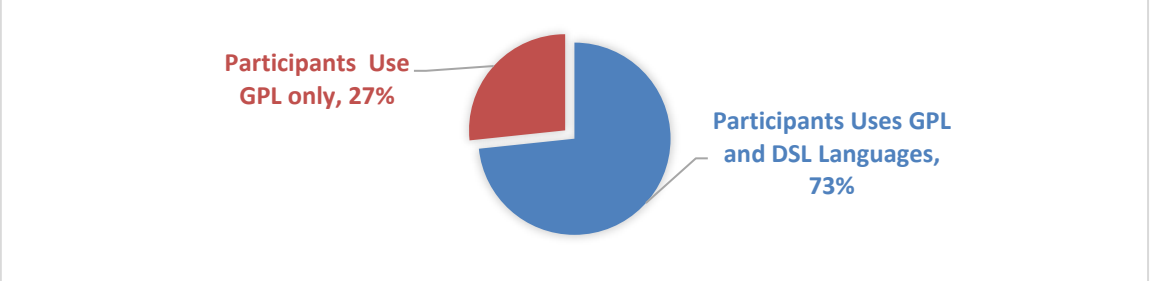

Figure 6-20: Participants Using GPL vs. DSL

- **Knowledge of WSN Technology:** The recruited participants had different knowledge levels of WSN technology, which was considered a good way to assess SenNet using participants with different knowledge levels in technology background, as shown in Table 6-17.

Table 6-17: Participants' Knowledge of WSN Technology

	Knowledge of WSN Technology
No Information	20%
Poor	40%
Average	27%
Good	7%
Excellent	7%

- **WSN Programming Experience:** All Participants had no experience in WSN programming.

6.7.5 Experiment Materials, Tools and Equipments

The participants were provided with a short and customised tutorial for nesC and SenNet, in addition to the nesC reference manual (Gay, Levis, Culler, & Brewer, 2009), prior to starting the tasks. In addition, the presentations and tutorials were formulated to match the tasks given to the participants. In any study that is comparing two languages for programmer usability, differences in the educational materials and references are a critical key. Therefore, we made a significant effort to ensure language materials of the same quality were provided.

Each participant was provided with a PC equipped with TinyOS 2.1.2 and nesC 1.3 environments. To help participants develop the required TinyOS applications using a comfortable environment, we provided them with an Eclipse Juno environment equipped with Yeti2[20] plug-in, as well as ready to work SenNet language packages and sufficient space to be familiar with the two languages.

6.7.6 Experiment Protocol

The general experiment agenda is illustrated in Table 6-18. The total time spent for the controlled experiment was around 04:40 hours, divided into three sessions: the first session related to a welcoming speech to identify the purpose of this user study and filling in of the

[20] Yeti2: http://tos-ide.ethz.ch/wiki/index.php

pre-experiment survey (10 min). This was followed by a presentation (120 min) discussing the basic concepts of WSN, nesC and SenNet. This session ended with a 10-min discussion to explain any issues not clear to the participants. The second session started with 30 min to give the participants the opportunity to be familiar with the nesC and SenNet interfaces, besides explaining the required tasks to be done by the participants, then 90 min was used to program the tasks by participants, while also recording the time needed to accomplish each task. Participants started working on the nesC tasks first, then after completing the nesC tasks, they started working on the SenNet tasks; these tasks were done by the participants in sequence (T1, T2, and T3) finishing with T4 for SenNet. Finally, in the third session (30 min) the participants were asked to fill in the final post-experiment questions.

Table 6-18: The Experiment Time Table

Session	Sub-session Activities	Time	Total Session Time
1	Welcoming and participants complete a pre-experiment survey	10 min	130 min
	A presentation about WSN, nesC, and SenNet	120 min	
2	Explain the required tasks to be programmed by the participants and provide the required materials.	30 min	120 min
	Tasks Implementation (T1, T2 and T3) using nesC and SenNet. Besides, T4 implementation using SenNet.	90 min	
3	Participants fill the post-experiment questions.	30 min	30 min

6.7.7 Experiment Results Discussion

This section will discuss the results obtained in the controlled experiment by evaluating the participants' feedback and observing the time spent to accomplish the set of tasks given to them. The participants' feedback and the tasks results were processed using Microsoft Excel and SPSS software applications.

To validate the results' integrity and check their reliability, we ran Cronbach's alpha index (Santos, 1999) to examine their internal consistency, as this measurement is widely used to verify and analyse the reliability of Likert-type question results, and the preferred alpha index value is > 0.7 (Creswell, 2009; Shull, Singer, & Sjøberg, 2008). The calculated Cronbach's alpha index for our results is 0.785, which can be considered a good result, reflecting highly inter-correlated results; Table 6-19 shows the Cronbach's Alpha results for each question.

Table 6-19: The Cronbach's Alpha Index for the Whole Data Set (23 Question Feedback)

Cronbach's Alpha Results Per Question Feedback Results	
Q1	.759
Q2	.792
Q3	.773
Q4	.753
Q5	.773
Q6	.761
Q7	.769
Q8	.736
Q9	.756
Q10	.796
Q11	.759
Q12	.769
Q13	.799
Q14	.787
Q15	.795
Q16	.787
Q17	.784
Q18	.783
Q19	.787
Q20	.787
Q21	.783
Q22	.798
Q23	.762

The participants' feedback regarding the clarity of the presented and provided materials, besides the task descriptions are illustrated in Table 6-20: most participants (**90%**) found the presentations, materials, and tasks description clear, informative and easy to understand, with a mean value = **4.46**[21].

Table 6-20: Introductory Questions Results

	Very Easy	**Easy**	**Neutral**	**Difficult**	**Very Difficult**
Q1: Were the SenNet and nesC tutorials and presentation given easy to understand?	60%	27%	13%	0%	0%
Q2: Were the descriptions of the tasks clear?	53%	40%	7%	0%	0%
Average	**57%**	**33%**	**10%**	**0%**	**0%**

[21] 5SUF: 5-Scale User Feedback Mean Value, as the range is (5=Strongly Agree, 4=Agree, 3=Neutral, 2=Disagree, 1=Strongly Disagree)

6.7.7.1 Metrics Results

This section discusses results of the previous identified metrics:

- **M11** (Effectiveness) results show that all participants (**100%**) agree that SenNet helped them to develop WSN application successfully, with a mean value of participants' feedback = **4.533**; Table 6-21 reveals that all participants completed their tasks using SenNet.

- **M12** (Efficiency), this metric is measured using M121, M122, and M123. These three metrics' results are described below:

    1. **M121** this metric measures SenNet efficiency using participants' feedback, where the results indicate that **91%** of the participants consider SenNet is efficient and decreases the development time and the number of activities for application development with a mean value = **4.51**, and **90%** of the participants chose using SenNet instead of nesC for WSN application development with a mean value = **2.9**[22]. Most of the participants (**80%**) chose SenNet as the language that needs less technology background than nesC, and all participants chose SenNet as the language that requires less programming skills than nesC.

    2. **M122** (Time-Saving for Task Accomplishment) Table 6-21 shows the detailed time needed for each task using SenNet and nesC. The average times needed to complete T1, T2 and T3 using nesC were 00:15:12, 00:16:48 and 00:10:24 minutes respectively, while participants performed these tasks using SenNet in 00:04:08, 00:07:04 and 00:04:24 minutes respectively. Calculating the Time-Saving for Task Accomplishment using Equation 6-1 shows that the average percentages of time saved for T1, T2 and T3 using SenNet were **72%**, **58%** and **55%**.

    3. **M123** (Total Time Saving for Tasks Accomplishment) that can be calculated using Equation 6-2, SenNet saved **63%** of the time required to complete the set of tasks compared to nesC, which shows the potential power of SenNet in saving development time and effort.

[22] 3SUF: 3-Scale User Feedback Mean Value, as the range is (3=SenNet, Neutral=2, 1=nesC)

Table 6-21: SenNet and nesC Task Completion Times (M122 and M123)

	nesC			SenNet			Time-Saving for Task Accomplishment			Total Time Saving for Tasks Accomplishment
	T1	T2	T3	T1	T2	T3	T1	T2	T3	
P-1	00:14:00	00:17:00	00:12:00	00:06:00	00:05:00	00:03:00	57%	71%	75%	67%
P-2	00:13:00	00:16:00	00:13:00	00:07:00	00:07:00	00:04:00	46%	56%	69%	57%
P-3	00:14:00	00:16:00	00:07:00	00:02:00	00:04:00	00:06:00	86%	75%	14%	68%
P-4	00:17:00	00:18:00	00:12:00	00:02:00	00:09:00	00:05:00	88%	50%	58%	66%
P-5	00:13:00	00:17:00	00:14:00	00:03:00	00:09:00	00:05:00	77%	47%	64%	61%
P-6	00:17:00	00:15:00	00:07:00	00:03:00	00:05:00	00:04:00	82%	67%	43%	69%
P-7	00:20:00	00:15:00	00:07:00	00:05:00	00:07:00	00:03:00	75%	53%	57%	64%
P-8	00:13:00	00:15:00	00:07:00	00:02:00	00:05:00	00:06:00	85%	67%	14%	63%
P-9	00:15:00	00:20:00	00:12:00	00:06:00	00:09:00	00:06:00	60%	55%	50%	55%
P-10	00:20:00	00:21:00	00:08:00	00:07:00	00:09:00	00:04:00	65%	57%	50%	59%
P-11	00:19:00	00:19:00	00:08:00	00:02:00	00:09:00	00:03:00	89%	53%	63%	70%
P-12	00:14:00	00:16:00	00:14:00	00:02:00	00:06:00	00:03:00	86%	63%	79%	75%
P-13	00:18:00	00:18:00	00:15:00	00:05:00	00:09:00	00:06:00	72%	50%	60%	61%
P-14	00:11:00	00:15:00	00:09:00	00:06:00	00:08:00	00:03:00	45%	47%	67%	51%
P-15	00:10:00	00:14:00	00:11:00	00:04:00	00:05:00	00:05:00	60%	64%	55%	60%
Average	**00:15:12**	**00:16:48**	**00:10:24**	**00:04:08**	**00:07:04**	**00:04:24**	**72%**	**58%**	**55%**	**63%**

- **M13** (Likeability), most participants (**92%)** chose SenNet as the language they found more acceptable and comfortable to work with than nesC, with a mean value = **2.91**. In the same way **93%** of the participants chose SenNet as the tool they would recommend in future for WSN application development.

- **M14** (Learnability), all participants (**100%**) chose SenNet as the language in which they could learn, use and remember language commands and vocabularies more easily than nesC, with a mean value = **3**.

- **M15** (Programming Technique), most participants (**80%)** preferred network-level programming to node-level programming in WSN application development, with a mean value = **2.6**.

- **M16** (Ability to Maintain Existing Application), most participants (**93%)** chose SenNet as it was easier than nesC in maintaining existing applications, especially in the case of maintaining applications developed by others, with a mean value = **2.93**.

- **M17** (Mind to program mapping), most participants (**73%)** chose SenNet as a programming language that is easy and provides better support in mapping a problem into a program code, with a mean value = **2.66**.

- **M21 (**appropriateness), most participants (**82%)** specified that SenNet was appropriate for WSN application development, with a mean value = **4.12**, and they considered that most of the WSN domain concepts and scenarios could be represented in SenNet.

Referring to Table 6-14 that lays out the main user study experiment metrics and their baseline values, Table 6-22 shows the final experiment results, where all metrics have higher values than the baseline values. These results indicate that SenNet gives beginner developers the required power to start developing WSN applications, as it hides complexity and enables developers to develop their applications smoothly and easily. Furthermore, SenNet offers many activities and tasks not available in nesC, besides offering the node and network-level development.

Table 6-22: User Study Experiment Metrics Results

Goals	Metrics		Range	Baseline	Value
Usability	M11 effectiveness		1 - 5	3	**4.5**
	M12 Efficiency	M121 SenNet Efficiency - UX	1 - 5	3	**4.5**
			1 - 3	2	**2.9**
		M122 Time-Saving for Task Accomplishment (T1, T2, and T3)	0% - 100%	50%	**72%, 58%, 55%**
		M123 Total Time Saving for Tasks Accomplishment	0% - 100%	50%	**63%**
	M13 Likeability		1 - 3	2	**2.9**
	M14 Learnability		1 - 3	2	**3**
	M15 Programming Technique		1 - 3	2	**2.6**
	M16 Maintaining existing applications		1 - 3	2	**2.9**
	M17 Mind to program mapping		1 - 3	2	**2.6**
Functional Suitability	M21 Appropriateness		1 - 5	3	**4.1**

6.7.7.2 Final Experiment Results

As a final conclusion, regarding usability, in terms of M11 and M12, most of the participants (**96%**) found SenNet was usable and helped them to achieve their required WSN application successfully with reduced development effort, where SenNet saves **63%** of the time required to complete a set of tasks compared to nesC. In the same way, **82%** of the participants found SenNet functionally suitable to be used for WSN application development in terms of appropriateness. Detailed user study experiment results are shown in Figure 6-21 and Table 6-23.

Furthermore, with regards to SenNet versus nesC usability (M12, M13, M14, M15, M16 and M17), **88%** of the participants found SenNet more usable than nesC for WSN application development, as illustrated in Figure 6-22 and Table 6-24.

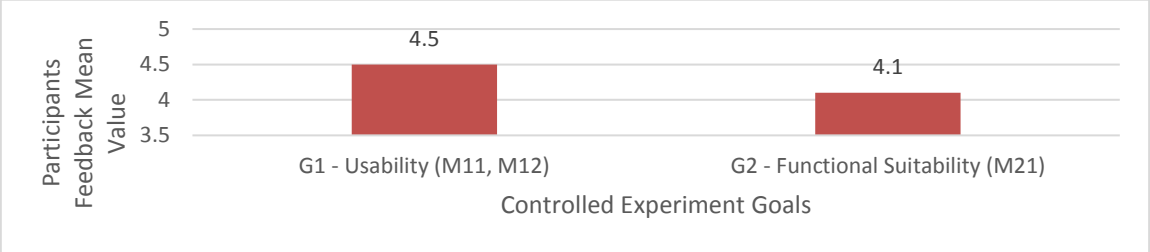

Figure 6-21: User Study Experiment Goals Final Results

Table 6-23: Final User Study Experiment Goals Results

	Mean	Agree Strongly	Agree	Neutral	Disagree	Disagree Strongly
G1 - Usability (M11, M12)	4.5	57%	39%	4%	0%	0%
G2 - Functional Suitability (M21)	4.1	32%	50%	16%	2%	0%

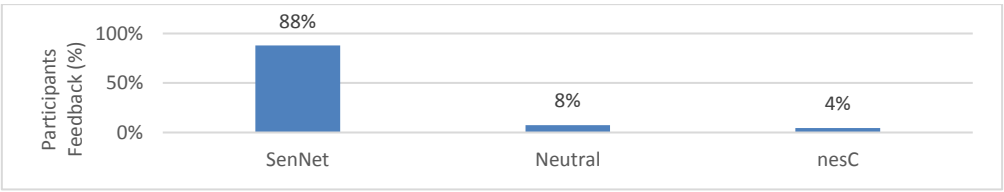

Figure 6-22: SenNet vs. nesC Usability

Table 6-24: SenNet vs. nesC Usability

	Mean	**SenNet**	**Neutral**	**nesC**
Usability (M12, M13, M14, M15, M16, M17)	2.8	88%	8%	4%

6.8 Evaluation Summary

This chapter has demonstrate the usefulness of SenNet and prove its functionality, a simple home temperature monitoring scenario has been developed, which proved that SenNet reduces the development efforts compared to nesC. Then it discussed many methods that were used to evaluate and assess the value of SenNet language and whether it is useful for beginner developers to develop WSN applications or not. All these methods were used to answer the questions proposed at the beginning of this chapter.

Q1. Can SenNet hide programming complexity and facilitate the development process?

Q2. Can developers learn and use SenNet language easily and rapidly?

Q3. Is SenNet suitable to explain WSN concepts and cover most domain scenarios?

Regarding **Q1**, concerning whether SenNet can hide development complexity or not, we have answered this question by conducting two methods of assessment, the first method has assessed SenNet and nesC in terms of specific scenarios using two code analysis measurements. This proved that SenNet significantly saved the total number of LOC and vocabularies that developers should deal with to develop an application, where SenNet is a small and compact language that helps developers to do their jobs with a minimal number of instructions. For the same purpose, a controlled experiment was conducted using 15 participants to assess SenNet and nesC, and the experiment results illustrated that most of the participants considered SenNet was more usable and required less technology background and programming skills than nesC; Table 6-25 discusses these two method results in detail.

Table 6-25: Question-1 Metrics Result

Method Used	Metrics		Possible Values Range	Baseline	Final Results
Application Source Code Analysis	Saving in LOC		0% - 100%	50%	88.4%
	Reducing in Code Difficulty		0% - 100%	50%	96.6%
	Saving in Programming Effort		0% - 100%	50%	99.8%
	Saving in Vocabulary		0% - 100%	50%	87.1%
	Saving in Program Length		0% - 100%	50%	92.8%
	Saving in Program Volume		0% - 100%	50%	96.4%
User Study Experiment	M11 Effectiveness		0% - 100%	50%	100%
	M12 Efficiency	M121 SenNet Efficiency Based on UX23	0% - 100%	50%	91%
			0% - 100%	50%	90%
		M122 Time-Saving for Task Accomplishment (T1, T2, and T3)	0% - 100%	50%	72%, 58%, 55%
		M123 Total Time Saving for Tasks Accomplishment	0% - 100%	50%	63%
	M15 Programming Technique		0% - 100%	50%	88.4%

To answer **Q2,** which related to whether SenNet language learning and using is easy, we have responded to this question using the conducted user study that shown SenNet is more likeable and learnable than nesC; Table 6-26 discusses these two method results in detail.

[23] UX: User Experience

Table 6-26: Question-2 Metrics Result

Method Used	Metrics	Possible Values Range	Baseline	Final Results
User Study Experiment	M13 Likeability	0% - 100%	50%	92%
	M14 Learnability	0% - 100%	50%	100%
	M16 Maintaining Existing Applications	0% - 100%	50%	93%
	M17 Mind to program mapping	0% - 100%	50%	73%

Finally, to answer **Q3,** which deals with whether SenNet is an appropriate tool to develop WSN applications, we have run three evaluation methods to prove that SenNet is suitable to develop WSN applications. The first method is scope of application functionality, where SenNet language was analysed based on Bai guidelines for WSN application functionalities, and the results showed that SenNet covers 75% of the WSAN functionalities. In the second method, we developed two real business scenarios to prove SenNet's suitability. The final method used to assess SenNet suitability was through a user study, where most of the participants found SenNet functionally suitable to develop WSN applications; Table 6-27 discusses these three method results in detail.

Table 6-27: Question-3 Metrics Result

Method Used	Metrics	Possible Values Range	Baseline	Final Results
Scope of Application Functionality	Scope of Application Functionality	0% - 100%	50%	75%
Business Case Study Applicability	Applicability to Develop Real-Business Applications	0 - 1	1	1
User Study Experiment	M21 Appropriateness	0% - 100%	50%	82%

Chapter 7: Discussion, Conclusion and Future Work

7.1 Introduction

This chapter presents a summary of the work, describing the key contributions, strengths and limitations of the research.

7.2 Discussion (Research Outcomes)

The focus of this research project is to reduce complexity in developing WSN systems. Figure 7-1 and the following questions showed how our research has succeeded in answering and addressing the research problems, besides highlighting the main research theme:

- **How Was the Research Problem Identified?**

To identify the research problem accurately, three surveys have been presented: The first survey was concerned with the main challenging and problematic dimensions that face developing WSN applications mentioned in section **1.3.1** and published in our paper (Salman & Al-Yasiri, 2016a), where 38 scientific papers were assessed. In this survey, we categorised the difficulties that face WSN application development in two dimensions, the accidental complexity and the inherent complexity dimensions.

The second survey concerned the different types of WSN applications developers and showing their knowledge gap, which was mentioned in section **1.3.2** that assessed more than 80 scientific research papers looking for potential WSN developer types; a summary version of this survey has been published (Salman & Al-Yasiri, 2016a, 2016b). In this study, we tried to show all possible developer types for WSN applications and presented their knowledge gaps in terms of WSN technology background and programming skills.

The third survey was concerned with the related solutions that proposed by the research community, which was mentioned in section **2.4**, where 24 solutions to reduce complexity in WSN domain have been studied. Furthermore, the discussed solutions have assessed in terms of a set of feature criteria that suggested in previous studies.

- **How Has the Research Problem Been Addressed?**

As a result of the conclusions in the reviews above particularly the third review, which discussed the related solutions, a set of design requirements have been outlined in section **2.5**. These design requirements were taken into consideration while developing our proposed solution SenNet explained in chapters **3**, **4** and **5**.

Chapter **3** has addressed the first part of our proposed solution which is the SenNet meta-model. That models the sensor networks domains (Application-Level and Network-Level) that build on the basis of the ISO/IEC 29182, which is considered a methodology that a developer can use to find his path to developing a WSN application (Salman & Al-Yasiri, 2016a, 2016b). This meta-model will be the foundation for the second part of our research solution.

As the second part of our research, The MDD approach is used to utilise the above identified meta-model and define the SenNet language. SenNet language is shown in Chapters 4 and 5: Chapter **4** discussed the SenNet internal view, the main components and the source-to-source translation processes. In Chapter **5**, the SenNet language programmer's view is represented, such as the text and tree-based editor views, the general SenNet application syntax and the different and offered commands syntax are discussed.

The proposed solution tries to tackle the research problem by hiding irrelevant information and presenting the network-level application development strategy as a development abstraction; in addition, the identified language primitives should be easy to learn and remember by the beginner developers.

- **How Has the Addressed Solution Been Assessed?**

The proposed solution assessment and evaluation is described in Chapter **6**, where many evaluation methods have been used to assess SenNet in terms of usability and functional suitability.

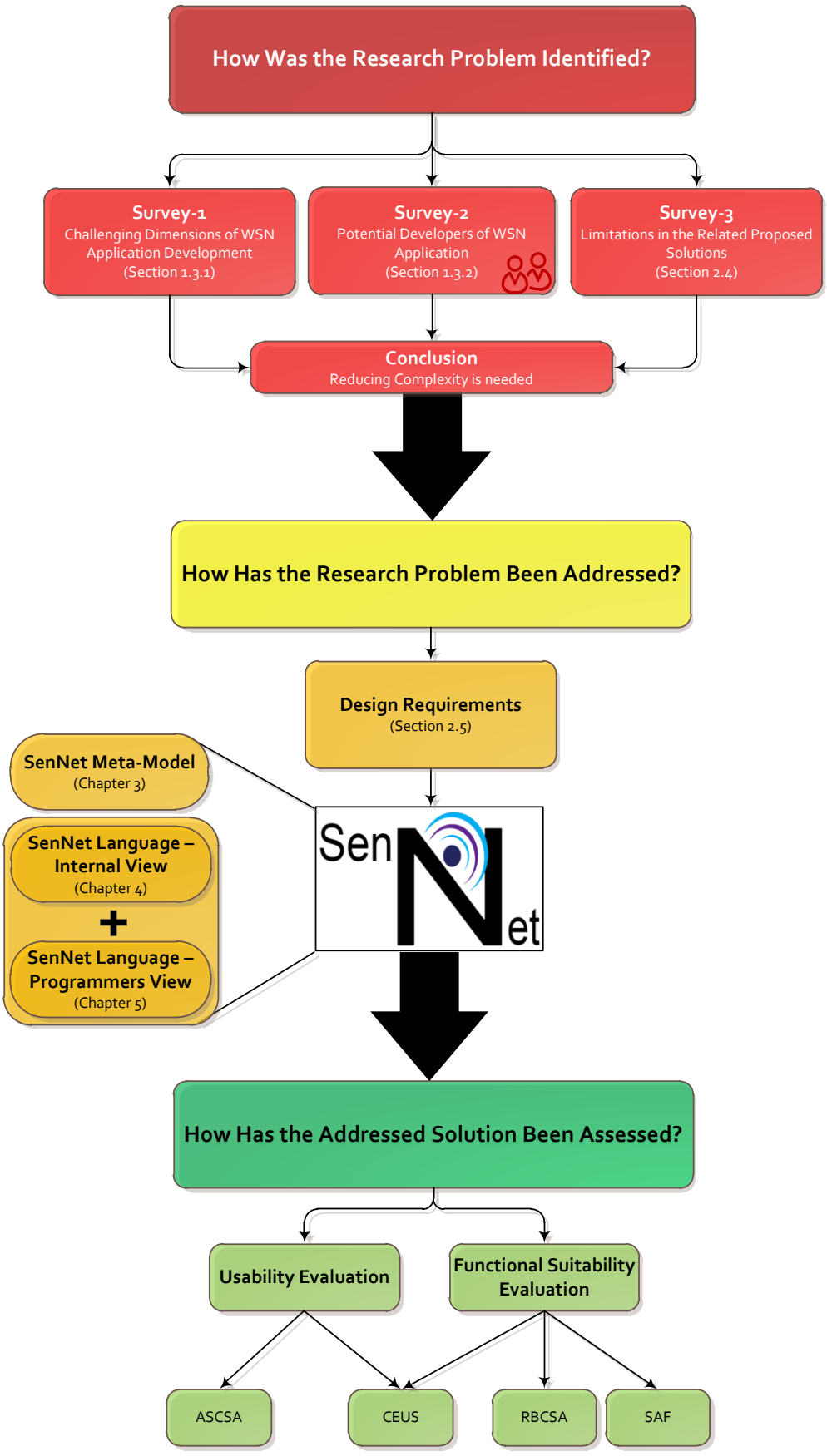

Figure 7-1: The Outline of The Thesis Structure

7.3 Conclusion

We have presented different aspects of the SenNet language and evaluated it using different methods. SenNet demonstrates advances over other WSN development methods in several areas:

- **Usability:** SenNet provides an abstracted solution, which uses concepts that are either familiar to the developer or only require a short time to grasp. For example, in the Contiki operating system, if the developer needs to use a timer using C language, then the developer has to learn and think in terms of hardware ticks. Equally, in TinyOS and nesC, the developer must either use hardware ticks or time unit. Similarly, SenNet allows using the millisecond as a time unit. SenNet has been proven to be more usable than nesC for inexperienced WSN users by results in a small-scale user study, besides other methods that prove the same concept.

- **Functional Suitability:** SenNet is a language designed for use in developing WSN applications, applications that are not restricted to a particular application domain, for example human health or structural monitoring applications such as SPINE (Bellifemine et al., 2011), which is a DSL designed for developing human health monitoring applications. SenNet showed its functional suitability to develop WSN applications by developing two real business case studies, besides other methods that prove the same concept.

- **Multi-Level Development Abstraction:** SenNet is designed to offer the possibility to develop WSN applications either in node-level or network-level, and giving the ability to model node, group of nodes and network. Which offers significant advances compared to currently available programming languages and development methods such as SEAL (Elsts et al., 2013). On the other hand, some proposed development methods use the MDD approach to defining three different modelling layers. Each layer deals with a particular part of the network activities, which entails identifying either specific DSL or DSML to be used with each layer, such as in LWiSSy (Dantas et al., 2013). In contrast to these two approaches, SenNet offers both node and network-level using one affordable and easy to grasp the language.

- **Portability:** WSN hardware architecture and platforms come in different varieties; different sensor node platforms may be used in one application. Therefore, portability is a

critical requirement for WSN applications. Because SenNet uses HIL interfaces, as shown in most of the case studies developed and presented in this thesis, this means that the amount of platform dependent code is very small, so porting the application code to another platform is fast and easy.

Furthermore, a number of concluding remarks can be highlighted:

I. The development of a WSN application using MDD & DSL approaches allows the code to be much more concise and free from low-level details compared to the implementation of the same application using conventional programming languages.
II. Using DSL allows the use of concepts that are already familiar to the developer and are easy to grasp.
III. Using high-level abstracted DSL with the aid of EMF, more facilities can be offered, such as a tree-based editor view; furthermore, it can be adapted easily to a visual development environment. So, the development process will be a more exciting journey, and encourage beginner developers to get into this world easily and smoothly.
IV. If the developer has a specific algorithm for a certain domain application, then giving the developer the ability to embed his algorithm within a SenNet application is a great advantage.

7.4 Limitations

The research work introduced in this thesis has some limitations, which they are beyond the scope of this study:

- The scope of SenNet language is limited to sensing job types.
- The current SenNet version supports standard data aggregation processing functions, such as max, min and average.
- The current SenNet version uses independent hardware interfaces provided by TinyOS, which implies that special hardware types are not supported. As for each customised hardware, different independent interfaces should be developed.
- Node mobility is not supported by the current SenNet version, due to the limitations of current mobility routing and localisation algorithms.

7.5 Future Work

To address the above limitations and improve the language's applicability, the followings future improvements may be undertaken:

- **Embedding Actuation Facilities:** Expanding the SenNet meta-model and language semantic by defining new actuation tasks and activities.
- **Adding Advanced Functions:** This can be done by embedding more algorithms and functions related to:
    1. Aggregation and Fusion algorithms.
    2. Mobility routing and localisation algorithms
- **Expanding the set of supported hardware types**: Expanding the SenNet commands to support special types of hardware using TinyOS HAL and HPL interfaces.

More future work aspects can be done for SenNet, such as:

- **Generate Source Code for Other Languages:** The CGC component can be reconfigured to generate source code for languages other than nesC, such as C or Java.
- **Providing Advanced Visual Programming:** More sophisticated visual programming and development editor facilities can be integrated into the SenNet package.
- **Expanding the Language Primitives:** The SenNet language can be expanded to offer creating or defining new functions. On the one hand this expansion will provide the developer with great flexibility. One the other hand, the new language version should not lose the high-level of abstraction it includes and change it to a new form of GPL languages.
- **Validating the Generated Application:** The TinyOS component graph can be integrated with the SenNet package to validate the functionality of the auto-generated nesC code applications.

References

Áemília, P., Ľubomír, W., Sergej, C., & Ján, K. (2011). The Effect of Abstraction in Programming Languages. *Journal of Computer Science and Control Systems*, *4*(1), 137.

Afanasov, M., Mottola, L., & Ghezzi, C. (2013). Towards context-oriented programming in wireless sensor networks. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication - UbiComp '13 Adjunct* (pp. 151–154). New York, New York, USA: ACM Press. http://doi.org/10.1145/2494091.2494141

Afanasov, M., Mottola, L., & Ghezzi, C. (2014). Context-Oriented Programming for Adaptive Wireless Sensor Network Software. In *2014 IEEE International Conference on Distributed Computing in Sensor Systems* (pp. 233–240). Marina del Rey (CA, USA): IEEE. http://doi.org/10.1109/DCOSS.2014.31

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, Techniques, & Tools* (2nd Editio). United States of America: Pearson Education Limited.

Akbal-Delibas, B., Boonma, P., & Suzuki, J. (2009). Extensible and Precise Modeling for Wireless Sensor Networks. In *Information Systems: Modeling, Development, and Integration* (pp. 551–562). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-01112-2_55

Akyildiz, I. F., & Vuran, M. C. (2010). *Wireless sensor networks*. (I. F. Akyildiz, Ed.). West Sussex, England: JohnWiley & Sons Ltd. Retrieved from http://210.32.200.159/download/20100130212654891.pdf

Alkandari, A., Alnasheet, M., Alabduljader, Y., & Moein, S. M. (2012). Water monitoring system using Wireless Sensor Network (WSN): Case study of Kuwait beaches. In *2012 Second International Conference on Digital Information Processing and Communications (ICDIPC)* (pp. 10–15). IEEE. http://doi.org/10.1109/ICDIPC.2012.6257270

Ammari, H. M. (2014). *The Art of Wireless Sensor Networks Volume 1: Fundamentals*. (H. M. Ammari, Ed.) (Vol. 1). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-40066-7

Antonopoulos, C., Asimogloy, K., Chiti, S., D'Onofrio, L., Gianfranceschi, S., He, D., … Vlahoy, G. (2016). Integrated Toolset for WSN Application Planning, Development, Commissioning and Maintenance: The WSN-DPCM ARTEMIS-JU Project. *Sensors*, *16*(6), 804. http://doi.org/10.3390/s16060804

Aoun, C. G., Alloush, I., Kermarrec, Y., Champeau, J., & Zein, O. K. (2015). A Modeling Approach for Marine Observatory. *Sensors & Transducers*, *185*(2), 129–139.

Atmel. (2006). *Atmel: 8-bit Atmel Microcontroller ATmega128L datasheet*. Retrieved from http://www.atmel.com/images/doc2467.pdf

Atoum, I., & Bong, C. H. (2015). Measuring Software Quality in Use: State-of-the-Art and Research Challenges. *ASQ.Software Quality Professional*, *17*(2), 4–15. Retrieved from http://arxiv.org/abs/1503.06934

Avilés-López, E., & García-Macías, J. A. (2009). TinySOA: a service-oriented architecture for wireless sensor networks. *Service Oriented Computing and Applications*, *3*(2), 99–108. http://doi.org/10.1007/s11761-009-0043-x

Bader, S., & Oelmann, B. (2013). Concealing the complexity of node programming in wireless sensor networks. In *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing* (pp. 177–182). IEEE. http://doi.org/10.1109/ISSNIP.2013.6529785

Baggio, A. (2005). *Wireless sensor networks in precision agriculture*. *Workshop on Real-World Wireless Sensor Networks (REALWSN 2005)*. Stockholm, Sweden: ACM.

Bai, L. (2011). *Simplifying Design of Wireless Sensor Networks with Programming Languages , Compilers , and Synthesis*. University of Michigan, USA.

Bai, L., Dick, R., & Dinda, P. (2009). Archetype-based design: Sensor network programming for application experts, not just programming experts. In *Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on* (pp. 85–96). San Francisco, CA: IEEE. Retrieved from http://dl.acm.org/citation.cfm?id=1602175

Bai, L., Dick, R. P., Chou, P. H., & Dinda, P. a. (2011). Automated construction of fast and accurate system-level models for wireless sensor networks. *2011 Design, Automation & Test in Europe*, (1), 1–6. http://doi.org/10.1109/DATE.2011.5763178

Bai, L., Dick, R. P., Dinda, P. A., & Chou, P. H. (2011). Simplified programming of faulty sensor networks via code transformation and run-time interval computation. In *Conference & Exhibition of Design, Automation & Test in Europe (DATE), 2011* (pp. 1–6). Grenoble, France: IEEE. http://doi.org/10.1109/DATE.2011.5763023

Bakshi, A., Prasanna, V. K., Reich, J., & Larner, D. (2005). The Abstract Task Graph: A Methodology for Architecture-Independent Programming of Networked Sensor Systems. In *EESR '05 Proceedings of the 2005 workshop on End-to-end, sense-and-respond systems, applications and services* (pp. 19–24). USENIX Association.

Basili, V. R. (1992). *Software Modeling and Measurement: The Goal/Question/Metric paradigm* (Technical Report). MD, USA.

Beal, J., Dulman, S., Usbeck, K., Viroli, M., & Correll, N. (2012). Organizing the Aggregate: Languages for Spatial Computing. *Formal and Practical Aspects of Domain-Specific Languages*, 436–501. http://doi.org/10.4018/978-1-4666-2092-6.ch016

Beckmann, K., & Thoss, M. (2010). A model-driven software development approach using OMG DDS for wireless sensor networks. In *Software Technologies for Embedded and Ubiquitous Systems* (pp. 95–106). Springer Berlin Heidelberg. Retrieved from http://link.springer.com/chapter/10.1007/978-3-642-16256-5_11

Beigl, M., Krohn, A., Riedel, T., Zimmer, T., Decker, C., & Isomura, M. (2006). The uPart experience: Building a wireless sensor network. In *Proceedings of the fifth international conference on Information processing in sensor networks - IPSN '06* (pp. 366–373). Nashville, TN, USA: IEEE. http://doi.org/10.1145/1127777.1127832

Bellifemine, F., Fortino, G., Giannantonio, R., Gravina, R., Guerrieri, A., & Sgroi, M. (2011). SPINE: A domain-specific framework for rapid prototyping of WBSN applications. *Software: Practice and Experience*, *41*(3), 237–265. http://doi.org/10.1002/spe.998

Bettini, L. (2013). *Implementing Domain-Specific Languages with Xtext and Xtend*. *Packt Publishing*. Birmingham, UK: Packt Publishing Ltd. Retrieved from http://www.packtpub.com/implementing-domain-specific-languages-with-xtext-and-xtend/book

Bevan, N. (2009). Extending Quality in Use to Provide a Framework for Usability Measurement. In K. M. (Ed.), *Human Centered Design* (HCD 2009, Vol. 5619 LNCS,

pp. 13–22). Springer, Berlin, Heidelberg. http://doi.org/10.1007/978-3-642-02806-9_2

Bézivin, J. (2005). On the Unification Power of Models. *Software & Systems Modeling*, *4*(2), 171–188. http://doi.org/10.1007/s10270-005-0079-0

Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., … Han, R. (2005). MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. *Mobile Networks and Applications*, *10*(4), 563–579. http://doi.org/10.1007/s11036-005-1567-8

Birken, K. (2014). Building Code Generators for DSLs Using a Partial Evaluator for the Xtend Language. In M. T. & S. B. (Eds.), *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change. ISoLA 2014* (Vol. 8802, pp. 407–424). Springer-Verlag Berlin Heidelberg. http://doi.org/10.1007/978-3-662-45234-9_29

Boonma, P., Somchit, Y., & Natwichai, J. (2013). A Model-Driven Engineering Platform for Wireless Sensor Networks. In *2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing* (pp. 671–676). COMPIEGNE, France: IEEE. http://doi.org/10.1109/3PGCIC.2013.115

Boonma, P., & Suzuki, J. (2010a). Moppet: A Model-Driven Performance Engineering Framework for Wireless Sensor Networks. *The Computer Journal*, *53*(10), 1674–1690. http://doi.org/10.1093/comjnl/bxp129

Boonma, P., & Suzuki, J. (2010b). TinyDDS: An Interoperable and Configurable Publish/Subscribe Middleware for Wireless Sensor Networks. In A. M. Hinze & A. Buchmann (Eds.), *Principles and Applications of Distributed Event-Based Systems* (pp. 206–231). USA: IGI Global. http://doi.org/10.4018/978-1-60566-697-6.ch009

Boonma, P., & Suzuki, J. (2011). Model-Driven Performance Engineering for Wireless Sensor Networks with Feature Modeling and Event Calculus. In *Proceedings of the 3rd workshop on Biologically inspired algorithms for distributed systems - BADS '11* (pp. 17–24). Karlsruhe, Germany: ACM Press. http://doi.org/10.1145/1998570.1998574

Boulis, A., Han, C.-C., & Srivastava, M. B. (2003). Design and Implementation of a Framework for Efficient and Programmable Sensor Networks. In *Proceedings of the 1st international conference on Mobile systems, applications and services - MobiSys '03* (pp. 187–200). New York, New York, USA: ACM Press. http://doi.org/10.1145/1066116.1066121

Brooke, J. (1996). SUS - A quick and dirty usability scale. In P. W. Jordan, B. Thomas, I. L. McClelland, & B. Weerdmeester (Eds.), *Usability Evaluation In Industry* (pp. 189–194). Taylor & Francis Ltd.

Brooke, J. (2013). SUS : A Retrospective. *Journal of Usability Studies*, *8*(2), 29–40. Retrieved from http://www.usabilityprofessionals.org/upa_publications/jus/2013february/brooke1.html%5Cnhttp://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

Brooks, F. (1987). No Silver Bullet Essence and Accidents of Software Engineering. *IEEE Computer*, *20*(4), 10–19. http://doi.org/10.1109/MC.1987.1663532

Bulusu, N., Heidemann, J., & Estrin, D. (2000). GPS-less low-cost outdoor localization for very small devices. *IEEE Personal Communications*, *7*(5), 28–34. http://doi.org/10.1109/98.878533

Buonadonna, P., Tolle, G., & Gay, D. (2010). BaseStation Application. Intel Research

Berkeley. Retrieved from https://github.com/tinyos/tinyos-main/blob/master/apps/BaseStation/BaseStationP.nc

Cao, Q., Abdelzaher, T., Stankovic, J., & He, T. (2008). The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)* (pp. 233–244). USA: IEEE. http://doi.org/10.1109/IPSN.2008.54

Carboni, D. (2010). PYSENSE: PYTHON DECORATORS FOR WIRELESS SENSOR MACROPROGRAMMING. In *Proceedings of the 5th International Conference on Software and Data Technologies* (pp. 165–169). Athens, Greece: SciTePress - Science and and Technology Publications. http://doi.org/10.5220/0003038801650169

Carlson, L., & Richardson, L. (n.d.). *Ruby Cookbook*. (M. Loukides, Ed.) (1st ed.). O'Reilly Media.

Cecchinel, C., Mosser, S., & Collet, P. (2014). Software Development Support for Shared Sensing Infrastructures: A Generative and Dynamic Approach. In Ina Schaefer & I. Stamelos (Eds.), *Software Reuse for Dynamic Systems in the Cloud and Beyond* (Ina Schaef, pp. 221–236). Springer International Publishing. http://doi.org/10.1007/978-3-319-14130-5_16

Cecchinel, C., Mosser, S., & Collet, P. (2016). Automated Deployment of Data Collection Policies over Heterogeneous Shared Sensing Infrastructures. In *23rd Asia-Pacific Software Engi- neering Conference*. Hamilton, New Zealand: HAL. Retrieved from https://hal.archives-ouvertes.fr/hal-01411084

Ceriotti, M., Corra, M., D'Orazio, L., Doriguzzi, R., Facchin, D., Guna, S. T., … Torghele, C. (2011). Is there light at the ends of the tunnel? Wireless sensor networks for adaptive lighting in road tunnels. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks* (pp. 187–198). Chicago, IL, USA: IEEE.

Cha, H., Choi, S., Jung, I., Kim, H., Shin, H., Yoo, J., & Yoon, C. (2007). RETOS: Resilient, Expandable, and Threaded Operating System for Wireless Sensor Networks. In *2007 6th International Symposium on Information Processing in Sensor Networks* (pp. 148–157). Cambridge, MA, USA: IEEE. http://doi.org/10.1109/IPSN.2007.4379674

Chandra, T. B., & Dwivedi, A. K. (2015). Programming Languages for Wireless Sensor Networks : A Comparative Study. In *Computing for Sustainable Global Development (INDIACom), 2015 2nd International Conference on* (pp. 1702–1708). New Delhi: IEEE.

Chandrasekar, R. K., Uluagac, A. S., & Beyah, R. (2013). PROVIZ: An integrated visualization and programming framework for WSNs. In *38th Annual IEEE Conference on Local Computer Networks - Workshops* (pp. 146–149). Sydney, NSW, Australia: IEEE. http://doi.org/10.1109/LCNW.2013.6758511

Cheong, E., Liebman, J., Liu, J., & Zhao, F. (2003). TinyGALS: A Programming Model for Event-Driven Embedded Systems. In *Proceedings of the 2003 ACM symposium on Applied computing - SAC '03* (pp. 698–704). Melbourne, Florida, USA: ACM Press. http://doi.org/10.1145/952532.952668

Cheong, E., & Liu, J. (2005). galsC: A Language for Event-Driven Embedded Systems. In *Design, Automation and Test in Europe* (pp. 1050–1055). Munich, Germany: IEEE Comput. Soc. http://doi.org/10.1109/DATE.2005.165

Chung, W.-Y., Lee, Y.-D., & Jung, S.-J. (2008). A Wireless Sensor Network Compatible Wearable U-healthcare Monitoring System Using Integrated ECG, Accelerometer and

SpO2. In *30th Annual International Conference of the IEEEEngineering in Medicine and Biology Society, 2008. EMBS 2008.* (pp. 1529–1532). Vancouver, BC, Canada: IEEE. http://doi.org/10.1109/IEMBS.2008.4649460

Cox, E. P. (1980). The Optimal Number of Response Alternatives for a Scale: A Review. *Journal of Marketing Research*, *17*(4), 407–422. http://doi.org/10.2307/3150495

Creswell, J. W. (2009). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (3rd Editio). United States of America: SAGE Publications, Inc.

Cuenca, F., Bergh, J. Van den, Luyten, K., & Coninx, K. (2015). A User Study for Comparing the Programming Efficiency of Modifying Executable Multimodal Interaction Descriptions. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools - PLATEAU 2015* (pp. 31–38). Pittsburgh, PA, USA: ACM Press. http://doi.org/10.1145/2846680.2846686

Culler, D. (2003). 10 emerging technologies that will change your world. *MIT's Magazine of Innovation: Technology Review*, *106*(1), 37. Retrieved from http://www2.technologyreview.com/featured-story/401775/10-emerging-technologies-that-will-change-the/2/

Curtis, B., Kellner, M. I., & Over, J. (1992, September 1). Process modeling. *Communications of the ACM - Special Issue on Analysis and Modeling in Software Development*, *35*(9), 75–90. http://doi.org/10.1145/130994.130998

Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, *45*(3), 621–645. http://doi.org/10.1147/sj.453.0621

Damaševičius, R. (2006). On the Quantitative Estimation of Abstraction Level Increase in Metaprograms. *Computer Science and Information Systems*, *3*(1), 53–64. http://doi.org/10.2298/CSIS0601053D

Daniel, F., Eriksson, J., Finne, N., Fuchs, H., Karnouskos, S., Montero, P. M., … Voigt, T. (2013). makeSense : Real-world Business Processes through Wireless Sensor Networks. In G. Fortino, S. Karnouskos, P. J. Marrón, & J. L. M. Lastra (Eds.), *Proceedings of 4th International Workshop on Networks of Cooperating Objects for Smart Cities 2013 (CONET/UBICITEC 2013)* (pp. 58–72). Philadelphia, USA: CEUR-WS.org.

Dantas, P., Rodrigues, T., Batista, T., Delicato, F. C., Pires, P. F., Li, W., & Zomaya, A. Y. (2013). LWiSSy: A Domain Specific Language to Model Wireless Sensor and Actuators Network Systems. In *2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (pp. 7–12). San Francisco, USA: IEEE. http://doi.org/10.1109/SESENA.2013.6612258

De-Farias, C. M., Brito, I. C., Pirmez, L., Delicato, F. C., Pires, P. F., Rodrigues, T. C., … Batista, T. (2016). COMFIT: A development environment for the Internet of Things. *Future Generation Computer Systems*, (i). http://doi.org/10.1016/j.future.2016.06.031

De-Farias, C. M., Li, W., Delicato, F. C., Pirmez, L., Zomaya, A. Y., Pires, P. F., & Souza, J. N. De. (2016). A Systematic Review of Shared Sensor Networks. *ACM Computing Surveys*, *48*(4), 1–50. http://doi.org/10.1145/2851510

De Roover, C., Scholliers, C., Amerijckx, W., D'Hondt, T., & De Meuter, W. (2013). CrimeSPOT: A language and runtime for developing active wireless sensor network applications. *Science of Computer Programming*, *78*(10), 1951–1970. http://doi.org/10.1016/j.scico.2012.07.018

Delamo, M., Felici-Castell, S., Pérez-Solano, J. J., & Foster, A. (2015). Designing an open

source maintenance-free Environmental Monitoring Application for Wireless Sensor Networks. *Journal of Systems and Software*, *103*, 238–247. http://doi.org/10.1016/j.jss.2015.02.013

Delicato, F. C., Pires, P. F., & Zomaya, A. Y. (2014). Middleware Platforms: State of the Art, New Issues, and Future Trends. In H. M.Ammari (Ed.), *The Art of Wireless Sensor Networks (Volume 1: Fundamentals)* (Vol. 1, pp. 645–674). Springer Berlin Heidelberg.

Doddapaneni, K., Ever, E., Gemikonakli, O., Malavolta, I., Mostarda, L., & Muccini, H. (2012). A model-driven engineering framework for architecting and analysing Wireless Sensor Networks. In *2012 Third International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (pp. 1–7). Zurich, Switzerland: IEEE. http://doi.org/10.1109/SESENA.2012.6225729

Dunkels, A., Eriksson, J., Mottola, L., Voigt, T., Oppermann, F. J., Uzl, K. R., … Karnouskos, S. (2010). *D.1.1. Application and Programming Survey* (Technical Report).

Dunkels, A., Gronvall, B., & Voigt, T. (2004). Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks* (pp. 455–462). FL, USA: IEEE (Comput. Soc.). http://doi.org/10.1109/LCN.2004.38

Elsts, A. (2013). *A Framework to Facilitate Wireless Sensor Network Application Development* (Doctoral Thesis). University of Latvia. Retrieved from https://www.researchgate.net/publication/259575286

Elsts, A., Bijarbooneh, F., Jacobsson, M., & Sagonas, K. (2015). ProFuN TG: A Tool for Programming and Managing Dependable Sensor Network Applications. In *2015 IEEE 40th Local Computer Networks Conference Workshops (LCN Workshops)* (pp. 751–759). Clearwater Beach, FL, USA: IEEE. http://doi.org/10.1109/LCNW.2015.7365924

Elsts, A., Judvaitis, J., & Selavo, L. (2013). SEAL: A Domain-Specific Language for Novice Wireless Sensor Network Programmers. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications* (pp. 220–227). Santander, Spain: IEEE. http://doi.org/10.1109/SEAA.2013.16

Elsts, A., & Selavo, L. (2012). A user-centric approach to wireless sensor network programming languages. In *2012 Third International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (pp. 29–30). Zurich, Switzerland: IEEE. http://doi.org/10.1109/SESENA.2012.6225731

Elsts, A., & Selavo, L. (2013). Improving the Usability of Wireless Sensor Network Operating Systems. *Stud. Inform. Univ.*, *11*(1), 35–68. Retrieved from http://studia.complexica.net/Art/RI110103.pdf%5Cnpapers2://publication/uuid/AD48D3BD-D225-4FE0-8D54-3A593E7670E9

Essaadi, F., Ben Maissa, Y., & Dahchour, M. (2017). MDE-Based Languages for Wireless Sensor Networks Modeling: A Systematic Mapping Study. In E. Sabir, H. Medromi, & M. Sadik (Eds.), *Lecture Notes in Electrical Engineering* (Vol. 366, pp. 331–346). Singapore: Springer Singapore. http://doi.org/10.1007/978-981-10-1627-1_26

Eswaran, A., Rowe, A., & Rajkumar, R. (2005). Nano-RK: An Energy-Aware Resource-Centric RTOS for Sensor Networks. In *26th IEEE International Real-Time Systems Symposium (RTSS'05)* (pp. 256–265). Miami, FL, USA: IEEE. http://doi.org/10.1109/RTSS.2005.30

Eysholdt, M., & Behrens, H. (2010). Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM international conference companion on*

Object oriented programming systems languages and applications companion (pp. 307–309). Nevada, USA: ACM Press. http://doi.org/10.1145/1869542.1869625

Fajar, M., Nakanishi, T., Hisazumi, K., & Fukuda, A. (2012). A Decision Making Framework for Developing Agricultural Wireless Sensor Network Systems. In *Proc. 8th Asian Conf. for Information Technology in Agriculture (AFITA)*.

Fenton, N., & Bieman, J. (2014). *Software Metrics: A Rigorous and Practical Approach*. (R. LeBlanc, Ed.) (3rd ed.). Taylor & Francis Group, LLC. http://doi.org/10.1201/b17461

Ferro, G., Silva, R., & Lopes, L. (2015). Towards Out-of-the-Box Programming of Wireless Sensor-Actuator Networks. In *2015 IEEE 18th International Conference on Computational Science and Engineering* (pp. 110–119). Porto, Portugal: IEEE Comput. Soc. http://doi.org/10.1109/CSE.2015.20

Finstad, K. (2010). Response Interpolation and Scale Sensitivity: Evidence Against 5-Point Scales. *Journal of Usability Studies*, *5*(3), 104–110.

Flouri, K., Saukh, O., Sauter, R., Jalsan, K. E., Bischoff, R., Meyer, J., & Feltrin, G. (2012). A versatile software architecture for civil structure monitoring with wireless sensor networks. *Smart Structures and Systems*, *10*(3), 209–228. http://doi.org/10.12989/sss.2012.10.3.209

Fowler, M. (2010). *Domain-Specific Languages* (1 Edition). Addison-Wesley Professional. http://doi.org/10.1007/978-3-642-03034-5

Frey, J.-E. (2008). WISA - wireless control in theory, practice and production. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation* (pp. xvii–xvii). Hamburg, Germany: IEEE. http://doi.org/10.1109/ETFA.2008.4638352

Friedman, H. H., & Friedman, L. W. (1986). On the Danger of Using too few Points in a Rating Scale: A Test of Validity. *Journal of Data Collection*, *26*(2), 60–63. Retrieved from https://ssrn.com/abstract=2333162

Gaglione, A., Lo, B., & Yang, G. (2014). An Appstore Framework for Body Sensor Networks. In *11th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*. Zurich, Switzerland.

García-hernández, C. F., Ibargüengoytia-gonzález, P. H., García-hernández, J., & Pérez-díaz, J. A. (2007). Wireless sensor networks and applications: a survey. *International Journal of Computer Science and Network Security*, *7*(3), 264–273. Retrieved from http://paper.ijcsns.org/07_book/200703/20070338.pdf

Gay, D., Levis, P., Culler, D., & Brewer, E. (2009). *nesC 1.3 Language Reference Manual*. TinyOS Core Working Group. Retrieved from http://tinyprod.net/docs/nesc-ref.pdf

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., & Culler, D. (2003). The nesC language: A Holistic Approach to Networked Embedded Systems. In *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation - PLDI '03* (pp. 1–11). San Diego, California, USA: ACM. http://doi.org/10.1145/780822.781133

Geihs, K., Mottola, L., Picco, G. Pietro, & Römer, K. (2011). Second international workshop on software engineering for sensor network applications. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11* (pp. 1198–1199). Honolulu, HI, USA: IEEE. http://doi.org/10.1145/1985793.1986044

Geihs, K., Reichle, R., Wagner, M., & Khan, M. U. (2009). Modeling of Context-Aware Self-Adaptive Applications in Ubiquitous and Service-Oriented Environments. In B. H.

C. Cheng, R. de Lemos, H. Giese, P. Inverardi, & J. Magee (Eds.), *Software Engineering for Self-Adaptive Systems* (Vol. 5525 LNCS, pp. 146–163). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-02161-9_8

Getting Started with TinyOS. (n.d.). Retrieved from www.tinyos.net

Ghosh, D. (2011a). *DSLs in Action*. (C. Kane, Ed.). USA: Manning Publications Co.

Ghosh, D. (2011b, June 1). DSL for the Uninitiated. (M. Y. Vardi, Ed.)*Communications of the ACM*, *54*(7), 10. http://doi.org/10.1145/1989748.1989750

Goeken, M., & Alter, S. (2009). Towards Conceptual Metamodeling of IT Governance Frameworks Approach - Use - Benefits. In *2009 42nd Hawaii International Conference on System Sciences* (pp. 1–10). Big Island, HI, USA: IEEE. http://doi.org/10.1109/HICSS.2009.471

Gonzalez-Perez, C., McBride, T., & Henderson-Sellers, B. (2005). A Metamodel for Assessable Software Development Methodologies. *Software Quality Journal*, *13*(2), 195–214. http://doi.org/10.1007/s11219-005-6217-7

Gordon, D., & Beigl, M. (2009). D-Bridge: A platform for developing low-cost WSN product solutions. In *2009 Sixth International Conference on Networked Sensing Systems (INSS)* (pp. 1–4). Pittsburgh, PA, USA: IEEE. http://doi.org/10.1109/INSS.2009.5409950

Gordon, D., Beigl, M., & Neumann, M. A. (2010). dinam: A Wireless Sensor Network Concept and Platform for Rapid Development. In *2010 Seventh International Conference on Networked Sensing Systems (INSS)* (pp. 57–60). Kassel, Germany: IEEE. http://doi.org/10.1109/INSS.2010.5573290

Gu, Y., Ren, F., Ji, Y., & Li, J. (2016). The Evolution of Sink Mobility Management in Wireless Sensor Networks: A Survey. *IEEE Communications Surveys & Tutorials*, *18*(1), 507–524. http://doi.org/10.1109/COMST.2015.2388779

Hailpern, B., & Tarr, P. (2006). Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, *45*(3), 451–461. http://doi.org/10.1147/sj.453.0451

Handziski, V., Polastre, J., Hauer, J., Sharp, C., Wolisz, A., & Culler, D. (2005). Flexible hardware abstraction for wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.* (pp. 145–157). Istanbul, Turkey: IEEE. http://doi.org/10.1109/EWSN.2005.1462006

Handziski, V., Polastre, J., Hauer, J., Sharp, C., Wolisz, A., Culler, D., & Gay, D. (2004). *TEP 2: Hardware Abstraction Architecture* (2.x). *TinyOS Enhancement Proposal*. TinyOS Core Working Group.

Hansen, M., & Kusy, B. (2011). Cross-platform wireless sensor network development. In *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on* (pp. 153–154). Chigaco, IL, USA: IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5779091

Hansen, M. T., & Kusy, B. (2011). TinyInventor : A Holistic Approach to Sensor Network Application Development. In *Extending the Internet to Low power and Lossy Networks. IP+SN 2011* (pp. 153–154). Chigaco, IL, USA: ACM. Retrieved from http://www.forskningsdatabasen.dk/en/catalog/2185860199

Harrop, P. (2012). *Wireless Sensor Networks and the new Internet of Things*. *IDTechEx*. IDTechEx. Retrieved from http://www.idtechex.com/research/articles/wireless-sensor-networks-and-the-new-internet-of-things-00004770.asp

Harrop, P., & Das, R. (2012). *Wireless Sensor Networks 2011-2021: The new market for*

Ubiquitous Sensor Networks (USN). IDTechEx. Retrieved from http://www.idtechex.com/research/reports/wireless-sensor-networks-2011-2021-000275.asp

Hemel, Z., Kats, L. C. L., Groenewegen, D. M., & Visser, E. (2010). Code generation by model transformation: a case study in transformation modularity. *Software & Systems Modeling*, *9*(3), 375–402. http://doi.org/10.1007/s10270-009-0136-1

Henderson-Sellers, B., & Bulthuis, A. (1998). *Object-Oriented Metamethods*. New York, NY: Springer. http://doi.org/10.1007/978-1-4612-1748-0

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., & Pister, K. (2000, December 1). System Architecture Directions for Networked Sensors. *ACM SIGOPS Operating Systems Review*, pp. 93–104. New York, NY, USA: ACM. http://doi.org/10.1145/384264.379006

Hu, F., Jiang, M., Celentano, L., & Xiao, Y. (2008). Robust medical ad hoc sensor networks (MASN) with wavelet-based ECG data mining. *Ad Hoc Networks*, *6*(7), 986–1012. http://doi.org/10.1016/j.adhoc.2007.09.002

Hudak, P., Johnsson, T., Kieburtz, D., Nikhil, R., Partain, W., Peterson, J., … Hughes, J. (1992). Report on the programming language Haskell. *ACM SIGPLAN Notices*, *27*(5), 1–164. http://doi.org/10.1145/130697.130699

Hughes, D., Greenwood, P., Blair, G., Coulson, G., Grace, P., Pappenberger, F., … Beven, K. (2008). An experiment with reflective middleware to support grid- based flood monitoring. *Concurrency and Computation: Practice and Experience*, *20*(11), 1303–1316. http://doi.org/10.1002/cpe.1279

Hughes, D., Thoelen, K., Maerien, J., Matthys, N., Horre, W., Del Cid, J., … Joosen, W. (2012). LooCI: The Loosely-coupled Component Infrastructure. In *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium* (pp. 236–243). Cambridge, MA, USA: IEEE. http://doi.org/10.1109/NCA.2012.30

Hussain, A., & Ferneley, E. (2008). Usability metric for mobile application. In *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services - iiWAS '08* (pp. 567–570). New York, New York, USA: ACM Press. http://doi.org/10.1145/1497308.1497412

Intana, A. (2015). *Formal Engineering Methodologies for Wireless Sensor Network Development with Simulation* (Doctoral Thesis). University of Southampton, UK. Retrieved from http://eprints.soton.ac.uk/id/eprint/387248

Intana, A., Poppleton, M. R., & Merrett, G. V. (2014). A Formal Co-Simulation Approach forWireless Sensor Network Development. In *Proceedings of the 14th InternationalWorkshop on Automated Verification of Critical Systems (AVoCS 2014)* (pp. 1–15). Electronic Communications of the EASST. http://doi.org/10.14279/tuj.eceasst.70.969

Intana, A., Poppleton, M. R., & Merrett, G. V. (2015). A Model-based Trace Testing Approach for Validation of Formal Co-simulation Models. In *DEVS '15 Proceedings of the Symposium on Theory of Modeling & Simulation* (pp. 181–188). Alexandria, Virginia, USA.

ISO/IEC. (1996). *Information technology — Syntactic metalanguage — Extended BNF* (ISO/IEC 14977). Geneva: The British Standards Institution. Retrieved from http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

ISO/IEC. (2011). *Systems and software Quality Requirements and Evaluation (SQuaRE)* (ISO/IEC 25010). Switzerland: The British Standards Institution. Retrieved from https://bsol-bsigroup-com.lcproxy.shu.ac.uk/Bibliographic/BibliographicInfoData/000000000030215101

ISO/IEC. (2013). *Information technology — Sensor networks: Sensor Network Reference Architecture (SNRA) Part 4: Entity models* (ISO/IEC 29182-4). Switzerland: The British Standards Institution.

ISO/IEC. (2014a). *Information technology — Sensor networks: Sensor Network Reference Architecture (SNRA) — Part 3: Reference architecture views* (ISO/IEC 29182-3). Switzerland: The British Standards Institution.

ISO/IEC. (2014b). *Information technology — Sensor networks: Sensor Network Reference Architecture (SNRA) — Part 6: Applications* (ISO/IEC 29182-6). Switzerland: The British Standards Institution.

Jedlitschka, A., Ciolkowski, M., & Pfahl, D. (2008). Reporting Experiments in Software Engineering. In F. Shull, J. Singer, & D. I. K. Sjøberg (Eds.), *Guide to Advanced Empirical Software Engineering* (pp. 201–228). London: Springer-Verlag London Limited. http://doi.org/10.1007/978-1-84800-044-5_8

Jie Liu, Chu, M., Liu, J., Reich, J., & Feng Zhao. (2003). State-Centric Programming for Sensor-Actuator Network Systems. *IEEE Pervasive Computing*, *2*(4), 50–62. http://doi.org/10.1109/MPRV.2003.1251169

Jörges, S. (2013). *Construction and Evolution of Code Generators: A Model-Driven and Service-Oriented Approach*. Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-36127-2

Julien, C., & Wehrle, K. (2013). 4th International Workshop on Software Engineering for sensor network applications (SESENA 2013). In *2013 35th International Conference on Software Engineering (ICSE)* (pp. 1551–1552). San Francisco, CA, USA: IEEE. http://doi.org/10.1109/ICSE.2013.6606782

Kabac, M., Consel, C., & Volanschi, N. (2016). Leveraging Declarations over the Lifecycle of Large-Scale Sensor Applications. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)* (pp. 211–219). Toulouse, France: IEEE. http://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0051

Kahraman, G., & Bilgen, S. (2015). A framework for qualitative assessment of domain-specific languages. *Software & Systems Modeling*, *14*(4), 1505–1526. http://doi.org/10.1007/s10270-013-0387-8

Keshavarz, G. (2011). Metric for Early Measurement of Software Complexity. *International Journal on Computer Science and Engineering (IJCSE)*, *3*(6), 2482–2490.

Khedo, K. K., & Subramanian, R. K. (2009). A Service Oriented Component-Based Middleware Architecture for Wireless Sensor Networks. *International Journal of Computer Science and Network Security (IJCSNS)*, *9*(3). Retrieved from http://paper.ijcsns.org/07_book/200903/20090324.pdf

Khemapech, I., Miller, A., & Duncan, I. (2005). *Simulating Wireless Sensor Networks* (Technical Report). University of St Andrews. Retrieved from http://ahvaz.ist.unomaha.edu/azad/temp/sac/05-khemapech-simulation-wirless-sensor-

networks.pdf

Kirbaş, I., & Bayilmiş, C. (2012). HealthFace: A web-based remote monitoring interface for medical healthcare systems based on a wireless body area sensor network. *Turkish Journal of Electrical Engineering and Computer Sciences*, *20*(4), 629–638. http://doi.org/10.3906/elk-1011-934

Klatt, B. (2007). *Xpand: A Closer Look at the model2text Transformation Language* (Technical Report). Germany: Chair for Software Design and Quality (SDQ), Institute for Program Structures and Data Organization (IPD), University of Karlsruhe.

Kollár, J., Pietriková, E., & Chodarev, S. (2012). Abstraction in programming languages according to domain-specific patterns. *Acta Electrotechnica et Informatica*, *12*(2), 9–15. http://doi.org/10.2478/v10198-012-0017-3

Kolovos, D. S., García-Domínguez, A., Rose, L. M., & Paige, R. F. (2017). Eugenia: towards disciplined and automated development of GMF-based graphical model editors. *Software & Systems Modeling*, *16*(1), 229–255. http://doi.org/10.1007/s10270-015-0455-3

Kosar, T., Martinez López, P. E., Barrientos, P. A., & Mernik, M. (2008). A preliminary study on various implementation approaches of domain-specific language. *Information and Software Technology*, *50*(5), 390–405. http://doi.org/10.1016/j.infsof.2007.04.002

Kouche, A. El. (2013). *Wireless Sensor Network Platform for Harsh Industrial Environments* (Doctoral Thesis). Queen's University, Ontario, Canada. Retrieved from http://hdl.handle.net/1974/8342

Kovatsch, M., Mayer, S., & Ostermaier, B. (2012). Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing* (pp. 751–756). IEEE. http://doi.org/10.1109/IMIS.2012.104

Koziolek, H. (2008). Goal, Question, Metric. In I. Eusgeld, F. C. Freiling, & R. Reussner (Eds.), *Dependability Metrics* (pp. 39–42). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-68947-8_6

Kumar, S. A. A., & Simonsen, K. I. F. (2014). Towards a Model-Based Development Approach for Wireless Sensor-Actuator Network Protocols Position. In *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems - CyPhy '14* (pp. 35–39). Berlin, Germany: ACM Press. http://doi.org/10.1145/2593458.2593465

Kumar S., A. A., Ovsthus, K., & Kristensen., L. M. (2014). An Industrial Perspective on Wireless Sensor Networks — A Survey of Requirements, Protocols, and Challenges. *IEEE Communications Surveys & Tutorials*, *16*(3), 1391–1412. http://doi.org/10.1109/SURV.2014.012114.00058

Kunii, T. L., & Hisada, M. (2000). Overcoming Software Complexity by Constructing Abstraction Hierarchies - The Principles and Applications. In *Proceedings Sixth IEEE International Conference on Engineering of Complex Computer Systems. ICECCS 2000* (pp. 126–130). Tokyo, Japan: IEEE Comput. Soc. http://doi.org/10.1109/ICECCS.2000.873936

Kurpjuweit, S., & Winter, R. (2007). Viewpoint-based Meta Model Engineering. In Manfred Reichert, S. Strecker, & K. Turowski (Eds.), *the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures* (pp. 143–158). Germany: Gesellschaft für Informatik. Retrieved from

http://stl.mie.utoronto.ca/publications/aimag-em.pdf

Kurtz, T. E. (1978, August 1). BASIC. *ACM SIGPLAN Notices - Special Issue: History of Programming Languages Conference*, pp. 103–118. http://doi.org/10.1145/960118.808376

Kushwaha, M., Amundson, I., Koutsoukos, X., Neema, S., & Sztipanovits, J. (2007). OASiS: A Programming Framework for Service-Oriented Sensor Networks. In *2007 2nd International Conference on Communication Systems Software and Middleware* (pp. 1–8). Bangalore, India: IEEE. http://doi.org/10.1109/COMSWA.2007.382431

Kyungtae Kang, Min-Young Nam, & Lui Sha. (2013). Model-Based Analysis of Wireless System Architectures for Real-Time Applications. *IEEE Transactions on Mobile Computing*, *12*(2), 219–232. http://doi.org/10.1109/TMC.2011.260

Lange, S., Lösche, J., & Piotrowski, K. (2014). Tool-supported Requirements-based Topology Design for Wireless Sensor Networks. In *the 2014 Federated Conference on Computer Science and Information Systems* (pp. 1043–1047). Warsaw, Poland: IEEE. http://doi.org/10.15439/2014F210

Lárusdóttir, M. K., & Ármannsdóttir, S. E. (2005). A Case Study of Software Replacement. In *Proceedings of the international conference on software development* (pp. 129–140). Reykjavik, Iceland.

Laukkarinen, T., Suhonen, J., & Hännikäinen, M. (2012). A Survey of Wireless Sensor Network Abstraction for Application Development. *International Journal of Distributed Sensor Networks*, *8*(12), 740268. http://doi.org/10.1155/2012/740268

Lazarescu, M. (2017). Wireless Sensor Networks for the Internet of Things: Barriers and Synergies. In *Components and Services for IoT Platforms* (pp. 155–186). Cham: Springer International Publishing. http://doi.org/10.1007/978-3-319-42304-3_9

Lazarescu, M., & Lavagno, L. (2016). Wireless Sensor Networks. In S. Ha & J. Teich (Eds.), *Handbook of Hardware/Software Codesign* (pp. 1–42). Dordrecht: Springer Netherlands. http://doi.org/10.1007/978-94-017-7358-4_38-1

Leelavathi, G., Shaila, K., Venugopal, K. R., & Patnaik, L. M. (2013). Design Issues on Software Aspects and Simulation Tools for Wireless Sensor Networks. *International Journal of Network Security & Its Applications*, *5*(2), 47–64. http://doi.org/10.5121/ijnsa.2013.5204

Levis, P. (2006). *TinyOS Programming*. Retrieved from http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf

Levis, P. (2012). Experiences from a Decade of TinyOS Development. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)* (pp. 207–220). Hollywood, CA, USA: USENIX Association. Retrieved from http://dl.acm.org/citation.cfm?id=2387880.2387901

Levis, P., & Culler, D. (2002). Maté: A Tiny Virtual Machine for Sensor Networks. In *ASPLOS X Proceedings of the 10th international conference on Architectural support for programming languages and operating systems* (pp. 85–95). San Jose, California, USA: ACM Press. http://doi.org/10.1145/605397.605407

Levis, P., & Gay, D. (2009). *TinyOS Programming*. Cambridge: Cambridge University Press.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., … Culler, D. (2005). TinyOS: An Operating System for Sensor Networks. In W. Weber, J. M. Rabaey, & E. Aarts (Eds.), *Ambient Intelligence* (pp. 115–148). Berlin/Heidelberg:

Springer-Verlag Berlin Heidelberg. http://doi.org/10.1007/3-540-27139-2_7

Liu, T., & Martonosi, M. (2003, October 1). Impala: A Middleware System for Managing Autonomic, Parallel Sensor Systems. *ACM SIGPLAN Notices*, pp. 107–118. http://doi.org/10.1145/966049.781516

Lopes, L., & Martins, F. (2016). A safe-by-design programming language for wireless sensor networks. *Journal of Systems Architecture*, *63*, 16–32. http://doi.org/10.1016/j.sysarc.2016.01.004

Losilla, F., Vicente-Chicote, C., Álvarez, B., Iborra, A., & Sánchez, P. (2007). Wireless Sensor Network Application Development: An Architecture-Centric MDE Approach. In F. Oquendo (Ed.), *Software Architecture* (pp. 179–194). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-75132-8_15

Luo, L., Abdelzaher, T. F., He, T., & Stankovic, J. A. (2006). EnviroSuite: An Environmentally Immersive Programming Framework for Sensor Networks. *ACM Transactions on Embedded Computing Systems*, *5*(3), 543–576. http://doi.org/10.1145/1165780.1165782

Mahmood, M. A., Seah, W. K. G., & Welch, I. (2015). Reliability in wireless sensor networks: A survey and challenges ahead. *Computer Networks*, *79*, 166–187. http://doi.org/10.1016/j.comnet.2014.12.016

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., & Anderson, J. (2002). Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications - WSNA '02* (p. 88). New York, New York, USA: ACM Press. http://doi.org/10.1145/570738.570751

Maissa, Y. B., Kordon, F., Mouline, S., & Thierry-Mieg, Y. (2012). Modeling and analyzing wireless sensor networks with VeriSensor. In *PNSE'12 International Workshop on Petri Nets and Software Engineering* (Vol. 851, pp. 60–76). Hamburg, Germany.

Malan, D., Fulford-Jones, T., Welsh, M., & Moulton, S. (2004). Codeblue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In *International Workshop on Wearable and Implantable Body Sensor Networks* (pp. 12–14). London, UK. Retrieved from http://icawww.epfl.ch/luo/WAMES 2004_files/WAMESproceedings.pdf#page=12

Malavolta, I., & Muccini, H. (2014). A Study on MDE Approaches for Engineering Wireless Sensor Networks. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 149–157). Verona, Italy: IEEE. http://doi.org/10.1109/SEAA.2014.61

Marques, E., Balegas, V., Barroca, B. F., Barisic, A., & Amaral, V. (2012). The RPG DSL: A Case Study of Language Engineering using MDD for Generating RPG Games for Mobile Phones. In *DSM '12 Proceedings of the 2012 workshop on Domain-specific modeling* (pp. 13–18). Tucson, Arizona, USA: ACM Press. http://doi.org/10.1145/2420918.2420923

Marques, I. L., da Silva Teofilo, M. R., & Rosa, N. S. (2013). Durin: A development environment for Wireless Sensor Network. In *2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (pp. 19–23). San Francisco, CA, USA: IEEE. http://doi.org/10.1109/SESENA.2013.6612260

Matthys, N., Huygens, C., Hughes, D., Michiels, S., & Joosen, W. (2012). A Component and Policy-Based Approach for Efficient Sensor Network Reconfiguration. In *2012 IEEE International Symposium on Policies for Distributed Systems and Networks* (pp. 53–60).

Chapel Hill, NC, USA: IEEE. http://doi.org/10.1109/POLICY.2012.17

Meana-Llorián, D., González García, C., Pelayo G-Bustelo, B. C., Cueva Lovelle, J. M., & Garcia-Fernandez, N. (2016). IoFClime: The fuzzy logic and the Internet of Things to control indoor temperature regarding the outdoor ambient conditions. *Future Generation Computer Systems*. http://doi.org/10.1016/j.future.2016.11.020

Mellor, S. J., Clark, A. N., & Futagami, T. (2003). Model-driven development - Guest editor's introduction. *IEEE Software*, *20*(5), 14–18. http://doi.org/10.1109/MS.2003.1231145

Mellor, S. J., Scott, K., Uhl, A., & Weise, D. (2004). *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley.

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, *37*(4), 316–344. http://doi.org/10.1145/1118890.1118892

Miller, J. S., Dinda, P. a., & Dick, R. P. (2009). Evaluating a BASIC approach to sensor network node programming. In *the 7th ACM Conference on Embedded Networked Sensor Systems* (pp. 155–168). Berkeley, California, USA: ACM. http://doi.org/10.1145/1644038.1644054

Mohamed, N., & Al-Jaroodi, J. (2011). A survey on service-oriented middleware for wireless sensor networks. *Service Oriented Computing and Applications*, *5*(2), 71–85. http://doi.org/10.1007/s11761-011-0083-x

Mohorcic, M., Smolnikar, M., & Javornik, T. (2013). Wireless Sensor Network Based Infrastructure for Experimentally Driven Research. In *Wireless Communication Systems (ISWCS 2013), Proceedings of the Tenth International Symposium on* (pp. 375–379). Ilmenau, Germany: VDE.

Moody, D. L. (2005). Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. *Data & Knowledge Engineering*, *55*(3), 243–276. http://doi.org/10.1016/j.datak.2004.12.005

Mottola, L., & Picco, G. Pietro. (2011). Programming wireless sensor networks. *ACM Computing Surveys (CSUR)*, *43*(3), 1–51. http://doi.org/10.1145/1922649.1922656

Mottola, L., & Picco, G. Pietro. (2012). Middleware for wireless sensor networks: an outlook. *Journal of Internet Services and Applications*, *3*(1), 31–39. http://doi.org/10.1007/s13174-011-0046-7

Mozumdar, M. M. R., Gregoretti, F., Lavagno, L., Vanzago, L., & Olivieri, S. (2008). A Framework for Modeling, Simulation and Automatic Code Generation of Sensor Network Application. In *5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08.* (pp. 515–522). San Francisco, California, USA: IEEE. http://doi.org/10.1109/SAHCN.2008.68

Nachman, L., Huang, J., Shahabdeen, J., Adler, R., & Kling, R. (2008). IMOTE2: Serious Computation at the Edge. In *2008 International Wireless Communications and Mobile Computing Conference* (pp. 1118–1123). Crete Island, Greece: IEEE. http://doi.org/10.1109/IWCMC.2008.194

Naiditch, D. (1999). Selecting a Programming Language for Your Project. *IEEE Aerospace and Electronic Systems Magazine*, *14*(9), 11–14. http://doi.org/10.1109/62.793447

Nakamura, E. F., Loureiro, A. a. F., & Frery, A. C. (2007). Information fusion for wireless sensor networks. *ACM Computing Surveys*, *39*(3).

http://doi.org/10.1145/1267070.1267073

Naumowicz, T., Schröter, B., & Schiller, J. (2009). Prototyping a software factory for wireless sensor networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems - SenSys '09* (pp. 369–370). Berkeley, California, USA: ACM Press. http://doi.org/10.1145/1644038.1644106

Nguyen, X. T., Tran, H. T., Baraki, H., & Geihs, K. (2015). FRASAD: A framework for model-driven IoT Application Development. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)* (pp. 387–392). IEEE. http://doi.org/10.1109/WF-IoT.2015.7389085

Novikov, A. M., & Novikov, D. A. (2013). *Research Methodology: From Philosophy of Science to Research Design*. (J.-L. Forrest, Ed.). Taylor & Francis Group, LLC.

Oldevik, J., Neple, T., Grønmo, R., Aagedal, J., & Berre, A.-J. (2005). Toward Standardised Model to Text Transformations. In K. D. Hartman A. (Ed.), *Model Driven Architecture – Foundations and Applications* (pp. 239–253). Springer, Berlin, Heidelberg. http://doi.org/10.1007/11581741_18

Oppermann, F. J., Boano, C. A., & Römer, K. (2014). A Decade of Wireless Sensing Applications: Survey and Taxonomy. In H. M. Ammari (Ed.), *The Art of Wireless Sensor Networks* (Vol. 1, pp. 11–50). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-40009-4_2

Oppermann, F. J., Romer, K., Mottola, L., Picco, G. Pietro, & Gaglione, A. (2014). Design and compilation of an object-oriented macroprogramming language for wireless sensor networks. In *39th Annual IEEE Conference on Local Computer Networks Workshops* (pp. 574–582). Edmonton, AB, Canada: IEEE. http://doi.org/10.1109/LCNW.2014.6927705

Oshana, R., & Kareling, M. (2013). Software Engineering for Embedded Systems: Methods, Practical Techniques and Applications. Elsevier Inc.

Ouadjaout, A., Miné, A., Lasla, N., & Badache, N. (2016). Static analysis by abstract interpretation of functional properties of device drivers in TinyOS. *Journal of Systems and Software*, *120*, 114–132. http://doi.org/10.1016/j.jss.2016.07.030

Parr, T. (2007). *The Definitive ANTLR Reference: Building Domain-Specific Languages*. *The Pragmatic Bookshelf*. The Pragmatic Bookshelf. Retrieved from http://www.pragprog.com/titles/tpantlr/the-definitive-antlr-reference

Peter, S., & Langendorfer, P. (2012). Tool-Supported Methodology for Component-Based Design of Wireless Sensor Network Applications. In *2012 IEEE 36th Annual Computer Software and Applications Conference Workshops* (pp. 526–531). IEEE. http://doi.org/10.1109/COMPSACW.2012.98

Picco, G. Pietro. (2010). Software Engineering and Wireless Sensor Networks: Happy Marriage or Consensual Divorce? In *Proceedings of the FSE/SDP workshop on Future of software engineering research - FoSER '10* (pp. 283–286). Santa Fe, New Mexico, USA: ACM Press. http://doi.org/10.1145/1882362.1882421

Piotrowski, K., & Peter, S. (2013). Sens4U: Wireless Sensor Network Applications for Environment Monitoring Made Easy. In *2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (pp. 37–42). San Francisco, CA, USA: IEEE. http://doi.org/10.1109/SESENA.2013.6612264

Polastre, J., Szewczyk, R., & Culler, D. (2005). Telos: enabling ultra-low power wireless

research. In *IIPSN '05 Proceedings of the 4th international symposium on Information processing in sensor networks* (pp. 364–369). Los Angeles, California, USA: IEEE. http://doi.org/10.1109/IPSN.2005.1440950

Productivity Commission. (2013). *On efficiency and effectiveness : some definitions.* COMMONWEALTH OF AUSTRALIA.

Randhawa, S. (2014). Research Challenges in Wireless Sensor Network: A State of the Play. In *Conference Proceeding of National conference on convergence of science, engineering & management in education and research*. PUNJAB, INDIA. Retrieved from http://arxiv.org/abs/1404.1469

Ravichandran, S., Chandrasekar, R. K., Selcuk Uluagac, A., & Beyah, R. (2016). A simple visualization and programming framework for wireless sensor networks: PROVIZ. *Ad Hoc Networks*, *53*, 1–16. http://doi.org/10.1016/j.adhoc.2016.06.015

Rawat, P., Singh, K. D., Chaouchi, H., & Bonnin, J. M. (2014). Wireless sensor networks: a survey on recent developments and potential synergies. *The Journal of Supercomputing*, *68*(1), 1–48. http://doi.org/10.1007/s11227-013-1021-9

Riva, O., & Borcea, C. (2007). The Urbanet Revolution: Sensor Power to the People! *IEEE Pervasive Computing*, *6*(2), 41–49. http://doi.org/10.1109/MPRV.2007.46

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*(2), 137–172. http://doi.org/10.1076/csed.13.2.137.14200

Rodrigues, T., Batista, T., Delicato, F. C., & Pires, P. de F. (2015). Architecture-Driven Development Approach for WSAN Applications. In *2015 IEEE 13th International Conference on Embedded and Ubiquitous Computing* (pp. 68–75). Porto, Portugal: IEEE. http://doi.org/10.1109/EUC.2015.15

Rodrigues, T., Batista, T., Delicato, F. C., Pires, P. F., & Zomaya, A. Y. (2013). Model-driven approach for building efficient Wireless Sensor and Actuator Network applications. In *2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (pp. 43–48). San Francisco, CA, USA: IEEE. http://doi.org/10.1109/SESENA.2013.6612265

Rodrigues, T., Dantas, P., Delicato, F. C., Pires, P. F., Pirmez, L., Batista, T., … Zomaya, A. (2011). Model-Driven Development of Wireless Sensor Network Applications. In *2011 IFIP 9th International Conference on Embedded and Ubiquitous Computing* (pp. 11–18). Melbourne, VIC, Australia: IEEE. http://doi.org/10.1109/EUC.2011.50

Rodrigues, T., Delicato, F. C., Batista, T., Pires, P. F., & Pirmez, L. (2015). An approach based on the domain perspective to develop WSAN applications. *Software & Systems Modeling*, *1*(1), 1–29. http://doi.org/10.1007/s10270-015-0498-5

Romer, K., & Mattern, F. (2004). The design space of wireless sensor networks. *IEEE Wireless Communications*, *11*(6), 54–61. http://doi.org/10.1109/MWC.2004.1368897

Rothenberg, J. (1989). The Nature of Modeling. In L. E. Widman, K. A. Loparo, & N. R. Nielsen (Eds.), *AI, Simulation & Modeling* (p. 75−92). John Wiley & Sons. Retrieved from www.rand.org/pubs/notes/2007/N3027.pdf

Sadilek, D. (2007). Prototyping Domain-Specific Languages for Wireless Sensor Networks. In *Proc. of the 4th Int. Workshop on Software Language Engineering*. Retrieved from http://www2.informatik.hu-berlin.de/~sadilek/publications/2007/ATEM07prototypingDSLs.pdf

Sadilek, D. (2008). Domain-Specific Languages for Wireless Sensor Networks. In T. Kuehne, W. Reisig, & F. Steimann (Eds.), *Modellierung* (pp. 237–242). Berlin, Germany: Gesellschaft fuer Informatik. Retrieved from http://subs.emis.de/LNI/Proceedings/Proceedings127/article2121.html

Salman, A. J., & Al-Yasiri, A. (2016a). Developing Domain-Specific Language for Wireless Sensor Network application development. In *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)* (pp. 301–308). Barcelona, Spain: IEEE. http://doi.org/10.1109/ICITST.2016.7856718

Salman, A. J., & Al-Yasiri, A. (2016b). SenNet: A Programming Toolkit to Develop Wireless Sensor Network Applications. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)* (pp. 1–7). Larnaca, Cyprus: IEEE. http://doi.org/10.1109/NTMS.2016.7792476

Santos, J. R. A. (1999). Cronbach's alpha: A tool for assessing the reliability of scales. *Journal of Extension*, *37*(2), 1–5. Retrieved from http://www.joe.org/joe/1999april/tt3.php?ref

Schmitt, C., Kuckuk, S., Kostler, H., Hannig, F., & Teich, J. (2014). An Evaluation of Domain-Specific Language Technologies for Code Generation. In *2014 14th International Conference on Computational Science and Its Applications* (pp. 18–26). Guimaraes, Portugal: IEEE. http://doi.org/10.1109/ICCSA.2014.16

Schuette, R., & Rotthowe, T. (1998). The Guidelines of Modeling – An Approach to Enhance the Quality in Information Models. In L. TW., R. S., & L. L. M. (Eds.), *International Conference on Conceptual Modeling* (pp. 240–254). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-49524-6_20

Seidewitz, E. (2002). What Models Means. *IEEE Software*, *20*(5), 26–32. http://doi.org/10.1109/MS.2003.1231147

Selic, B. (2003). The Pragmatics of Model-Driven Development. *IEEE Software*, *20*(5), 19–25. http://doi.org/10.1109/MS.2003.1231146

Serna, M. A., Sreenan, C. J., & Fedor, S. (2015). A visual programming framework for wireless sensor networks in smart home applications. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)* (pp. 1–6). Singapore, Singapore: IEEE. http://doi.org/10.1109/ISSNIP.2015.7106946

Shimizu, R., Tei, K., Fukazawa, Y., & Honiden, S. (2012). Case Studies on the Development ofWireless Sensor Network Applications Using Multiple Abstraction Levels. In *2012 Third International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (pp. 22–28). Zurich, Switzerland: IEEE. http://doi.org/10.1109/SESENA.2012.6225730

Shimizu, R., Tei, K., Fukazawa, Y., & Honiden, S. (2014). Toward A Portability Framework with Multi-Level Models for Wireless Sensor Network Software. In *2014 International Conference on Smart Computing* (pp. 253–260). Hong Kong, China: IEEE. http://doi.org/10.1109/SMARTCOMP.2014.7043866

Shimizu, R., Tei, K., Fukazawa, Y., & Shinichi, S. (2011). Model driven development for rapid prototyping and optimization of wireless sensor network applications. In *Proceeding of the 2nd workshop on Software engineering for sensor network applications - SESENA '11* (pp. 31–36). Waikiki, Honolulu, HI, USA: ACM Press. http://doi.org/10.1145/1988051.1988058

Shull, F., Singer, J., & Sjøberg, D. I. K. (2008). *Guide to Advanced Empirical Software Engineering* (Vol. 1). Springer-Verlag London Limited. http://doi.org/10.1017/CBO9781107415324.004

Simon, D., Cifuentes, C., Cleal, D., Daniels, J., & White, D. (2006). Java™ on the bare metal of wireless sensor devices. In *Proceedings of the 2nd international conference on Virtual execution environments - VEE '06* (pp. 78–88). Ottawa, Ontario, Canada: ACM Press. http://doi.org/10.1145/1134760.1134773

Singh, A. P., Vyas, O. P., & Varma, S. (2014). Flexible Service Oriented Network Architecture for Wireless Sensor Networks. *International Journal of Computers Communications & Control (IJCCC)*, *9*(5), 610–622.

Skubch, H., Wagner, M., Reichle, R., & Geihs, K. (2011). A Modelling Language for Cooperative Plans in Highly Dynamic Domains. *Mechatronics*, *21*(2), 423–433. http://doi.org/10.1016/j.mechatronics.2010.10.006

Smith, R. B. (2007). SPOTWorld and the Sun SPOT. In *Proceedings of the 6th international conference on Information processing in sensor networks - IPSN '07* (pp. 5650–566). Cambridge, Massachusetts, USA: ACM Press. http://doi.org/10.1145/1236360.1236442

Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2008). *EMF: Eclipse Modeling Framework*. (E. Gamma, L. Nackman, & J. Wiegand, Eds.) (2nd ed.). Addison-Wesley Professional.

Stephan, M., & Cordy, J. R. (2012). *A Survey of Methods and Applications of Model Comparison* (Technical Report 2011-582). *Queen's University*. Kingston, Ontario, Canada.

Strazdins, G., & Selavo, L. (2014). Wireless Sensor Network Software Design Rules. *Baltic Journal of Modern Computing*, *2*(2), 84–115.

Sugihara, R., & Gupta, R. K. (2008). Programming models for sensor networks. *ACM Transactions on Sensor Networks*, *4*(2), 1–29. http://doi.org/10.1145/1340771.1340774

Sun, Y., Demirezen, Z., Mernik, M., Gray, J., & Bryant, B. (2008). Is My DSL a Modeling or Programming Language? In J. Lawall & L. Réveillère (Eds.), *Domain-Specific Program Development*. Nashville, United States. Retrieved from https://hal.archives-ouvertes.fr/hal-00350257

Taherkordi, A., Eliassen, F., & Johnsen, E. B. (2013). Behavioural design of sensor network applications using activity-driven states. In *2013 4th International Workshop on Software Engineering for Sensor Network Applications (SESENA)* (pp. 13–18). San Francisco, CA, USA: IEEE. http://doi.org/10.1109/SESENA.2013.6612259

Tei, K., Shimizu, R., Fukazawa, Y., & Honiden, S. (2015). Model-Driven-Development-Based Stepwise Software Development Process for Wireless Sensor Networks. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *45*(4), 675–687. http://doi.org/10.1109/TSMC.2014.2360506

Texas Instruments. (2008). *MSP430 Microcontroller Datasheet* (No. SLAS241). Retrieved from http://www.ti.com/lit/ds/slas241i/slas241i.pdf

Thang, N. X. (2015). *Model-driven development of sensor network applications with optimization of non- functional constraints* (Doctoral Thesis). University of Kassel, Germany.

The TinyOS 2.x Working Group. (2005). TinyOS 2.0. In *Proceedings of the 3rd international conference on Embedded networked sensor systems - SenSys '05* (p. 320). San Diego,

California, USA: ACM Press. http://doi.org/10.1145/1098918.1098985

Tranquillini, S., Spieß, P., Daniel, F., Karnouskos, S., Casati, F., Oertel, N., … Voigt, T. (2012). Process-Based Design and Integration of Wireless Sensor Network Applications. In Barros A., G. A., & K. E. (Eds.), *Business Process Management* (pp. 134–149). Springer, Berlin, Heidelberg. http://doi.org/10.1007/978-3-642-32885-5_10

Tullis, T., & Albert, B. (2013). *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. (M. Dunkerley, Ed.) (2nd ed.). Elsevier Inc.

Tunc, H., Taddese, A., Volgyesi, P., Sallai, J., Valdastri, P., & Ledeczi, A. (2016). Web-based integrated development environment for event-driven applications. In *SoutheastCon 2016* (pp. 1–8). Norfolk, VA, USA: IEEE. http://doi.org/10.1109/SECON.2016.7506646

Valero, M., Uluagac, S., Venkatachalam, S., Ramalingam, K. C., & Beyah, R. (2012). The Monitoring Core: A framework for sensor security application development. In *2012 IEEE 9th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS 2012)* (pp. 263–271). Las Vegas, NV, USA: IEEE. http://doi.org/10.1109/MASS.2012.6502525

Venčkauskas, A., Štuikys, V., Jusas, N., & Burbaitė, R. (2016). Model-Driven Approach for Body Area Network Application Development. *Sensors*, *16*(5), 670. http://doi.org/10.3390/s16050670

Vicente-chicote, C., Losilla, F., Alvarez, B., Iborra, A., & Sanchez, P. (2007). Applying MDE to the development of flexible and reusable wireless sensor networks. *International Journal of Cooperative Information Systems*, *16*, 393–412. http://doi.org/10.1142/S021884300700172X

Vieira, L. F. M., Vitorino, B. A. D., Vieira, M. A. M., Silva, D. C., Fernandes, A. O., & Loureiro, A. A. F. (2005). WISDOM: a Visual Development Framework for Multi-platform Wireless Sensor Networks. In *2005 IEEE Conference on Emerging Technologies and Factory Automation* (Vol. 2, pp. 527–534). Catania, Italy: IEEE. http://doi.org/10.1109/ETFA.2005.1612721

Voelter, M., Benz, S., Dietrich, C., Engelmann, B., Helander, M., Kats, L., … Wachsmuth, G. (2013). *DSL Engineering: Designing, Implementing and Using Domain-Specific Languages*. dslbook.org. Retrieved from http://publications.st.ewi.tudelft.nl/bibtex/2551?noheader=1

Volter, M. (2011). From Programming to Modeling - and Back Again. *IEEE Software*, *28*(6), 20–25. http://doi.org/10.1109/MS.2011.139

Walker, Z., Moh, M., & Moh, T.-S. (2007). A Development Platform for Wireless Sensor Networks with Biomedical Applications. In *2007 4th IEEE Consumer Communications and Networking Conference* (pp. 768–772). Las Vegas, NV, USA: IEEE. http://doi.org/10.1109/CCNC.2007.156

Wang, B., & Baras, J. S. (2012). Integrated Modeling and Simulation Framework for Wireless Sensor Networks. In *2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises* (pp. 268–273). IEEE. http://doi.org/10.1109/WETICE.2012.28

Wang, M., Cao, J., Li, J., & Dasi, S. (2008). Middleware for Wireless Sensor Networks: A Survey. *Journal of Computer Science and Technology*, *23*(3), 305–326. http://doi.org/10.1007/s11390-008-9135-x

Wang, Q., & Balasingham, I. (2010). Wireless Sensor Networks - An Introduction. In G. V Merrett & Y. K. Tan (Eds.), *Wireless Sensor Networks: Application-Centric Design*. InTech. http://doi.org/10.5772/13225

Weise, T., Zapf, M., Khan, M. U., & Geihs, K. (2009). Combining Genetic Programming and Model-Driven Development. *International Journal of Computational Intelligence and Applications*, *8*(1), 37–52. http://doi.org/10.1142/S1469026809002436

Werner-Allen, G., Lorincz, K., Ruiz, M., Marcillo, O., Johnson, J., Lees, J., & Welsh, M. (2006). Deploying a Wireless Sensor Network on an Active Volcano. *IEEE Internet Computing*, *10*(2), 18–25. http://doi.org/10.1109/MIC.2006.26

Yaacoub, E., Kadri, A., Mushtaha, M., & Abu-Dayya, A. (2013). Air Quality Monitoring and Analysis in Qatar using a Wireless Sensor Network Deployment. In *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)* (pp. 596–601). Sardinia, Italy: IEEE. http://doi.org/10.1109/IWCMC.2013.6583625

Yu, S., & Zhou, S. (2010). A Survey on Metric of Software Complexity. In *2010 2nd IEEE International Conference on Information Management and Engineering* (Vol. 5, pp. 352–356). Chengdu, China: IEEE. http://doi.org/10.1109/ICIME.2010.5477581

Yuce, M. R. (2010). Implementation of wireless body area networks for healthcare systems. *Sensors and Actuators A: Physical*, *162*(1), 116–129. http://doi.org/10.1016/j.sna.2010.06.004

Zhang, Q., Li, J., Rong, J., Xu Weiheng, & He Jinping. (2011). Application of WSN in precision forestry. In *IEEE 2011 10th International Conference on Electronic Measurement & Instruments* (pp. 320–323). Chengdu, China: IEEE. http://doi.org/10.1109/ICEMI.2011.6038006

Zhao, F., & Liu, J. (2005). Towards semantic services for sensor-rich information systems. In *2nd International Conference on Broadband Networks, 2005.* (pp. 44–51). IEEE. http://doi.org/10.1109/ICBN.2005.1589709

Zheng, X., Perry, D. E., & Julien, C. (2014). BraceForce: A Middleware to Enable Sensing Integration in Mobile Applications for Novice Programmers. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems - MOBILESoft 2014* (pp. 8–17). Hyderabad, India: ACM Press. http://doi.org/10.1145/2593902.2593907

Zouinkhi, A., Mekki, K., & Abdelkrim, M. N. (2014). Application and Network Layers Design for Wireless Sensor Network to Supervise Chemical Active Product Warehouse. *International Journal of Computer Science, Engineering and Applications (IJCSEA)*, *4*(6), 53–72. http://doi.org/10.5121/ijcsea.2014.4605

APPENDIX A: Conferences and Training Courses

No.	Preliminary Study and Training Courses	Date
1	Attending MSc Module "Network programming and simulation"	2nd semester 2014/2015
2	Attending MSc Module "Software Architecture and Security"	2nd semester 2014/2015
3	Attending MSc Module "Wireless and Mobile Telecommunications"	2nd semester 2014/2015
4	College Dean's Annual Research Showcase Event- Submitting a poster and an abstract	18th June 2014
5	Attending symposium (PGNET2014, Liverpool)	23-24 /06/ 2014
6	College Dean's Annual Research Showcase Event- Submitting a poster and an abstract	28th June 2015
7	Attending The 11th International Conference for Internet Technology and Secured Transactions (ICITST-2016)	5-7 December 2016

APPENDIX B: Surveys

B.1 Sample of Research Articles Referred to WSN Programming Complexity

The below table includes a sample list of the academic articles that highlighted the complexity in developing and programming WSN applications.

Academic Article	Article Type
(Essaadi et al., 2017)	Book Section
(Ouadjaout et al., 2016)	Journal Article
(Chandra & Dwivedi, 2015)	Conference Paper
(Delamo et al., 2015)	Journal Article
(Tei et al., 2015)	Journal Article
(Afanasov et al., 2014)	Conference Paper
(Cecchinel, Mosser, & Collet, 2014)	Book Section
(Delicato et al., 2014)	Book Section
(Malavolta & Muccini, 2014)	Conference Paper
(Shimizu et al., 2014)	Conference Paper
(Afanasov, Mottola, & Ghezzi, 2013)	Conference Paper
(Dantas et al., 2013)	Conference Paper
(Elsts et al., 2013)	Conference Paper
(Julien & Wehrle, 2013)	Conference Paper
(Beal et al., 2012)	Journal Article
(Shimizu et al., 2012)	Conference Paper
(Ceriotti et al., 2011)	Conference Paper
(Geihs et al., 2011)	Conference Paper
(Mottola & Picco, 2011)	Journal Article
(Shimizu et al., 2011)	Conference Paper
(Dunkels et al., 2010)	Technical Report
(Picco, 2010)	Conference Paper
(Khedo & Subramanian, 2009)	Journal Article
(Sadilek, 2008)	Conference Paper
(Sugihara & Gupta, 2008)	Journal Article
(Losilla et al., 2007)	Book Section
(Sadilek, 2007)	Conference Paper
(Vicente-chicote, Losilla, Alvarez, Iborra, & Sanchez, 2007)	Journal Article
(Bakshi, Prasanna, Reich, & Larner, 2005)	Conference Paper
(Boulis, Han, & Srivastava, 2003)	Conference Paper

B.2 Sample of Research Articles Referred to WSN Novice Developers

The below table includes a sample list of the academic articles that highlighted the importance of novice developers and how we have to encourage them to develop their own applications.

Academic Article	Article Type
(Ravichandran et al., 2016)	Journal Article
(Tunc et al., 2016)	Conference Paper
(Chandra & Dwivedi, 2015)	Conference Paper
(Nguyen, Tran, Baraki, & Geihs, 2015)	Conference Paper
(Serna, Sreenan, & Fedor, 2015)	Conference Paper
(Singh, Vyas, & Varma, 2014)	Journal Article
(Strazdins & Selavo, 2014)	Journal Article
(Zheng, Perry, & Julien, 2014)	Conference Paper
(Elsts & Selavo, 2013)	Journal Article
(Valero et al., 2012)	Conference Paper
(Bai, Dick, Dinda, & Chou, 2011)	Conference Paper
(Hansen & Kusy, 2011)	Book Section
(Gordon et al., 2010)	Conference Paper
(Avilés-López & García-Macías, 2009)	Journal Article
(Bai, Dick, & Dinda, 2009)	Conference Paper
(Miller et al., 2009)	Conference Paper
(Khemapech, Miller, & Duncan, 2005)	Technical Report
(Vieira et al., 2005)	Conference Paper
(Zhao & Liu, 2005)	Conference Paper
(Levis & Culler, 2002)	Conference Paper

B.3 Sample of Research Documents Referred to Domain Experts

The below table includes a sample list of the academic articles that highlighted the importance of domain experts and how we have to encourage them to develop their own applications.

Academic Article	Article Type
(Essaadi et al., 2017)	Book Section
(Lazarescu, 2017)	Book Section
(Antonopoulos et al., 2016)	Journal Article
(Cecchinel et al., 2016)	Conference Paper
(De-Farias, Brito, et al., 2016)	Journal Article

(Kabac et al., 2016)	Conference Paper
(Lopes & Martins, 2016)	Journal Article
(Lazarescu & Lavagno, 2016)	Book Section
(Venčkauskas, Štuikys, Jusas, & Burbaitė, 2016)	Journal Article
(Elsts et al., 2015)	Conference Paper
(Chandra & Dwivedi, 2015)	Conference Paper
(Aoun, Alloush, Kermarrec, Champeau, & Zein, 2015)	Journal Article
(Rodrigues, Batista, et al., 2015)	Conference Paper
(Rodrigues, Delicato, et al., 2015)	Journal Article
(Serna et al., 2015)	Conference Paper
(Tei et al., 2015)	Journal Article
(Delicato et al., 2014)	Book Section
(Gaglione, Lo, & Yang, 2014)	Conference Paper
(Lange, Lösche, & Piotrowski, 2014)	Conference Paper
(Malavolta & Muccini, 2014)	Conference Paper
(Oppermann, Boano, et al., 2014)	Book Section
(Oppermann, Romer, et al., 2014)	Conference Paper
(Shimizu et al., 2014)	Conference Paper
(Elsts et al., 2013)	Conference Paper
(Bader & Oelmann, 2013)	Conference Paper
(Dantas et al., 2013)	Conference Paper
(Kyungtae Kang, Min-Young Nam, & Lui Sha, 2013)	Journal Article
(Mohorcic, Smolnikar, & Javornik, 2013)	Conference Paper
(Oshana & Kareling, 2013)	Book Section
(Piotrowski & Peter, 2013)	Conference Paper
(Rodrigues et al., 2013)	Conference Paper
(Elsts & Selavo, 2012)	Conference Paper
(Fajar, Nakanishi, Hisazumi, & Fukuda, 2012)	Conference Paper
(Flouri et al., 2012)	Journal Article
(Maissa, Kordon, Mouline, & Thierry-Mieg, 2012)	Conference Paper
(Matthys, Huygens, Hughes, Michiels, & Joosen, 2012)	Conference Paper
(Mottola & Picco, 2012)	Journal Article
(Peter & Langendorfer, 2012)	Conference Paper
(Shimizu et al., 2012)	Conference Paper
(Tranquillini et al., 2012)	Book Section
(Wang & Baras, 2012)	Conference Paper
(Bai, Dick, Chou, & Dinda, 2011)	Journal Article
(Bai, Dick, Dinda, et al., 2011)	Conference Paper
(Hansen & Kusy, 2011)	Conference Paper
(Mottola & Picco, 2011)	Journal Article
(Rodrigues et al., 2011)	Conference Paper
(Beckmann & Thoss, 2010)	Book Section
(Carboni, 2010)	Conference Paper
(Picco, 2010)	Conference Paper
(Bai et al., 2009)	Conference Paper

(Miller et al., 2009)	Conference Paper
(Naumowicz, Schröter, & Schiller, 2009)	Conference Paper
(Kushwaha, Amundson, Koutsoukos, Neema, & Sztipanovits, 2007)	Conference Paper
(Vicente-chicote et al., 2007)	Journal Article
(Walker, Moh, & Moh, 2007)	Conference Paper
(Luo et al., 2006)	Journal Article
(Romer & Mattern, 2004)	Journal Article
(Jie Liu, Chu, Liu, Reich, & Feng Zhao, 2003)	Journal Article
(García-hernández, Ibargüengoytia-gonzález, García-hernández, & Pérez-díaz, 2007)	Journal Article

APPENDIX C: Demonstration Case Study

****

```
/*
 * Copyright (c) 2002-2005 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */
/*
 * @author Phil Buonadonna
 * @author Gilman Tolle
 * @author David Gay
 * Revision:   $Id: BaseStationP.nc,v 1.12 2010-06-29 22:07:14 scipio Exp $
 */
/*
 * BaseStationP bridges packets between a serial channel and the radio.
 * Messages moving from serial to radio will be tagged with the group
 * ID compiled into the BaseStation, and messages moving from radio to
 * serial will be filtered by that same group id.
 */
#include "AM.h"
#include "Serial.h"
module BaseStationP @safe() {
  uses {
    interface Boot;
    interface SplitControl as SerialControl;
    interface SplitControl as RadioControl;
    interface AMSend as UartSend[am_id_t id];
    interface Receive as UartReceive[am_id_t id];
    interface Packet as UartPacket;
    interface AMPacket as UartAMPacket;
    interface AMSend as RadioSend[am_id_t id];
    interface Receive as RadioReceive[am_id_t id];
    interface Receive as RadioSnoop[am_id_t id];
    interface Packet as RadioPacket;
    interface AMPacket as RadioAMPacket;
    interface Leds;
  }
}
implementation
{
  enum {
    UART_QUEUE_LEN = 12,
    RADIO_QUEUE_LEN = 12,
  };
  message_t  uartQueueBufs[UART_QUEUE_LEN];
  message_t  * ONE_NOK uartQueue[UART_QUEUE_LEN];
  uint8_t    uartIn, uartOut;
  bool       uartBusy, uartFull;
  message_t  radioQueueBufs[RADIO_QUEUE_LEN];
  message_t  * ONE_NOK radioQueue[RADIO_QUEUE_LEN];
  uint8_t    radioIn, radioOut;
  bool       radioBusy, radioFull;

  task void uartSendTask();
  task void radioSendTask();
  void dropBlink() {
    call Leds.led2Toggle();
  }
  void failBlink() {
    call Leds.led2Toggle();
  }
  event void Boot.booted() {
    uint8_t i;
    for (i = 0; i < UART_QUEUE_LEN; i++) uartQueue[i] = &uartQueueBufs[i];
    uartIn = uartOut = 0;
    uartBusy = FALSE;
```

```
    uartFull = TRUE;
    for (i = 0; i < RADIO_QUEUE_LEN; i++) radioQueue[i] = &radioQueueBufs[i];
    radioIn = radioOut = 0;
    radioBusy = FALSE;
    radioFull = TRUE;
    if (call RadioControl.start() == EALREADY) radioFull = FALSE;
    if (call SerialControl.start() == EALREADY) uartFull = FALSE;
  }
  event void RadioControl.startDone(error_t error) {
    if (error == SUCCESS) {radioFull = FALSE;
    }
  }
  event void SerialControl.startDone(error_t error) {
    if (error == SUCCESS) {uartFull = FALSE;
    }
  }
  event void SerialControl.stopDone(error_t error) {}
  event void RadioControl.stopDone(error_t error) {}
  uint8_t count = 0;
  message_t* ONE receive(message_t* ONE msg, void* payload, uint8_t len);
  event message_t *RadioSnoop.receive[am_id_t id](message_t *msg,void *payload,uint8_t len)
{return receive(msg, payload, len);
  }
  event message_t *RadioReceive.receive[am_id_t id](message_t *msg,
                                                    void *payload,uint8_t len) {
    return receive(msg, payload, len);
  }
  message_t* receive(message_t *msg, void *payload, uint8_t len) {
    message_t *ret = msg;
    atomic {
      if (!uartFull)
        {
          ret = uartQueue[uartIn];
          uartQueue[uartIn] = msg;
          uartIn = (uartIn + 1) % UART_QUEUE_LEN;
          if (uartIn == uartOut)
            uartFull = TRUE;
          if (!uartBusy)
            {
              post uartSendTask();
              uartBusy = TRUE;
            }
        }
      else
        dropBlink();
    }
    return ret;
  }
  uint8_t tmpLen;
  task void uartSendTask() {
    uint8_t len;
    am_id_t id;
    am_addr_t addr, src;
    message_t* msg;
    am_group_t grp;
    atomic
      if (uartIn == uartOut && !uartFull)
        {
          uartBusy = FALSE;
          return;
        }
    msg = uartQueue[uartOut];
    tmpLen = len = call RadioPacket.payloadLength(msg);
    id = call RadioAMPacket.type(msg);
    addr = call RadioAMPacket.destination(msg);
    src = call RadioAMPacket.source(msg);
    grp = call RadioAMPacket.group(msg);
    call UartPacket.clear(msg);
    call UartAMPacket.setSource(msg, src);
    call UartAMPacket.setGroup(msg, grp);
    if (call UartSend.send[id](addr, uartQueue[uartOut], len) == SUCCESS)
      call Leds.led1Toggle();
    else
      {
        failBlink();
        post uartSendTask();
      }
```

```
  }
  event void UartSend.sendDone[am_id_t id](message_t* msg, error_t error) {
    if (error != SUCCESS)
      failBlink();
    else
      atomic
        if (msg == uartQueue[uartOut])
          {
            if (++uartOut >= UART_QUEUE_LEN)
              uartOut = 0;
            if (uartFull)
              uartFull = FALSE;
          }
    post uartSendTask();
  }
  event message_t *UartReceive.receive[am_id_t id](message_t *msg,
                                                   void *payload,uint8_t len) {
    message_t *ret = msg;
    bool reflectToken = FALSE;
    atomic
      if (!radioFull)
        {
          reflectToken = TRUE;
          ret = radioQueue[radioIn];
          radioQueue[radioIn] = msg;
          if (++radioIn >= RADIO_QUEUE_LEN)
            radioIn = 0;
          if (radioIn == radioOut)
            radioFull = TRUE;
          if (!radioBusy)
            {
              post radioSendTask();
              radioBusy = TRUE;
            }
        }
      else
        dropBlink();
    if (reflectToken) {
      //call UartTokenReceive.ReflectToken(Token);
    }
    return ret;
  }
  task void radioSendTask() {
    uint8_t len;
    am_id_t id;
    am_addr_t addr,source;
    message_t* msg;
    atomic
      if (radioIn == radioOut && !radioFull)
        {
          radioBusy = FALSE;
          return;
        }
    msg = radioQueue[radioOut];
    len = call UartPacket.payloadLength(msg);
    addr = call UartAMPacket.destination(msg);
    source = call UartAMPacket.source(msg);
    id = call UartAMPacket.type(msg);
    call RadioPacket.clear(msg);
    call RadioAMPacket.setSource(msg, source);
    if (call RadioSend.send[id](addr, msg, len) == SUCCESS)
      call Leds.led0Toggle();
    else {
        failBlink();
        post radioSendTask(); } }
  event void RadioSend.sendDone[am_id_t id](message_t* msg, error_t error) {
    if (error != SUCCESS)failBlink();
    else
      atomic
        if (msg == radioQueue[radioOut]) {
            if (++radioOut >= RADIO_QUEUE_LEN)radioOut = 0;
            if (radioFull)
              radioFull = FALSE;
          }
    post radioSendTask();
**********************************************************************
```

APPENDIX D: SenNet Grammar

```
**************************************************************************
SeNetApp returns SeNetApp:
        {SeNetApp}
        'SeNetApp'
        name=EString
        '{'
                ('id' id=EInt)?
                ('jobs' '{' jobs+=AbstractJob ( "," jobs+=AbstractJob)* '}' )?
        '}';
AbstractJob returns AbstractJob:
        NodeDataProcessing | SenseNowJob | SinkJob | SenseJob | NetworkLevelSpecialAlgorithm |
NetworkDataProcessing;
AbstractNode returns AbstractNode:
        SensorNode | ClusterHeadNode | SinkNode | ComputationNode;
AbstractNetwork returns AbstractNetwork:
        FlatNetwork | ClusteredNetwork;
AbstractStartEndingJobTrigger returns AbstractStartEndingJobTrigger:
        ReceiveMessageTrigger | ReceiveSerialMsgTrigger | StartJobTrigger_Impl | StopJobTrigger_Impl |
PushButtonTrigger;
AbstractAction returns AbstractAction:
        ReadNodeMemoryAction | WriteNodeMemoryAction | SendMessageAction | BlinkAction |
SendSerialPortMsgAction;
AbstractSensor returns AbstractSensor:
        AccelerometerSensor | LightSensor | LocationSensor | MicrophoneSensor | TemperatureSensor |
PressureSensor | HumiditySensor | VoltageSensor;
AbstractFlatNode returns AbstractFlatNode:
        SensorNode | ComputationNode;
AbstractClusterNode returns AbstractClusterNode:
        SensorNode | ClusterHeadNode;
EInt returns ecore::EInt:
        '-'? INT;
EString returns ecore::EString:
        STRING | ID;
NodeDataProcessing returns NodeDataProcessing:
        {NodeDataProcessing}
        'NodeDataProcessing'
        '{'
                ('id' id=EInt)?
                ('nodeDataProcessing' nodeDataProcessing=Aggregation)?
                ('sensingSamplingRate' sensingSamplingRate=ELong)?
                ('dataProcessingRate' dataProcessingRate=ELong)?
                ('JobTargetNode' JobTargetNode=AbstractNode)?
                ('JobTargetNetwork' JobTargetNetwork=AbstractNetwork)?
                ('start/EndTrigger' '{' start/EndTrigger+=AbstractStartEndingJobTrigger ( ","
start/EndTrigger+=AbstractStartEndingJobTrigger)* '}' )?
                ('jobaction' '{' jobaction+=AbstractAction ( "," jobaction+=AbstractAction)* '}' )?
        '}';

SenseNowJob returns SenseNowJob:
        {SenseNowJob}
        'SenseNowJob'
        '{'
                ('id' id=EInt)?
                ('JobTargetNode' JobTargetNode=AbstractNode)?
                ('JobTargetNetwork' JobTargetNetwork=AbstractNetwork)?
                ('start/EndTrigger' '{' start/EndTrigger+=AbstractStartEndingJobTrigger ( ","
start/EndTrigger+=AbstractStartEndingJobTrigger)* '}' )?
                ('jobaction' '{' jobaction+=AbstractAction ( "," jobaction+=AbstractAction)* '}' )?
        '}';
SinkJob returns SinkJob:
        {SinkJob}
        'SinkJob'
        '{'
                ('id' id=EInt)?
                ('JobTargetNode' JobTargetNode=AbstractNode)?
                ('JobTargetNetwork' JobTargetNetwork=AbstractNetwork)?
```

```
                ('start/EndTrigger' '{' start/EndTrigger+=AbstractStartEndingJobTrigger ( ","
start/EndTrigger+=AbstractStartEndingJobTrigger)* '}' )?
                ('jobaction' '{' jobaction+=AbstractAction ( "," jobaction+=AbstractAction)* '}' )?
        '}';
SenseJob returns SenseJob:
        {SenseJob}
        'SenseJob'
        '{'
                ('id' id=EInt)?
                ('sensingSamplingRate' sensingSamplingRate=ELong)?
                ('JobTargetNode' JobTargetNode=AbstractNode)?
                ('JobTargetNetwork' JobTargetNetwork=AbstractNetwork)?
                ('start/EndTrigger' '{' start/EndTrigger+=AbstractStartEndingJobTrigger ( ","
start/EndTrigger+=AbstractStartEndingJobTrigger)* '}' )?
                ('jobaction' '{' jobaction+=AbstractAction ( "," jobaction+=AbstractAction)* '}' )?
        '}';
NetworkLevelSpecialAlgorithm returns NetworkLevelSpecialAlgorithm:
        {NetworkLevelSpecialAlgorithm}
        'NetworkLevelSpecialAlgorithm'
        '{'
                ('id' id=EInt)?
                ('specialAlgorithmJob' specialAlgorithmJob=Fusion)?
                ('sensingSamplingRate' sensingSamplingRate=ELong)?
                ('algorithmSamplingRate' algorithmSamplingRate=ELong)?
                ('nodeIDList' '{' nodeIDList+=EInt ( "," nodeIDList+=EInt)* '}' )?
                ('JobTargetNode' JobTargetNode=AbstractNode)?
                ('JobTargetNetwork' JobTargetNetwork=AbstractNetwork)?
                ('start/EndTrigger' '{' start/EndTrigger+=AbstractStartEndingJobTrigger ( ","
start/EndTrigger+=AbstractStartEndingJobTrigger)* '}' )?
                ('jobaction' '{' jobaction+=AbstractAction ( "," jobaction+=AbstractAction)* '}' )?
        '}';
NetworkDataProcessing returns NetworkDataProcessing:
        {NetworkDataProcessing}
        'NetworkDataProcessing'
        '{'
                ('id' id=EInt)?
                ('networkDataProcessing' networkDataProcessing=Aggregation)?
                ('sensingSamplingRate' sensingSamplingRate=ELong)?
                ('dataProcessingRate' dataProcessingRate=ELong)?
                ('nodeIDList' '{' nodeIDList+=EInt ( "," nodeIDList+=EInt)* '}' )?
                ('JobTargetNode' JobTargetNode=AbstractNode)?
                ('JobTargetNetwork' JobTargetNetwork=AbstractNetwork)?
                ('start/EndTrigger' '{' start/EndTrigger+=AbstractStartEndingJobTrigger ( ","
start/EndTrigger+=AbstractStartEndingJobTrigger)* '}' )?
                ('jobaction' '{' jobaction+=AbstractAction ( "," jobaction+=AbstractAction)* '}' )?
        '}';
enum Position returns enums::Position:
                                mobile = 'mobile' | static = 'static';
SensorNode returns SensorNode:
        {SensorNode}
        'SensorNode'
        '{'
                ('id' id=EInt)?
                ('position' position=Position)?
                ('nodeResources' '{' nodeResources+=AbstractSensor ( ","
nodeResources+=AbstractSensor)* '}' )?
        '}';
ClusterHeadNode returns ClusterHeadNode:
        {ClusterHeadNode}
        'ClusterHeadNode'
        '{'
                ('id' id=EInt)?
                ('position' position=Position)?
                ('nodeResources' '{' nodeResources+=AbstractSensor ( ","
nodeResources+=AbstractSensor)* '}' )?
        '}';
SinkNode returns SinkNode:
        {SinkNode}
        'SinkNode'
        '{'
                ('id' id=EInt)?
                ('position' position=Position)?
                ('nodeResources' '{' nodeResources+=AbstractSensor ( ","
nodeResources+=AbstractSensor)* '}' )?
```

```
        '}';
ComputationNode returns ComputationNode:
        {ComputationNode}
        'ComputationNode'
        '{'
                ('id' id=EInt)?
                ('position' position=Position)?
                ('nodeResources' '{' nodeResources+=AbstractSensor ( ","
nodeResources+=AbstractSensor)* '}' )?
        '}';
AccelerometerSensor returns AccelerometerSensor:
        {AccelerometerSensor}
        'AccelerometerSensor'
        ;
LightSensor returns LightSensor:
        {LightSensor}
        'LightSensor'
        ;
LocationSensor returns LocationSensor:
        {LocationSensor}
        'LocationSensor'
        ;
MicrophoneSensor returns MicrophoneSensor:
        {MicrophoneSensor}
        'MicrophoneSensor'
        ;
TemperatureSensor returns TemperatureSensor:
        {TemperatureSensor}
        'TemperatureSensor'
        ;
PressureSensor returns PressureSensor:
        {PressureSensor}
        'PressureSensor'
        ;
HumiditySensor returns HumiditySensor:
        {HumiditySensor}
        'HumiditySensor'
        ;
VoltageSensor returns VoltageSensor:
        {VoltageSensor}
        'VoltageSensor'
        ;
enum InterNodeComm returns enums::InterNodeComm:
                                BT = 'BT' | Zigbee = 'Zigbee' | WLan = 'WLan' | RF = 'RF';
enum RoutingProtocol returns enums::RoutingProtocol:
                                leach = 'leach' | lqrp = 'lqrp' | spin = 'spin' | aodv = 'aodv' |
dodv = 'dodv' | dsr = 'dsr' | dsdv = 'dsdv' | gsr = 'gsr' | heed = 'heed' | gaf = 'gaf' | gear =
'gear' | tbf = 'tbf' | ead = 'ead' | pegasis = 'pegasis' | teen = 'teen' | speed = 'speed' | ctp =
'ctp' | dismentation = 'dismentation' | activeMessage = 'activeMessage';
FlatNetwork returns FlatNetwork:
        {FlatNetwork}
        'FlatNetwork'
        '{'
                ('id' id=EInt)?
                ('nbOfNodes' nbOfNodes=EInt)?
                ('interNodeComm' interNodeComm=InterNodeComm)?
                ('routingProtocol' routingProtocol=RoutingProtocol)?
                ('resources' '{' resources+=AbstractSensor ( "," resources+=AbstractSensor)* '}' )?
                ('sink' sink=SinkNode)?
                ('nodes' '{' nodes+=AbstractFlatNode ( "," nodes+=AbstractFlatNode)* '}' )?
        '}';
ClusteredNetwork returns ClusteredNetwork:
        {ClusteredNetwork}
        'ClusteredNetwork'
        '{'
                ('id' id=EInt)?
                ('nbOfNodes' nbOfNodes=EInt)?
                ('interNodeComm' interNodeComm=InterNodeComm)?
                ('routingProtocol' routingProtocol=RoutingProtocol)?
                ('resources' '{' resources+=AbstractSensor ( "," resources+=AbstractSensor)* '}' )?
                ('sink' sink=SinkNode)?
                ('clusters' '{' clusters+=Cluster ( "," clusters+=Cluster)* '}' )?
        '}';
Cluster returns Cluster:
```

```
        {Cluster}
        'Cluster'
        '{'
                ('id' id=EInt)?
                ('resources' '(' resources+=[AbstractSensor|EString] ( ","
resources+=[AbstractSensor|EString])* ')' )?
                ('nodes' '{' nodes+=AbstractClusterNode ( "," nodes+=AbstractClusterNode)* '}' )?
        '}';
ReceiveMessageTrigger returns ReceiveMessageTrigger:
        {ReceiveMessageTrigger}
        'ReceiveMessageTrigger'
        ;
ReceiveSerialMsgTrigger returns ReceiveSerialMsgTrigger:
        {ReceiveSerialMsgTrigger}
        'ReceiveSerialMsgTrigger'
        ;
StartJobTrigger_Impl returns StartJobTrigger:
        {StartJobTrigger}
        'StartJobTrigger'
        ;
StopJobTrigger_Impl returns StopJobTrigger:
        {StopJobTrigger}
        'StopJobTrigger'
        ;
PushButtonTrigger returns PushButtonTrigger:
        {PushButtonTrigger}
        'PushButtonTrigger'
        ;
ConditionalAction returns ConditionalAction:
        {ConditionalAction}
        'ConditionalAction'
        '{'
                ('sensorTerm' sensorTerm=EString)?
                ('logicalSymbol' logicalSymbol=LogicalSymbol)?
                ('value' value=ELong)?
                ('unit' unit=EString)?
        '}';
ReadNodeMemoryAction returns ReadNodeMemoryAction:
        {ReadNodeMemoryAction}
        'ReadNodeMemoryAction'
        '{'
                ('condition' condition=ConditionalAction)?
        '}';
WriteNodeMemoryAction returns WriteNodeMemoryAction:
        {WriteNodeMemoryAction}
        'WriteNodeMemoryAction'
        '{'
                ('condition' condition=ConditionalAction)?
        '}';
SendMessageAction returns SendMessageAction:
        {SendMessageAction}
        'SendMessageAction'
        '{'
                ('sendMessageTo' sendMessageTo=EInt)?
                ('condition' condition=ConditionalAction)?
        '}';
BlinkAction returns BlinkAction:
        {BlinkAction}
        'BlinkAction'
        '{'
                ('led' led=Leds)?
                ('status' status=LedStatus)?
                ('condition' condition=ConditionalAction)?
        '}';
SendSerialPortMsgAction returns SendSerialPortMsgAction:
        {SendSerialPortMsgAction}
        'SendSerialPortMsgAction'
        '{'
                ('condition' condition=ConditionalAction)?
        '}';
enum LogicalSymbol returns enums::LogicalSymbol:
                                equal = 'equal' | greaterThan = 'greaterThan' | greaterOrEqualThan =
'greaterOrEqualThan' | lessThan = 'lessThan' | lessOrEqualThan = 'lessOrEqualThan' | notEqual =
'notEqual';
```

```
ELong returns ecore::ELong:
        '-'? INT;
enum Leds returns enums::Leds:
                                led0 = 'led0' | led1 = 'led1' | led2 = 'led2';
enum LedStatus returns enums::LedStatus:
                                on = 'on' | off = 'off' | toggle = 'toggle';
enum Aggregation returns enums::Aggregation:
                                compression = 'compression' | max = 'max' | min = 'min' | avg = 'avg'
| count = 'count';
enum Fusion returns enums::Fusion:
                                classification = 'classification' | event_detection = 'event_detection'
| tracking = 'tracking' | decision_making = 'decision_making';
**************************************************************************************
```

APPENDIX E: SenNet CGC Implementation

E.1 AppCGenerator.xtend

****

```
class AppCGenerator extends AbstractSeNetGenerator {
  override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    resource.forEachNode[nodeId, nodeJob |
      val fileName = nodeName + "AppC.nc";
      fsa.generateFile(fileName, generateAppCSenNetpp(nodeJob).toString.trim)
    ]
  }
  def generateAppCSenNetpp(AbstractJob job) '''
    configuration «nodeName»AppC
    {

    }

    implementation {
      components «nodeName»C;
      components MainC;
      «nodeName»C.Boot -> MainC;

      «generateJob(job)»
    }
  '''

  def dispatch generateJob(SenseJob job) '''
    components new TimerMilliC();
    «nodeName»C.Timer -> TimerMilliC;

    «FOR action : job.jobaction»
      «generateAction(action)»
    «ENDFOR»

    «IF job.jobTargetNode != null»
      «FOR sensor : job.jobTargetNode.nodeResources»
        «generateSensor(sensor)»
      «ENDFOR»
    «ENDIF»
    «IF job.jobTargetNetwork != null»
      «FOR sensor : job.jobTargetNetwork.resources»
        «generateSensor(sensor)»
      «ENDFOR»
    «ENDIF»
  '''

  def dispatch generateJob(AbstractJob job) '''
    // Code generation not implemented for job: «job.eClass.name»
  '''

  def dispatch generateAction(SendMessageAction action) '''
    components ActiveMessageC;
    components new AMSenderC(AM_RADIO);
    components new AMReceiverC(AM_RADIO);
    «nodeName»C.Packet -> AMSenderC;
    «nodeName»C.AMPacket -> AMSenderC;
    «nodeName»C.AMSend -> AMSenderC;
    «nodeName»C.SplitControl -> ActiveMessageC;
    «nodeName»C.Receive -> AMReceiverC;
  '''

  def dispatch generateAction(BlinkAction job) '''
    components LedsC;
    «nodeName»C.Leds -> LedsC;
  '''
```

```
  def dispatch generateAction(AbstractAction action) '''
    // Code generation not implemented for action: «action.eClass.name»
  '''

  def dispatch generateSensor(TemperatureSensor sensor) '''
    components new SensirionSht11C() as Sensor;
    «nodeName»C.Read -> Sensor.Temperature;
  '''

  def dispatch generateSensor(AbstractSensor sensor) '''
    // Code generation not implemented for sensor: «sensor.eClass.name»
  '''
}
```

E.2 CGenerator.xtend

```
class CGenerator extends AbstractSeNetGenerator {

  override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    resource.forEachNode[nodeId, nodeJob |
      val fileName = nodeName + "C.nc";
      fsa.generateFile(fileName, generateAppCSenNetpp(nodeJob).toString.trim)
    ]
  }

  def generateAppCSenNetpp(AbstractJob job) '''
    «generateJobInclude(job)»

    module «nodeName»C
    {
      uses {
        interface Boot;
        interface SplitControl;
        «generateJobUsage(job)»
      }
    }
    Implementation {
      «generateJobImplementation(job)»
      event void Boot.booted()
      {
        «generateJobBoot(job)»
      }
      «generateJob(job)»
      «FOR action : job.jobaction»
        «generateActionEvent(action)»
      «ENDFOR»
    }
  '''
  def dispatch generateJob(SenseJob job) '''
    event void Timer.fired()
    {
    }
    event void Read.readDone(error_t result, uint16_t data)
    {
      «FOR action: job.jobaction»
        «generateActionWithCondition(action)»
      «ENDFOR»
    }
    event void SplitControl.startDone(error_t error)
    {
      if (error != SUCCESS)
      {
        call SplitControl.start();
      }
    }
    event void SplitControl.stopDone(error_t error)
```

```
  {
  }
'''

def dispatch generateJob(AbstractJob job) '''
  // Code generation not implemented for job: «job.eClass.name»
'''

def dispatch generateJobUsage(SenseJob job) '''
  interface Read<uint16_t>;
  interface Timer<TMilli>;
  «FOR action : job.jobaction»
    «generateActionUsage(action)»
  «ENDFOR»
'''

def dispatch generateJobUsage(AbstractJob job) { "" }

def dispatch generateJobImplementation(SenseJob job) '''
  «FOR action : job.jobaction»
    «generateActionImplementation(action)»
  «ENDFOR»
'''

def dispatch generateJobImplementation(AbstractJob job) '''
  // Code generation not implemented for job: «job.eClass.name»
'''

def dispatch generateJobBoot(SenseJob job) '''
  call Timer.startPeriodic(«job.sensingSamplingRate»);
  call SplitControl.start();
'''

def dispatch generateJobBoot(AbstractJob job) '''
  // Code generation not implemented for job: «job.eClass.name»
'''

def dispatch generateJobInclude(SenseJob job) '''
  #include "Timer.h"
  «FOR action : job.jobaction»
    «generateActionInclude(action)»
  «ENDFOR»
'''

def dispatch generateJobInclude(AbstractJob job) { "" }

def dispatch generateActionEvent(SendMessageAction job) '''
  event void AMSend.sendDone(message_t *msg, error_t error)
  {
    if (msg == & messagePacket)
    {
      radioBusy = FALSE;
    }
  }

  event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len)
  {
    return msg;
  }
'''

def dispatch generateActionEvent(AbstractAction job) '''
'''

def generateActionWithCondition(AbstractAction action) '''
  «IF action.condition != null»
    if («generateCondition(action.condition).toString.trim»)
    {
      «generateAction(action)»
    }
  «ELSE»
    «generateAction(action)»
  «ENDIF»
'''
```

```
  def generateCondition(ConditionalAction condition) '''
    «condition.sensorTerm» «condition.logicalSymbol.toOperator» «condition.value»
  '''

  def toOperator(LogicalSymbol symbol) {
    switch(symbol) {
      case EQUAL: "=="
      case GREATER_OR_EQUAL_THAN: ">="
      case GREATER_THAN: ">"
      case LESS_OR_EQUAL_THAN: "<="
      case LESS_THAN: "<"
      case NOT_EQUAL: "!="
    }
  }

  def dispatch generateAction(SendMessageAction action) '''
    if (radioBusy == FALSE)
    {
      ActiveMessage_t* msg = call Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
      msg -> NodeID = TOS_NODE_ID;
      msg -> TData = data;
      if ( call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t)))
      {
        radioBusy = TRUE;
      }
    }
  '''

  def dispatch generateAction(BlinkAction action) '''
    call Leds.«action.led.toString.toFirstLower»«action.status.toString.toFirstUpper»();
  '''

  def dispatch generateAction(AbstractAction action) {
    throw new UnsupportedOperationException("Yet to be implemented")
  }

  def dispatch generateActionInclude(SendMessageAction action) '''
    #include "AMsg.h"
  '''

  def dispatch generateActionInclude(AbstractAction action) '''
  '''

  def dispatch generateActionImplementation(SendMessageAction action) '''
    bool radioBusy;
    message_t messagePacket;
  '''

  def dispatch generateActionImplementation(AbstractAction action) '''
  '''

  def dispatch generateActionUsage(SendMessageAction action) '''
    interface Packet;
    interface AMPacket;
    interface AMSend;
    interface Receive;
  '''

  def dispatch generateActionUsage(BlinkAction action) '''
    interface Leds;
  '''

  def dispatch generateActionUsage(AbstractAction action) { "" }
}
```

****

E.3 HeaderGenerator.xtend

```
**********************************************************************************
class HeaderGenerator extends AbstractSeNetGenerator {

  override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    fsa.generateFile("AMsg.h", '''
      #ifndef AMSG_H
      #define AMSG_H
      typedef nx_struct ActiveMessage
      {
      nx_uint16_t NodeID;
      nx_uint16_t Data;
      } ActiveMessage_t;
      enum
      {
      AM_RADIO = 6
      };
      #endif /* AMSG_H */
    ''')
  }
}
**********************************************************************************
```

E.4 MakefileGenerator.xtend

```
**********************************************************************************
package org.wsn.sennet.xtext.generator
import org.eclipse.emf.ecore.resource.Resource
import org.eclipse.xtext.generator.IFileSystemAccess
class MakefileGenerator extends AbstractSeNetGenerator {
  override void doGenerate(Resource resource, IFileSystemAccess fsa) {
    resource.forEachNode[nodeId, nodeJob |
      var makefileName = "Makefile";
      if (nodeId > 0) {
        makefileName += nodeId
      }
      fsa.generateFile(makefileName, '''
        COMPONENT=«nodeName»AppC
        Include $(MAKERULES)
      ''')
    ]
  }
}
**********************************************************************************
```

APPENDIX F: Application Source Code Analysis Scenarios Implementation

F.1 Single Node SenseForward Scenario

F.1.1 SenNet Version

```
**********************************************************************************
SeNetApp SenseForward {
        jobs {
                SenseJob {
                        sensingSamplingRate 60000
                        JobTargetNode SensorNode {
                                nodeResources {
                                        TemperatureSensor
                                }
                        }
                        jobaction  {
                                SendMessageAction {
                                        sendMessageTo 1
                                }
                        }
                }
        }
}
**********************************************************************************
```

F.1.2 nesC Version

******************************nesC Configuration File ***************************

```
configuration SenseForward0AppC
{
}
implementation {
  components SenseForward0C;
  components MainC;
  SenseForward0C.Boot -> MainC;
  components new TimerMilliC();
  SenseForward0C.Timer -> TimerMilliC;
  components ActiveMessageC;
  components new AMSenderC(AM_RADIO);
  components new AMReceiverC(AM_RADIO);
  SenseForward0C.Packet -> AMSenderC;
  SenseForward0C.AMPacket -> AMSenderC;
  SenseForward0C.AMSend -> AMSenderC;
  SenseForward0C.SplitControl -> ActiveMessageC;
  SenseForward0C.Receive -> AMReceiverC;
  components new SensirionSht11C() as Sensor;
  SenseForward0C.Read -> Sensor.Temperature;
}
**********************************************************************************
```

******************************nesC Module File******************************

```
#include "Timer.h"
```

```
#include "AMsg.h"
module SenseForward0C
{
  uses {
    interface Boot;
    interface SplitControl;
    interface Read<uint16_t>;
    interface Timer<TMilli>;
    interface Packet;
    interface AMPacket;
    interface AMSend;
    interface Receive;
  }
}
Implementation {
  bool radioBusy;
  message_t messagePacket;
  event void Boot.booted()
  {
    call Timer.startPeriodic(60000);
    call SplitControl.start();
  }
  event void Timer.fired()
  {
  }
  event void Read.readDone(error_t result, uint16_t data)  {
    if (radioBusy == FALSE) {
      ActiveMessage_t* msg = call Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
      msg -> NodeID = TOS_NODE_ID;
      msg -> TData = data;
      if ( call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t))) {
        radioBusy = TRUE;
      }
    }
  }
  event void SplitControl.startDone(error_t error){
    if (error != SUCCESS){
      call SplitControl.start();
    }
  }
  event void SplitControl.stopDone(error_t error)
  {
  }
  event void AMSend.sendDone(message_t *msg, error_t error)
  {
    if (msg == & messagePacket){
      radioBusy = FALSE;
    }
  }
  event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len){
    return msg;
  }
}
```

****

***************************nesC Make File**************************************

```
COMPONENT=SenseForward0AppC
Include $(MAKERULES)
```

****

*****************************AMsg.h***************************************

```
        #ifndef AMSG_H
        #define AMSG_H
        typedef nx_struct ActiveMessage
        {
        nx_uint16_t NodeID;
        nx_uint16_t Data;
        } ActiveMessage_t;
        enum
        {
```

```
        AM_RADIO = 6
        };
#endif /* AMSG_H */
**********************************************************************************
```

F.1.3 HCM Operands and Operators Statistics

Single-Node SenseForward Operand and Operator Statistics							
SenNet				nesC			
Operands	Count	Operator	Count	Operands	Count	Operator	Count
1	1	{}	7	1	1	!=	1
600	1	jobaction	1	6	1	#	5
sensingSamplingRate	1	jobs	1	6000	1	$	1
		JobTargetNode	1	ActiveMessage	1	&	3
		nodeResources	1	ActiveMessage_t	4	()	20
		SendMessageAction	1	ActiveMessageC	2	*	5
		sendMessageTo	1	AM_RADIO	3	,	7
		SeNetApp	1	AMPacket	2	.	21
		SenseForwardApp	1	AMReceiverC	2	;	38
		SenseJob	1	AMSend	4	{}	18
		SensorNode	1	AMSenderC	4	<>	2
		TemperatureSensor	1	Amsg.h	1	=	7
				AMSG_H	2	==	2
				Boot	3	->	10
				booted	1	as	1
				Data	3	bool	1
				error	4	call	5
				error_t	4	COMPONENT	1
				event	7	components	7
				fired	1	configuration	1
				getPayload	1	define	1
				message_t	4	endif	1
				messagePacket	4	enum	1
				msg	7	typedef	1
				NodeID	2	uint16_t	2
				Packet	3	uint8_t	1
				payload	1	uses	1
				radioBusy	4	void	7
				result	1	stopDone	1
				SensirionSht11C	1	nx_struct	1
				Sensor	2	nx_uint16_t	2
				SingleNodeSenseForwardApp0AppC	2	if	4
				SingleNodeSenseForwardApp0C	10	ifndef	1
				SplitControl	6	implementation	2
				startPeriodic	1	Include	3
				TData	1	interface	8
				Temperature	1	len	1
				Timer	4	MainC	2
				Timer.h	1	MAKERULES	1
				TimerMilliC	2	module	1
				TMilli	1	Read	3
				TOS_NODE_ID	1	readDone	1
						Receive	4
						return	1
						send	1
						sendDone	1
						start	2
						startDone	1
						sizeof	2
						new	4

F.2 Network-Level SenseForward Scenario

F.2.1 SenNet

```
**********************************************************************************
SeNetApp NetSenseForwardApp {
        jobs {
                SenseJob {
                        sensingSamplingRate 60000
                        JobTargetNetwork FlatNetwork {
                                nbOfNodes 3
                                resources {
                                        TemperatureSensor
                                }
                        }
                        jobaction {
                                SendMessageAction {
                                        sendMessageTo 1
                                }
                        }
                }
        }
}
**********************************************************************************
```

F.2.2 nesC

****************************nesC Configuration File****************************

```
configuration NetSenseForwardApp2AppC
{
}
implementation {
  components NetSenseForwardApp2C;
  components MainC;
  NetSenseForwardApp2C.Boot -> MainC;
  components new TimerMilliC();
  NetSenseForwardApp2C.Timer -> TimerMilliC;
  components ActiveMessageC;
  components new AMSenderC(AM_RADIO);
  components new AMReceiverC(AM_RADIO);
  NetSenseForwardApp2C.Packet -> AMSenderC;
  NetSenseForwardApp2C.AMPacket -> AMSenderC;
  NetSenseForwardApp2C.AMSend -> AMSenderC;
  NetSenseForwardApp2C.SplitControl -> ActiveMessageC;
  NetSenseForwardApp2C.Receive -> AMReceiverC;
  components new SensirionSht11C() as Sensor;
  NetSenseForwardApp2C.Read -> Sensor.Temperature;
}
**********************************************************************************
```

****************************nesC Module File******************************

```
#include "Timer.h"
#include "AMsg.h"
module NetSenseForwardApp2C
{
  uses {
    interface Boot;
    interface SplitControl;
    interface Read<uint16_t>;
    interface Timer<TMilli>;
    interface Packet;
    interface AMPacket;
```

```
    interface AMSend;
    interface Receive;
  }
}
Implementation {
  bool radioBusy;
  message_t messagePacket;
  event void Boot.booted()
  {
    call Timer.startPeriodic(60000);
    call SplitControl.start();
  }
  event void Timer.fired()
  {
  }
  event void Read.readDone(error_t result, uint16_t data)
  {
    if (radioBusy == FALSE)
    {
      ActiveMessage_t* msg = call Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
      msg -> NodeID = TOS_NODE_ID;
      msg -> TData = data;
      if ( call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t)))
      {
        radioBusy = TRUE;
      }
    }
  }
  event void SplitControl.startDone(error_t error)
  {
    if (error != SUCCESS)
    {
      call SplitControl.start();
    }
  }
  event void SplitControl.stopDone(error_t error)
  {
  }
  event void AMSend.sendDone(message_t *msg, error_t error)
  {
    if (msg == & messagePacket)
    {
      radioBusy = FALSE;
    }
  }
  event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len)
  {
    return msg;
  }
}
```

****

**********Makefile**********

```
COMPONENT=NetSenseForwardApp2AppC
Include $(MAKERULES)
```

****

**********AMsg.h**********

```
#ifndef AMSG_H
#define AMSG_H
typedef nx_struct ActiveMessage
{
nx_uint16_t NodeID;
nx_uint16_t Data;
} ActiveMessage_t;
enum
{
AM_RADIO = 6
};
#endif /* AMSG_H */
```

****

F.2.3 HCM Operands and Operators Statistics

Network-Level SenseForward Operand and Operator Statistics							
SenNet				nesC			
Operands	**Count**	**Operator**	**Count**	**Operands**	**Count**	**Operator**	**Count**
1	1	{}	7	1	3	!=	3
3	1	FlatNetwork	1	6000	3	#	6
6000	1	jobaction	1	ActiveMessage_t	9	$	3
sensingSamplingRate	1	jobs	1	ActiveMessageC	6	&	9
		JobTargetNetwork	1	AM_RADIO	6	()	69
		nbOfNodes	1	AMPacket	6	*	15
		NetSenseForwardApp	1	AMReceiverC	6	,	21
		resources	1	AMSend	12	.	63
		SendMessageAction	1	AMSenderC	12	;	102
		sendMessageTo	1	AMsg.h	3	{}	48
		SeNetApp	1	Boot	9	<>	6
		SenseJob	1	booted	3	=	18
		TemperatureSensor	1	data	6	==	6
				error	12	->	30
				error_t	12	as	3
				fired	3	bool	3
				getPayload	3	call	15
				message_t	12	COMPONENT	3
				messagePacket	12	components	21
				msg	21	configuration	3
				NetSenseForwardApp2AppC	6	event	21
				NetSenseForwardApp2C	30	if	12
				NodeID	3	implementation	6
				Packet	9	Include	9
				payload	3	interface	24
				radioBusy	12	len	3
				result	3	MainC	6
				return	3	MAKERULES	3
				SensirionSht11C	3	module	3
				Sensor	6	new	12
				SplitControl	18	Read	9
				startPeriodic	3	readDone	3
				SUCCESS	3	Receive	12
				TData	3	send	3
				Temperature	3	sendDone	3
				Timer	12	sizeof	6
				Timer.h	3	start	6
				TimerMilliC	6	startDone	3
				TMilli	3	stopDone	3
				TOS_NODE_ID	3	uint16_t	6
				FALSE	6	uint8_t	3
				TRUE	3	uses	3
						void	21

F.3 Network-Level Event-Based Scenario

F.3.1 SenNet

```
**************************************************************************
SeNetApp NetEventBased {
        jobs {
                SenseJob {
                        sensingSamplingRate 60000
                        JobTargetNetwork FlatNetwork {
                                nbOfNodes 5
                                resources  {
                                        TemperatureSensor
                                }
                        }
                        jobaction {
                                SendMessageAction {
                                        sendMessageTo 1
                                        condition ConditionalAction {
                                                sensorTerm Temp
                                                logicalSymbol greaterThan
                                                value 40
                                                unit C
                                        }
                                }
                        }
                }
        }
}
**************************************************************************
```

F.3.2 nesC

**************************Sample of nesC Configuration File*************************

```
configuration NetEventBased2AppC
{
}
implementation {
  components NetEventBased2C;
  components MainC;
  NetEventBased2C.Boot -> MainC;
  components new TimerMilliC();
  NetEventBased2C.Timer -> TimerMilliC;
  components ActiveMessageC;
  components new AMSenderC(AM_RADIO);
  components new AMReceiverC(AM_RADIO);
  NetEventBased2C.Packet -> AMSenderC;
  NetEventBased2C.AMPacket -> AMSenderC;
  NetEventBased2C.AMSend -> AMSenderC;
  NetEventBased2C.SplitControl -> ActiveMessageC;
  NetEventBased2C.Receive -> AMReceiverC;
  components new SensirionSht11C() as Sensor;
  NetEventBased2C.Read -> Sensor.Temperature;
}
**************************************************************************
```

************************ Sample of nesC Module File *****************************

```
#include "Timer.h"
#include "AMsg.h"
module NetEventBased2C
{
  uses {
    interface Boot;
```

```
    interface SplitControl;
    interface Read<uint16_t>;
    interface Timer<TMilli>;
    interface Packet;
    interface AMPacket;
    interface AMSend;
    interface Receive;
  }
}
Implementation {
  bool radioBusy;
  message_t messagePacket;
  event void Boot.booted() {
    call Timer.startPeriodic(60000);
    call SplitControl.start();
  }
  event void Timer.fired() {
  }
  event void Read.readDone(error_t result, uint16_t data)  {
    if (Temp > 40){
      if (radioBusy == FALSE) {
        ActiveMessage_t* msg = call Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
        msg -> NodeID = TOS_NODE_ID;
        msg -> TData = data;
        if ( call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t))) {
          radioBusy = TRUE;
        }
      }
    }
  }
  event void SplitControl.startDone(error_t error) {
    if (error != SUCCESS) {
      call SplitControl.start();
    }
  }
  event void SplitControl.stopDone(error_t error){
  }
  event void AMSend.sendDone(message_t *msg, error_t error)
  {
    if (msg == & messagePacket)
    {
      radioBusy = FALSE;
    }
  }
  event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len)
  {
    return msg;
  }
}
```

****

Makefile

```
COMPONENT=NetEventBased2AppC
Include $(MAKERULES)
```

****

AMsg.h

```
#ifndef AMSG_H
#define AMSG_H
typedef nx_struct ActiveMessage
{
nx_uint16_t NodeID;
nx_uint16_t Data;
} ActiveMessage_t;
enum
{
AM_RADIO = 6
};
#endif /* AMSG_H */
```

****

F.3.3 HCM Operands and Operators Statistics

Network-Level SenseForward Operand and Operator Statistics							
SenNet				nesC			
Operands	**Count**	**Operator**	**Count**	**Operands**	**Count**	**Operator**	**Count**
1	1	{}	8	1	5	!=	5
5	1	condition	1	40	5	#	10
40	1	jobaction	1	3000	5	$	5
3000	1	jobs	1	ActiveMessage_t	15	&	15
C	1	JobTargetNetwork	1	ActiveMessageC	10	()	120
greaterThan	1	nbOfNodes	1	AM_RADIO	10	*	25
logicalSymbol	1	NetEventBased	1	AMPacket	10	,	35
sensingSamplingRate	1	resources	1	AMReceiverC	10	.	105
sensorTerm	1	SendMessageAction	1	AMSend	20	;	170
unit	1	sendMessageTo	1	AMSenderC	20	{}	85
value	1	SeNetApp	1	AMsg.h	5	<	10
Temp	1	SenseJob	1	Boot	15	=	30
		TemperatureSensor	1	booted	5	==	10
		ConditionalAction	1	data	10	>	15
		FlatNetwork	1	error	20	->	50
				error_t	20	as	5
				fired	5	bool	5
				getPayload	5	call	25
				interfaceTimer	5	COMPONENT	5
				message_t	20	components	35
				messagePacket	20	configuration	5
				NetEventBased2AppC	10	event	35
				NetEventBased2C	50	if	25
				NodeID	5	implementation	10
				Packet	15	Include	15
				payload	5	interface	35
				radioBusy	20	len	5
				result	5	MainC	10
				SensirionSht11C	5	MAKERULES	5
				Sensor	10	module	5
				SplitControl	30	new	20
				startPeriodic	5	Read	15
				TData	5	readDone	5
				Temp	5	Receive	20
				Temperature	5	return	5
				Timer	15	send	5
				Timer.h	5	sendDone	5
				TimerMilliC	10	sizeof	10
				TMilli	5	start	10
				TOS_NODE_ID	5	startDone	5
				FALSE	10	stopDone	5
				TRUE	5	SUCCESS	5
						uint16_t	10
						uint8_t	5
						uses	5
						void	35
						msg	35

APPENDIX G: Business Case Study Applicability Scenarios - nesC Implementation

G.1 Scenario-1

**********LMT10AppC.nc**********

```
configuration LMT10AppC{
}
implementation {
  components LMT10C;
  components MainC;
  LMT10C.Boot -> MainC;
  components new TimerMilliC() as T1;
  components new TimerMilliC() as T2;
  LMT10C.T1 -> T1;
  LMT10C.T2 -> T2;
  components ActiveMessageC;
  components new AMSenderC(AM_RADIO);
  components new AMReceiverC(AM_RADIO);
  LMT10C.Packet -> AMSenderC;
  LMT10C.AMPacket -> AMSenderC;
  LMT10C.AMSend -> AMSenderC;
  LMT10C.SplitControl -> ActiveMessageC;
  LMT10C.Receive -> AMReceiverC;
  components new Taos2550C() as Sensor;
  LMT10C.Read -> Sensor.VisibleLight;
}
```

**********LMT10C.nc**********

```
#include "Timer.h"
#include "AMsg.h"
module LMT10C
{
  uses {
    interface Boot;
    interface SplitControl;
    interface Read<uint8_t>;
    interface Timer<TMilli>;
    interface Packet;
    interface AMPacket;
    interface AMSend;
    interface Receive;
  }
}

implementation {
  bool radioBusy;
  message_t messagePacket;
  uint8_t avg counter results;
  event void Boot.booted()
  {
    call T1.startPeriodic(300000);
        call T2.startPeriodic(30000);
    call SplitControl.start();
  }
  event void T1.fired()
  {
  }
  event void T2.fired()
  {
        Call Read.read()
```

```
  }
  event void Read.readDone(error_t result, uint8_t data)
    {
        avg = avg + data;
        counter = counter + 1;
    }
        ActiveMessage_t* msg = call Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
        msg -> NodeID = TOS_NODE_ID;
        results = avg / counter;
        msg -> TData = results;
        if ( call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t)))
        {
          radioBusy = TRUE;
        }
 }
}
  event void SplitControl.startDone(error_t error)
  {
    if (error != SUCCESS)
    {
      call SplitControl.start();
    }
  }
  event void SplitControl.stopDone(error_t error)
  {
  }
  event void AMSend.sendDone(message_t *msg, error_t error)
  {
    if (msg == & messagePacket)
    {
      radioBusy = FALSE;
    }
  }
  event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len)
  {
    return msg;
  }
}
```

****

G.2 Scneario-2

**********TMSH2AppC.nc**********

```
configuration TMSH2AppC{
}
implementation {
  components TMSH2C;
  components MainC;
  TMSH2C.Boot -> MainC;
  components new TimerMilliC() as T1;
  components new TimerMilliC() as T2;
  TMSH2C.T1 -> T1;
  TMSH2C.T2 -> T2;
  components ActiveMessageC;
  components new AMSenderC(AM_RADIO);
  components new AMReceiverC(AM_RADIO);
  TMSH2C.Packet -> AMSenderC;
  TMSH2C.AMPacket -> AMSenderC;
  TMSH2C.AMSend -> AMSenderC;
  TMSH2C.SplitControl -> ActiveMessageC;
  TMSH2C.Receive -> AMReceiverC;
  components new SensirionSht11C() as Sensor;
  TMSH2C.Read -> Sensor.Temperature;
}
```

****

**********TMSH2C.nc**********

```
#include "Timer.h"
```

```
#include "AMsg.h"
module TMSH2C
{
  uses {
    interface Boot;
    interface SplitControl;
    interface Read<uint16_t>;
    interface Timer<TMilli>;
    interface Packet;
    interface AMPacket;
    interface AMSend;
    interface Receive;
  }
}

implementation {
  bool radioBusy;
  message_t messagePacket;
  uint16_t max results;
  event void Boot.booted()  {
    call T1.startPeriodic(600000);
        call T2.startPeriodic(60000);
    call SplitControl.start();
  }
  event void T1.fired()  {
  }
  event void T2.fired()  {
  }
  event void Read.readDone(error_t result, uint16_t data)
  {
    event void T1.fired()
  {
  event void T2.fired() {
    event void Read.readDone(error_t result, uint16_t data)
    {
        If (max < data);
{
max = data;
        }
    }
  }
        If (max > 20)
{
        ActiveMessage_t* msg = call Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
        msg -> NodeID = TOS_NODE_ID;
            msg -> TData = max;
        if ( call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t))){
          radioBusy = TRUE;
        }
   }
 }
}
  event void SplitControl.startDone(error_t error) {
    if (error != SUCCESS)
    {
      call SplitControl.start();
    }
  }
  event void SplitControl.stopDone(error_t error)
  {
  }
  event void AMSend.sendDone(message_t *msg, error_t error){
    if (msg == & messagePacket)
    {
      radioBusy = FALSE;
    }
  }
  event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len)
  {
    return msg;
  }
}
```

****

APPENDIX H: User Study Experiment

H.1 Ethical Approval Letter

University of Salford MANCHESTER

Research, Innovation and Academic Engagement Ethical Approval Panel

Research Centres Support Team
G0.3 Joule House
University of Salford
M5 4WT

T +44(0)161 295 5278

www.salford.ac.uk/

15 September 2016

Aymen Salman

Dear Aymen

RE: ETHICS APPLICATION ST16/137 – Building a Domain-Specific Language for Wireless Sensor Networks to facilitate Application Development

Based on the information you provided, I am pleased to inform you that your application ST16/137 has been approved.

If there are any changes to the project and/ or its methodology, please inform the Panel as soon as possible by contacting S&T-ResearchEthics@salford.ac.uk

Yours sincerely,

Prof Mohammed Arif
Chair of the Science & Technology Research Ethics Panel
Professor of Sustainability and Process Management,
School of Built Environment
University of Salford
Maxwell Building, The Crescent
Greater Manchester, UK M5 4WT
Phone: + 44 161 295 6829
Email: m.arif@salford.ac.uk

H.2 Pre-Experiment Questions

Pre-Experiment Questions Template		
No.	**Question**	**Answer**
1	Study Background (Example: Computer Engineering)	
2	Current education level (Example: Ph.D. Student 1st year)	
3	Which technology are you involved/interested in? (Example: Cloud Computing, IoT)	
4	Do you have any programming Skills?	☐ Yes
		☐ NO
5	List the programming languages you have used previously	
6	How do you rate your programming proficiency?	☐ Advance Programming Skills
		☐ Intermediate
		☐ Basic
		☐ No Programming Skills
7	How many years have you been working on these programming languages?	
8	How do you rate your WSN background information?	☐ Excellent
		☐ Good
		☐ Average
		☐ Poor
		☐ No information
9	Have you programmed WSN previously?	☐ Yes
		☐ NO

H.3 Post-Experiment Questions

Type	Comment
IQ	Introductory Questions
1	SenNet Direct Evaluation
2	SenNet vs. nesC
3	Open Question

Goal	**Metrics**	**Type**	**No.**	**Question**	
IQ			1	Were the SenNet and nesC tutorials and presentation given easy to understand?	☐ Very Easy
					☐ Easy
					☐ Neutral
					☐ Difficult
					☐ Very Difficult
IQ			2	Were the descriptions of the tasks clear?	☐ Very Easy
					☐ Easy
					☐ Neutral
					☐ Difficult
					☐ Very Difficult
Usability	M11 effectiveness	1	3	Do you consider that a developer could successfully develop WSN applications using SenNet?	☐ Agree Strongly
					☐ Agree
					☐ Neutral
					☐ Disagree
					☐ Disagree Strongly
					☐ Agree Strongly

	M121 Efficiency	1	4	Would you agree that SenNet is more efficient to use than nesC, in terms of efforts and time required to successfully develop WSN application?	☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly
		1	5	According to the tasks you have asked to program, would you agree that using SenNet will decrease the development time required to develop WSN application more than nesC?	☐ Agree Strongly ☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly
		2	6	Which programming language do you think would require less background in WSN technology?	☐ SenNet ☐ nesC ☐ Undecided
		2	7	Which programming language do you think would require less programming skills?	☐ SenNet ☐ nesC ☐ Undecided
		1	8	Would you agree that SenNet has the capability to help users achieve their tasks in an acceptable number of program development activities?	☐ Agree Strongly ☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly
	M13 Likeability, user perception	2	9	If you needed to develop a WSN application, which language toolkit would you prefer to use?	☐ SenNet ☐ nesC ☐ Undecided
		2	10	Which programming language environment you consider would be the most friendly tool?	☐ SenNet ☐ nesC ☐ Undecided
		2	11	Which programming language you would be more likely to recommend for programming WSN applications?	☐ SenNet ☐ nesC ☐ Undecided
		2	12	Which programming language you consider would have more attractive symbols?	☐ SenNet ☐ nesC ☐ Undecided
	M14 Learnability	2	13	Which programming language you consider would be easier to learn and use?	☐ SenNet ☐ nesC ☐ Undecided
		2	14	Which programming language do you consider its concepts and symbols are easier to learn and remember (e.g., ease of learning elements, ease of learning to develop a program)?	☐ SenNet ☐ nesC ☐ Undecided
	M15 Programming Technique	2	15	Is it easier to develop a WSN application using node-level or network-level programming?	☐ Specific-Node ☐ Network-Level ☐ Undecided
	M16 Maintaining existing applications	2	16	Which programming language do you consider easier to make a change or editing for existing application?	☐ SenNet ☐ nesC ☐ Undecided
	M17 Mind to program mapping	2	17	Which programming language do you consider would provide better support a problem-solving that can be mapped into a program easily?	☐ SenNet ☐ nesC ☐ Undecided
Functional Suitability	M21 Appropriateness	1	18	Do you consider that SenNet has satisfied all WSN application requirements?	☐ Agree Strongly ☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly

		1	19	All concepts and scenarios of the domain can be expressed in SenNet.	☐ Agree Strongly ☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly
		1	20	SenNet is appropriate for developing WSN applications (e.g. to express an algorithm).	☐ Agree Strongly ☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly
		1	21	SenNet constructs correspond to relevant domain concepts. The language does not include concepts that are not important for the domain.	☐ Agree Strongly ☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly
		1	22	SenNet does not contain conflicting elements.	☐ Agree Strongly ☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly
		1	23	SenNet is at the right abstraction level in that it is not too complex or detailed than necessary.	☐ Agree Strongly ☐ Agree ☐ Neutral ☐ Disagree ☐ Disagree Strongly
Open Question		3	24	What other operations and activities would you prefer to be added to the SenNet?	

H.4 Task-3

User Study Task-3

****************************SenNet Code Sample********************************

```
SeNetApp US41 {
        jobs {
                NodeDataProcessing {
                        id 100
                        nodeDataProcessing max
                        sensingSamplingRate 60000
                        dataProcessingRate 600000
                        JobTargetNode SensorNode {
                                nodeResources {
                                        TemperatureSensor
                                }
                        }
                        jobaction {
                                SendMessageAction {
                                        sendMessageTo 1
                                        condition ConditionalAction {
                                                sensorTerm Temp
                                                logicalSymbol greaterThan
                                                value 20
                                                unit C
                                        }
                                }
                        }
                }
        }
}
```

****

**************************nesCode SampleC.nc*****************************

```
#include "Timer.h"
#include "AMsg.h"
module nesCodeSampleC {
  uses {
    interface Boot;
    interface SplitControl;
    interface Read<uint8_t>;
    interface Timer<TMilli>;
    interface Packet;
    interface AMPacket;
    interface AMSend;
    interface Receive;
  }
}

implementation {
  bool radioBusy;
  message_t messagePacket;
  uint8_t avg counter results;
  event void Boot.booted() {
    call T1.startPeriodic(600000);
        call T2.startPeriodic(60000);
    call SplitControl.start();
  }
  event void T1.fired()
  {
  }
  event void T2.fired()
  {
        Call Read.read()
  }
  event void Read.readDone(error_t result, uint8_t data)
    {
        avg = avg + data;
        counter = counter + 1;
    }
        ActiveMessage_t* msg = call Packet.getPayload(&messagePacket,sizeof(ActiveMessage_t));
        msg -> NodeID = TOS_NODE_ID;
        results = avg / counter;
        msg -> TData = results;
        if ( call AMSend.send(1,&messagePacket,sizeof(ActiveMessage_t)))
        {
          radioBusy = TRUE;
        }
 }
}
  event void SplitControl.startDone(error_t error) {
    if (error != SUCCESS)
    {
      call SplitControl.start();
    }
  }
  event void SplitControl.stopDone(error_t error) {
  }
  event void AMSend.sendDone(message_t *msg, error_t error)
  {
    if (msg == & messagePacket)
    {
      radioBusy = FALSE;
    }
  }
  event message_t * Receive.receive(message_t *msg, void *payload, uint8_t len)
  {
    return msg;
  }
}
```

****