Integration of Wireless Multimedia Sensor Networks in a community sensor network

Author: Enzo Brands
Supervisor: José Maria Barcelo-Ordinas

A thesis submitted in fulfilment of the requirements for the degree of Master in Innovation and Research in Informatics

in the

Compnet research group
Facultat d’Informàtica de Barcelona
Universitat Politècnica de Catalunya

June 2014
Abstract

Facultat d’Informàtica de Barcelona
Compnet

Master in Innovation and Research in Informatics

Integration of Wireless Multimedia Sensor Networks in a community sensor network

by Enzo Brands

Wireless sensor networks have increased the amount of data that can be collected from users and the environment. These sensor networks are usually used as unimodal closed systems, and therefore the collected information is not always publicly available. Openly providing this data in a structured way makes it possible to share this information with other repositories or applications. The purpose of this master’s thesis is to integrate multimedia in the CommonSense platform, a community sensor platform which collects data coming from Wireless Sensor Networks (WSN) and makes the data publicly available in the Resource Description Framework (RDF) format. RDF provides metadata in a machine-readable form, so it can be interpreted by software, and easily shared with other repositories. To make the multimedia data available in the RDF format, a new ontology for the description of the metadata is developed. This ontology is based on the Semantic Sensor Network (SSN) and the Dublin Core (DC) ontologies used by the CommonSense platform to describe the sensor data, as well as a new ontology containing classes and properties to describe images in a WSN context.

To capture images, a Raspberry Pi equipped with a small USB camera is used. The data is sent over TCP/IP via a WiFi USB dongle. The lack of a sleep mode makes the Raspberry Pi not suitable to be used in a power constrained environment. To solve this problem a Sleepy Pi is used. The Sleepy Pi is an Arduino based device that can be used to wake or sleep the Raspberry Pi periodically, so the power consumption can be significantly reduced. The Sleepy Pi also provides analogue inputs, pulse-width modulation (PWM) and additional general-purpose input/output (GPIO), providing the possibility to wake up the Raspberry Pi based on sensor readings...
Acknowledgements

I would like to thank my supervisor professor José Maria Barcelo-Ordinas for providing me the opportunity to write my master’s thesis at Universitat Politècnica de Catalunya, and providing excellent support and feedback. I would also like to thank professor Jorge Garcia-Vidal who helped me very well in this research. A special thanks goes out to professor Josep Maria Torrents for helping me set up and execute experiments to measure power consumption. I would also like to thank Joan Salvatella and Oscar Trullols for the excellent feedback and explanation of the CommonSense platform.

Furthermore, I would like to thank my fellow students Farnoosh Farokhmanesh, Ismael Peral and Ugaitz Amozarrain for their critical views and feedback on my project. Lastly, I would like to thank professor Filip Van der Schueren and professor Peter Hellinckx of the University of Antwerp for their support and providing me with the opportunity to participate in the Erasmus exchange program...
Contents

Abstract i
Acknowledgements ii

Contents iii
List of Figures vi
List of Tables vii

1 Introduction 1
  1.1 Wireless multimedia sensor networks and open link data . . . . . . . . . . 1
    1.1.1 Main goal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

2 State of the art 3
  2.1 Web of data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  2.2 Multimedia in WSNs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  2.3 Wireless Multimedia Sensor Nodes . . . . . . . . . . . . . . . . . . . . . . 4
    2.3.1 eCAM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
    2.3.2 CITRIC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  2.4 CommonSense . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    2.4.1 Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
    2.4.2 Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
    2.4.3 Ontology based model . . . . . . . . . . . . . . . . . . . . . . . . . 7
    2.4.4 Database . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
    2.4.5 API . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
    2.4.6 Website . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3 Methodology 11
  3.1 Integration of multimedia in the CommonSense platform . . . . . . . . . . 11
    3.1.1 Ontology for images . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
      3.1.1.1 Web Ontology Language . . . . . . . . . . . . . . . . . . . . . 11
      3.1.1.2 Semantic Sensor Network . . . . . . . . . . . . . . . . . . . . . 12
      3.1.1.3 Dublin Core . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
      3.1.1.4 Protege . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    3.1.2 Reading and sending multimedia data . . . . . . . . . . . . . . . . . 12
      3.1.2.1 Open link data . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
      3.1.2.2 Resource Description Framework . . . . . . . . . . . . . . . . 13
3.2 Wireless Multimedia Sensor node ........................................ 13
   3.2.1 Raspberry Pi ......................................................... 14
   3.2.2 iLive ................................................................. 14
   3.2.3 MWiFi ................................................................. 14
   3.2.4 MiLive ................................................................. 14
   3.2.5 Sleepy Pi ................................................................. 14
   3.2.6 WiFi vs low power WiFi ............................................ 15
   3.2.7 HTTP vs CoAP ....................................................... 15
   3.2.8 Calculating battery lifetime ....................................... 16

4 Development of the proposal / technical / work ......................... 17
   4.1 Integration of Multimedia ............................................. 17
      4.1.1 Adapted model ..................................................... 17
      4.1.2 Semantic Sensor Images Ontology .............................. 17
         4.1.2.1 Class sensorImages ........................................ 18
         4.1.2.2 Classes compression and imageProcessing ............... 19
      4.1.3 Ontology based model .......................................... 20
      4.1.4 Requests .......................................................... 21
         4.1.4.1 Posting data to the server ................................ 21
         4.1.4.2 Requesting the data from the server .................... 22
      4.1.5 Database ........................................................ 24
         4.1.5.1 Tables ......................................................... 24
         4.1.5.2 Queries ....................................................... 26
   4.2 Multimedia sensor node .............................................. 26
      4.2.1 Raspberry Pi and Sleepy Pi ................................... 27
         4.2.1.1 Waking and shutting down the Raspberry Pi by RTC ... 27
         4.2.1.2 Waking and shutting down the Raspberry Pi by sound sensor .... 28
      4.2.2 Power consumption ............................................. 29
         4.2.2.1 CoAP vs. HTTP ............................................. 29
         4.2.2.2 Measurements .............................................. 30
         4.2.2.3 Battery life ............................................... 31
         4.2.2.4 WiFi vs low-power WiFi .................................... 33

5 Evaluation of the proposal / technical / work .......................... 35
   5.1 Objectives .................................................................. 35
      5.1.1 Adaption of CommonSense for multimedia traffic .......... 35
      5.1.2 Creation of a Wireless Multimedia Sensor Node ............ 36
   5.2 Future work ................................................................ 36
      5.2.1 CommonSense ....................................................... 36
      5.2.2 Wireless Multimedia Sensor Node ............................. 37

6 Conclusion ....................................................................... 38

A Acronyms ....................................................................... 39
List of Figures

2.1 CommonSense platform ........................................... 5
2.2 Entities conceptual map ......................................... 6
2.3 CommonSense ontology based model ............................. 7
2.4 MySQL database ERD ............................................. 8
2.5 API work-flow ................................................... 9

3.1 HTTP vs. CoAP protocol stack ................................. 15

4.1 Entities conceptual map ......................................... 18
4.2 sensorImages class ............................................... 19
4.3 compression and imageProcessing classes ..................... 20
4.4 Ontology based model ........................................... 20
4.5 Flowchart of creation process of the RDF/XML description of the image . . 23
4.6 RDF description of an image .................................. 24
4.7 RDF description of images from a stream ...................... 24
4.8 ERD of added tables ............................................. 25
4.9 Flow of Raspberry Pi ........................................... 27
4.10 Flow of Sleepy Pi .............................................. 28
4.11 Sound sensor schematic ....................................... 29
4.12 Current in idle state ........................................... 30
4.13 Current with camera idle and WiFi connected ............... 31
4.14 Current with camera running and WiFi connected .......... 31
4.15 Current when sending and taking a picture ................... 32
4.16 Current when sending and taking a picture ................... 33
4.17 Current using regular WiFi ................................... 34
4.18 Current using low-power WiFi ................................ 34

B.1 Sound sensor schematic ....................................... 41
List of Tables

2.1 Ontology-based and concept model mapping ...................................... 7
2.2 Available RDF descriptions ............................................................. 9

4.1 Ontology based and conceptual model mapping ................................. 20
4.2 Active loads measurement 1 ............................................................. 32
4.3 Active loads measurement 2 ............................................................. 32
4.4 Battery lifetimes .............................................................................. 33
Chapter 1

Introduction

1.1 Wireless multimedia sensor networks and open link data

The use of wireless sensor networks (WSN) has made it possible to collect a large amount of data about their environment. These sensor networks are usually used as unimodal closed systems. Yet the information provided by these systems could be useful for many other applications. To make data collected by WSNs openly available for applications to use, this data must be provided in a structured way. The purpose of this master’s thesis is to research how multimedia data collected by sensors can be integrated in CommonSense, a community sensor network which aims to collect sensor and user data and make it publicly available. All data in the CommonSense platform is available in RDF format. This research focusses on collecting image data and make it available in RDF format. A new Raspberry Pi based wireless sensor node is developed to provide an easy to use, low power node for capturing and sending image data.

1.1.1 Main goal

The main goal of this master’s thesis is to research the possibilities to integrate multimedia data coming from a Wireless Sensor Network in the Semantic Web. This research fits within the development of CommonSense, a community sensor platform described in section 2.4, and has the following goals.

- Integration of multimedia information coming from a wireless sensor network in the CommonSense platform
• Making this information available as Open Link Data

• Design of a low power Raspberry Pi based wireless multimedia sensor node
Chapter 2

State of the art

2.1 Web of data

Since 2007, the Open Link Data cloud has grown significantly, and currently consists out of billions of RDF statements from various sources [1]. The web of data is an additional layer of the traditional web of documents with similar properties [2]:

- Any type of data is possible
- Anyone can publish data
- Entities are connected via RDF links, which makes it possible to discover new data sources at run-time
- Data is self-describing. If an unfamiliar vocabulary is encountered, the URI can be dereferenced
- Wide choice of vocabularies to represent data

2.2 Multimedia in WSNs

Implementing multimedia traffic in WSNs can be a challenge considering energy consumption, fault tolerance and memory management are important factors. Due to power constraints sensor nodes need a low-complexity design and good compression efficiency. It is essential to find a good trade off between processing and communication power consumption. This section focusses on an important challenge of multimedia traffic in WSN, the communication between sensor and sink.
Generally, the transport layer is responsible for ensuring end-to-end reliability of data and congestion control in a network to support multimedia traffic. However, end-to-end reliability is not an important factor in WSN design. In most wireless sensor networks, sensor nodes are densely deployed, which results in a high correlation in the data from an observed event, reducing the need for reliability. In addition, the transport layer faces challenges such as bounded delay, jitter in data delivery, minimum bandwidth availability, multiple data priorities, and session maintenance.

The traditional transport protocols are mainly based on assumptions which do not apply to WSNs. These protocols are based on low interference and packet loss by congestion rather than a noisy medium. Traffic in a WSN is burstier, as the sensor nodes do not transmit data until an event is detected. All nodes detecting an event will send data simultaneously resulting in high bursts of traffic. These differences result in significant energy inefficiencies and throughput degradation [3].

Transmission of high-bandwidth data from WSN nodes may result in a high energy consumption or bandwidth congestion. In many applications only certain events are interesting to the observer. Due to transmission cost being an important factor in the energy consumption in WSNs, in-node processing can be used to detect events in images and avoid constant transmission. Designing a sensor node to use with image sensors requires a different approach than a generic sensor node. A power-efficient processing unit and sufficient memory is necessary to extract information from the data. The ARM architecture proves to be a good power efficient processing unit for use in WMSNs [4].

### 2.3 Wireless Multimedia Sensor Nodes

This section provides a summary of two already developed multimedia sensor nodes.

#### 2.3.1 eCAM

ECam is a compact high data-rate wireless sensor node equipped with a miniature camera. It uses a VGA quality digital video camera in combination with an Eco node [5]. The camera module already performs compression and is thereby optimized for low power consumption and bandwidth demand. The Eco node has a built in radio, accelerometer, temperature sensor, infrared sensor, antenna and battery and is only 1 cm³ in volume. After integration with the camera the eCAM is still considerably smaller than modules in many other sensor platforms. ECam is capable of a higher data rate than most platforms, having a theoretical peak bandwidth of 1 Mbps. The eCam uses
communication scheduling to increase the performance between the camera and the radio. The eCam sens its information to a base station which uses either Ethernet or WiFi to create a fast uplink to the host computer. The eCam consumes about 70 mA at 3.3V in transmission mode [6].

2.3.2 CITRIC

CITRIC is a hardware platform which integrates a camera, a frequency scalable CPU, 16 MB of FLASH memory and 64 MB of RAM on a single device. The device connects with a standard sensor network node to form a camera node. It is capable of in-network processing of images to reduce bandwidth demand and transmission power. The node has a wider variety of pattern recognition applications than most platforms, due to the higher computing power. Moreover, the node easily integrates with existing sensor networks because it communicates over the IEEE 802.15.4 protocol. The CITRIC nodes are connected to each other or external nodes, while some nodes communicate with gateway devices which are connected to the internet. The nodes first perform pre-processing before sending the captured images to the central server, where it is routed to various clients [7].

2.4 CommonSense

The CommonSense platform [8, 9] is an Open Link Data Community Sensing Platform. The platform aims to integrate data generated by users or sensor networks, and make this data publicly available for other applications or repositories. The main challenge of the project is an easy connection of the sensor to the platform, and simple access to the data for third party applications. The CommonSense platform will be integrated with other repositories, forming part of the Open Link Data. Figure 2.1 provides an overview of the platform.
2.4.1 Architecture

The CommonSense system architecture is divided into three subsystems: a MySQL database, a Node.js API, and a website built with the Django framework. The database stores all the relevant data coming from the WSNs, as well as all the data needed to run the website. The API provides RESTful URL queries to provide access to the data in RDF format. The API is written in JavaScript and runs with Node.js, which makes use of the Express web application framework. This API is also used by the sensor nodes to connect to the platform. The sensor nodes send their data in RDF/XML format, which is received by the API and stored in the database. The third subsystem is a website which provides an easy-to-use interface for users where they can look up information about the various projects, as well as create new projects or manage existing ones.

2.4.2 Model

The model used in the CommonSense platform is described in figure 2.2. A user can create or join a project, and can have a different role in this project. An administrator can approve users to join the project as well as define their role in it, while they are also allowed to send and read all sensor data. A collaborator is only allowed to send and read sensor data but has no rights to do user management. An Observer is only allowed to read the sensor data. A project consists of one or more streams of data which is considered to be a collection of data. A sensor can have more than one stream to identify the different kind of data it measures.

![Figure 2.2: Entities conceptual map](image)

Users have to register the sensor nodes if they want to be able to send data to the CommonSense platform. If the user registers the sensor node, it is defined as the owner.
of the node so it can modify it’s properties and be notified when a node is unresponsive. The node types are used to group nodes by their measuring capabilities. Datatypes are used to provide a description of the data and provide the measure of unit.

### 2.4.3 Ontology based model

To provide the data in RDF format various ontologies are used, as described in figure 2.3. Table 2.1 provides the mapping between the conceptual model and the ontology based model.

![Ontology based model](image.png)

**Figure 2.3: CommonSense ontology based model**

<table>
<thead>
<tr>
<th>Ontology based</th>
<th>Conceptual</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn:SensingDevice</td>
<td>Node</td>
</tr>
<tr>
<td>geo:SpatialThing</td>
<td>Node</td>
</tr>
<tr>
<td>foaf:Agent</td>
<td>User</td>
</tr>
<tr>
<td>foaf:Project</td>
<td>Project</td>
</tr>
<tr>
<td>DUL:UnitsOfMeasure</td>
<td>DataType</td>
</tr>
<tr>
<td>DUL:Amount</td>
<td>Data</td>
</tr>
<tr>
<td>ssn:SensorOutput</td>
<td>Stream, Node</td>
</tr>
<tr>
<td>ssn:Observation</td>
<td>Data, DataType</td>
</tr>
<tr>
<td>ssn:Property</td>
<td>DataType</td>
</tr>
</tbody>
</table>

**Table 2.1: Ontology-based and concept model mapping**

### 2.4.4 Database

The database is used to store all the necessary data to manage the user interface, as well as storing the data coming from the various WSNs.
Figure 2.4 provides an overview of the tables used in the CommonSense platform.
2.4.5 API

The API is developed with Node.js in conjunction with the Express web application framework. It has two purposes: to store data coming from the WSNs and provide this data to other applications requesting it. Storing data is achieved by sending a POST request, as requesting data is done with a GET request. A user has a personal API-key which is used to identify if the data is coming from a registered and valid device, or to protect user’s data from being publicly available. The API workflow is shown in figure 2.5.

![API workflow diagram](image)

**Figure 2.5**: API work-flow

Table 2.2 provides an overview of the RDF descriptions that can be obtained from the API.

<table>
<thead>
<tr>
<th>Data</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observations</td>
<td>A description of the observation made by a sensor</td>
</tr>
<tr>
<td>Sensors</td>
<td>A description of the sensor</td>
</tr>
<tr>
<td>Projects</td>
<td>A description of a project</td>
</tr>
<tr>
<td>Agents</td>
<td>A description of a user</td>
</tr>
<tr>
<td>Properties</td>
<td>A description of an observable property</td>
</tr>
<tr>
<td>Units of measure</td>
<td>A description of a unit of measure</td>
</tr>
</tbody>
</table>

**Table 2.2**: Available RDF descriptions
2.4.6 Website

The CommonSense website is created using Django, a Python-based web framework following the Model-View-Controller (MVC) architectural pattern. The MVC pattern is an abstract model separating the data from an application, the user interface and business logic (inputs, generating queries or data) into three different components: model, view and controller. The MVC pattern also defines the interaction between those components.

The model represents the data structures, and has functions to consult or update the database, and query other APIs for data. The view is the information the user views. The controller manages the interaction between the model and the view. All these models are loaded from the database. The website is currently available at \url{http://commonsense.pc.ac.upc.edu}. 

Chapter 3

Methodology

3.1 Integration of multimedia in the CommonSense platform

The CommonSense platform explained in section 2.4 has to be adapted to support multimedia traffic. This section focusses on the technologies used to integrate multimedia traffic in this platform.

3.1.1 Ontology for images

The CommonSense platform aims to provide all data from sensors as open link data. A new ontology for the description of images in a WMSN context has to be defined. If already existing namespaces do not provide sufficient properties for the description of these images, a new ontology has to be defined in the OWL format.

3.1.1.1 Web Ontology Language

The Web Ontology Language (OWL) is a W3C Semantic Web language designed to represent knowledge about things and their relations. OWL is a computational logic-based language, which means that it can be read and interpreted by computer programs. OWL documents can be published in the World Wide Web, and can reference or be referenced by other published OWL documents [10].
3.1.1.2 Semantic Sensor Network

The Semantic Sensor Network (SSN) ontology is designed by the W3C Semantic Sensor Networks Incubator Group (SSN-XG) to describe sensors, observations and related concepts. Moreover, it imports several other ontologies to describe domain concepts, time, locations, etc. This ontology will be used as much as possible for the description of properties which are applicable to images and the sensors [11].

3.1.1.3 Dublin Core

The DC ontology provides fifteen properties for use in resource description. As this ontology is widely used in the description of images, several properties can be used to describe the contents of an image making sure many applications know what they mean [12].

3.1.1.4 Protege

Protege is an open source ontology editor and framework to build intelligent systems developed by Stanford University. Protege is actively supported by a strong community providing documentation and plug-ins. This editor can be used to design a new vocabulary in the OWL format, as well as taking care of imports from other ontologies. If already existing ontologies prove to be insufficient to describe multimedia data, this application will be used to create a new ontology for multimedia in a WMSN context [13].

3.1.2 Reading and sending multimedia data

The CommonSense platform has to provide all multimedia data in RDF format. The RDF documents have to be made available through HTTP requests so any application can access them. As the CommonSense platform makes use of NodeJS [14] for handling the requests, this technology will also be used to provide access to the multimedia data. NodeJS scripts will also be used to read images and related RDF data from a multimedia sensor node.

The MySQL database has to be adapted to make sure the platform is able to store the multimedia data.
3.1.2.1 Open link data

The structure of data on the web is an important factor for the re-usability. Data provided in a standard, well defined structure, makes it easier for developers to create tools that rely on it. HyperText Markup Language (HTML) is suited for formatting textual documents, but has limited possibilities of structuring data on the web. The growing need of structuring data on the web has led to development of various technologies, such as microformats and web application programming interfaces (APIs). Although microformats provide some possibilities for structuring data, the difficulty of expressing relations between entities makes it not well suited for sharing data on the web. Web APIs have made it possible to get access to data in a structured way. However, most web APIs do not have a mechanism to provide links to related data, making it not suitable for creating a web of data.

Linking data on the web requires a standard mechanism to create connections between data. To provide such mechanism, the Resource Description Framework (RDF) has been developed [2].

3.1.2.2 Resource Description Framework

RDF is a standard model for data interchange on the Web [15], which makes use of Uniform Resource Identifiers (URIs) to describe and link data. The two most commonly used RDF serialization formats are RDF/XML [16] and RDFa [17]. RDF describes resources as a number of triples. Each triple consists out of a subject, predicate, and object. The subject of a triple is the URI that identifies the described data. The predicate is the URI used to describe the relationship between the subject and the object. The predicate URI comes from vocabularies, which are a collection of URIs used to describe the relationship between data. The object is again a URI which identifies data [18].

3.2 Wireless Multimedia Sensor node

This section focusses on the devices and technologies that can be used to create a WMSN node. The sensor node is based on the Raspberry Pi, which will depend on other hardware to perform power management, capture multimedia, and connecting to the internet wireless. The sensor node will make use of either HTTP or CoAP over WiFi or low-power WiFi to send data.
3.2.1 Raspberry Pi

A Raspberry Pi is a credit-card sized computer that can be plugged into a TV or computer monitor, and can be used with a regular keyboard and mouse. Several OS distributions are available for the Raspberry Pi, with Linux-based distributions as the most common. The Raspberry Pi is capable of performing general tasks just as a desktop computer. A Raspberry Pi B model is used in this project. It is equipped with GPIO pins, two USB 2.0 ports, HDMI and Composite RCA video outputs, 256/512 MB RAM memory (depending on the age), a 700 MHz ARM11 processor, SD-card reader for use as HDD, 3.5mm-jack audio output, and a 10/100 ethernet port. Power is supplied through a 5V micro-USB input or the GPIO [19].

3.2.2 iLive

The iLive is a scalar wireless sensor node dedicated to environment data collection and precision agriculture, developed by the LIMOS research group [20]. It supports a variety of environmental sensors and is equipped with an IEEE 802.15.4 transceiver. The iLive has two different cores: one NanoRisc and one 8-bit AVR RISC. The NanoRisc is an ultra-low power consumption 4-bit RISC and the AVR is a low power 8-bit microcontroller [21].

3.2.3 MWiFi

The MWiFi is a Raspberry Pi board which contains a three cores system on a chip (SoC) which runs on a standard Linux OS. The MWiFi supports different types of camera (USB and CSI) and is equipped with a WiFi module [21].

3.2.4 MiLive

The MiLive consists out of an iLive and an MWiFi fitted together, making it a three core device. The NanoRisc and Power Supply Unit (PSU) form the power management unit (PMU) of the Raspberry Pi. The PMU can shutdown parts of the MWiFi, so the power consumption can be drastically reduced [21].

3.2.5 Sleepy Pi

The Sleepy Pi is an Arduino based board designed to perform power management for a Raspberry Pi. Because the Raspberry Pi has no ability to go in sleep mode, it is not
suitable for use with a battery. The Sleepy Pi can be programmed to control the power of the Raspberry Pi, and communicate to perform operations. Moreover, it adds the following I/O not present on the Raspberry Pi (3.2.1) [22].

- Analogue Inputs
- PWM
- Additional GPIO

### 3.2.6 WiFi vs low power WiFi

In this research, the difference in power consumption between using a USB WiFi dongle and a low-power WiFi module is studied.

### 3.2.7 HTTP vs CoAP

The sensor node will either make use of the HTTP protocol over TCP, or the CoAP protocol over UDP.

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a stateless protocol which is used for communication between a webclient and a webserver [23].

Constrained Application Protocol (CoAP) provides Representational state transfer (REST) web service functionalities for resource constrained devices. CoAP consists of a subset of Hypertext Transfer Protocol (HTTP) functionalities, which have been redesigned to meet the requirements of constrained devices. The HTTP and CoAP protocol stacks are illustrated in figure 3.1.

![Figure 3.1: HTTP vs. CoAP protocol stack.](image)

The CoAP protocol stack uses User Datagram Protocol (UDP) to create lower overhead and enable multicast support. CoAP consists out of two layers. The transaction layer handles the message exchange between endpoints. Four types of messages can be exchanged [24]
• Confirmable: requires acknowledgement
• Non-confirmable: no acknowledgement needed
• Acknowledgement: confirmation of message
• Reset: confirmable message can not be processed

The request/response layer is responsible for the resource manipulation and transmission.

### 3.2.8 Calculating battery lifetime

The calculation of the estimated battery lifetime is performed with formula 3.1 [25].

\[
\text{Lifetime} = \frac{bc}{(cl_1 \times t_1) + (cl_2 \times t_2) + \ldots + (cl_n \times t_n)}
\]  

(3.1)

To calculate the battery life, the battery capacity (bc, in mAh) is divided by the circuit load (cl, in mA). Because the WMSN node has various loads depending on the task it is performing, all these different loads have to be taken into account. This can be realized by adding all the different loads (cl, in mA) in respect to their active time (t, in seconds).
Chapter 4

Development of the proposal / technical / work

4.1 Integration of Multimedia

The CommonSense platform is adapted so it can receive and store image data coming from a WMSN, and make this data publicly available in RDF format. To achieve this, new HTTP requests had to be developed to receive the image data, and make it available for other applications. To describe the image data a new ontology is created.

4.1.1 Adapted model

The conceptual model explained in section 2.4.2 is adapted to make use of images. A WMSN node uses the same properties as a regular node, hence no adjustments to the model are made for sensor nodes. A stream can now be a collection of images as well as a collection of data. Images have a datatype as well, which has the same properties of the datatype used for regular data. A link between images and data is also made, making sure there can be sensor data which is related to an image. Moreover, an image has two new entities describing the used compression and image processing on the image. The adapted model is shown in figure 4.1.

4.1.2 Semantic Sensor Images Ontology

The Semantic Sensor Images (SSI) ontology provides various classes and properties to describe an image and link it to related data and sensors. Moreover, it provides properties to store the actual image data or URL to the image. Creating the taxonomy from
already existing ontologies proved to be a problem due to the lack of WMSN data in an OLD context. Therefore, new classes and properties had to be developed to integrate the image metadata in the CommonSense platform. The SSN and DC ontology are used when there was no need to create a new class or property. This ontology is described in an OWL file (3.1.1.1) made publicly available on the CommonSense website [26].

4.1.2.1 Class sensorImages

The sensorImages class of the SSI ontology is used to describe images coming from a Wireless sensor network. The class contains three subclasses: image, relatedSensor and Observation, as can be seen in figure 4.2.

The image subclass is used to provide information about the used compression or image processing performed on the sensor node and providing the image data or URL. The subclass consists of three object properties and two data properties.

- hasImageFormat: description of the image file format (object property)
- hasImageProcessing: description of performed image processing on the sensor node (object property)
- hasCompression: description of performed compression on the sensor node (object property)
- imageURL: URL to the image (data property)
- imageDataBase64: image data in base64 format (data property)
The relatedSensor subclass is used to provide information about related data and sensors. The subclass contains four subclasses: childSensors, relatedSensors, childObservations, relatedObservations. These subclasses each have one object property to store data about child and related sensors. A child sensor is considered a sensor attached to the same node as the camera, and a related sensor can be any other sensor device working together with the node equipped with a camera.

The Observation class implements two properties of the SSN ontology. The property ObservationResultTime is used to mark the time of the image, and the property isProducedBy is used to link to a description of the sensor node.

Three other data properties are implemented from the DC ontology, although any other DC property can be used.

### 4.1.2.2 Classes compression and imageProcessing

The classes compression and imageProcessing are used to describe the used compression and image processing performed on the device. The compression data is mainly used to know the quality of the produced image, and the image processing data can be used to describe a pattern recognition performed on the image. Figure 4.3 provides an overview of the classes.
4.1.3 Ontology based model

The use of a new ontology to describe images made it necessary to create a new mapping between the conceptional model (see 4.1.1) and the ontology based model. Figure 4.4 shows the used classes and relations of the model. Table 4.1 provides the mapping between the conceptual model and the ontology based model.

<table>
<thead>
<tr>
<th>Ontology based</th>
<th>Conceptual</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssi:image</td>
<td>Images</td>
</tr>
<tr>
<td>ssi:hasCompression</td>
<td>Compression</td>
</tr>
<tr>
<td>ssi:hasImageProcessing</td>
<td>Image Processing</td>
</tr>
<tr>
<td>ssn:Property</td>
<td>DataType</td>
</tr>
<tr>
<td>ssi:imageDataBase64</td>
<td>Image</td>
</tr>
<tr>
<td>ssi:imageURL</td>
<td>Image</td>
</tr>
<tr>
<td>ssi:relatedSensor</td>
<td>Data</td>
</tr>
<tr>
<td>ssn:Observation</td>
<td>Data</td>
</tr>
<tr>
<td>ssn:SensingDevice</td>
<td>Node</td>
</tr>
</tbody>
</table>

Table 4.1: Ontology based and conceptual model mapping
4.1.4 Requests

4.1.4.1 Posting data to the server

The sensors capturing the images must provide the data in RDF format, implementing the SSI ontology (see subsection 4.1.2). Although embedding RDF data is possible in image files through the use of Exif, XMP, IPTC, or other data models, the possibilities for the description of images in a WMSN context are not sufficient. In order to insure that the RDF data and image do not have to be linked back again at the receiver, the image data has to be embedded in the RDF description at the sensor node. This approach also meant that only one HTTP request had to be made. However, storing the binary data in an RDF/XML document is not possible. Therefore, the binary data from the image is converted to base64 so it can be embedded in an XML document. The drawback to this approach is that the image data size increases with approximately 33% [27].

The RDF/XML document is sent to the server in a HTTP/POST request. The URL for the request is formed by providing the version, project and stream used in the CommonSense platform. Several headers need to be set in the request for the server to accept it. The Content-Type and Accept header must be set to application/rdf+xml. To make sure only allowed users can post data the request should also contain a User header to indicate which user is posting data, and an X-Apikey header containing the API key of this user. The following URL is used to request the image data:

/version/project/stream

When a request is received by the server, it will read the RDF data and look if it contains the SSI sensorImage class. If the class is found the server processes the data as an image. The image data is read out (imageDataBase64 property) and converted back to binary format for storage on the server, where it is publicly available on the CommonSense website. The database stores this URL so it can be requested with a HTTP GET request. Moreover, the following remaining data from the image is stored:

- Image format: the file format described in the CommonSense platform as a UnitOfMeasure (SSN)
- Child and related observations
- The time of the observation
- The sensor which produced the observation
• The title, description and creator

• Compression and Image processing (only when this is performed, the properties are not mandatory)

The database queries are explained in section 4.1.5.2.

If the SSI sensorImage class is not found the server processes the RDF data as a measurement coming from a sensor.

4.1.4.2 Requesting the data from the server

To provide the collected data to external applications some new requests are developed. One request returns an RDF/XML document which describes the image and links to it’s related data. A URL to the image is provided as well, making sure that external applications can download this image. Other requests are developed to provide a list of links to all the images coming from a particular stream as well as a description of possible compression and image processing.

The description of an image can be requested with the following URL:

/images/id

If this request is sent to the server, the id of the image is derived from the URL. The server looks up the image and performs a query to the database to get the necessary data to form the RDF document. The following data is queried from the database:

• The data of the image

• The related measurements of the image

• The child measurements of the image

• The used compression and image processing

If the queries are completed, the server creates a JavaScript object containing the data to be exported to an XML/RDF document. The XML parser uses the object property name as the XML tag, and the data inside the property as the value of the tag. Creating child elements in a root element of the XML document can be achieved by storing a new object in the property which contains the root element data. This can be done as many times as necessary to create the hierarchy of the XML elements. To create
an RDF resource inside the XML element, a new object must be created containing an @rdf:resource property. This way the XML parser knows it should create the resource instead of parsing the data inside the tag.

When the object to describe the image is formed, all the necessary object properties for the root and child elements are created. The rdf:about tag contains the URL to this document itself. The rdfs:label tag contains a description of the stream and node which made the observation. An image description can contain multiple related and child observations (ssi:relatedSensor). To create a list of these observations, an empty object is created inside the ssi:relatedSensor object property. This empty object will later be accessed to store the links to the observations. The reason for this approach is the need of a loop to create the list of observations, which cannot be executed in the creation of an object. If there are no related or child observations the relatedSensor property is deleted. The ssi:image element contains the used file type, which is an RDF resource linking to the description of the datatype and can either be JPEG or PNG. Moreover it contains the URL to the image and the used compression and image processing (if there is any). The last properties are the DC fields used to describe the image. An example of the RDF description can be seen in figure 4.6.

The description of all the images from a specific stream can be obtained with the following URL:

/imagesbystream/streamid

The returned RDF/XML document contains a list of all the RDF descriptions of images of that specific stream.

The object to describe the list of images from a specific stream is formed by creating an object containing several image elements which contain an RDF resource tag to the image descriptions. An example of the RDF description by stream can be seen in figure 4.7.
The following URLs are used to get the description of a compression or image processing technique:

```
/ compression/id

/imageprocessing/id
```

Both requests return an RDF/XML document with a description containing DC tags about the compression or image processing.

### 4.1.5 Database

#### 4.1.5.1 Tables

To support the storage of images, new tables are added to the MySQL database. Figure 4.8 provides an overview of the added and modified tables.
The Images table is used to store the image data coming from the sensor node. The title, creator, date and description fields are used to store the metadata of the image embedded in the RDF/XML document received from the sensor node. The value field is used to store the URL to the image. The one to many relationship between the tables Streams and Images is used to identify the stream where the image came from (the stream links to the sensor, see section 2.4.2). The one to many relationship between tables DataTypes and Images is used to identify the correct DataType (JPEG or PNG for images). The one to many relationship from the table Images to the tables UsedCompression and UsedImageProcessing are used to add respectively compression or image processing to an image. It uses the foreign keys idImage to link to an image and idCompression and idImageProcessing to link to respectively the compression and image processing. An image does not always have compression or image processing, meaning separate tables needed to be created to avoid NULL values in the database. The ImageData and ImageChildData are used to link respectively related and child observations to an image. A one to many relationship is used because it is possible to have more then one related or child observation of an image. The table Data has a one to many relationship with tables ImageData and ImageChildData which means it is also possible to link an observation to more than one image.
4.1.5.2 Queries

When a HTTP GET request for an image description is received the server executes five queries to the database. The first two executed queries are used to get the child and related data of the image.

```sql
SELECT Data.id AS idData, Data.idDataType, ImageData.id, ImageData.idImage FROM commonsense.Data RIGHT JOIN commonsense.ImageData ON commonsense.Data.id = commonsense.ImageData.idData WHERE idImage = 'ID'

SELECT Data.id AS idData, Data.idDataType, ImageChildData.id, ImageChildData.idImage FROM commonsense.Data RIGHT JOIN commonsense.ImageChildData ON commonsense.Data.id = commonsense.ImageChildData.idData WHERE idImage = 'ID'
```

Both queries use a RIGHT JOIN statement of the tables linking the child and related data and the table containing the sensor data to look up all the child and related data of an image.

The next two queries are basic SELECT queries to get the performed compression and image processing on the image.

```sql
SELECT idCompression FROM commonsense.UsedCompression WHERE idImage = 'ID'

SELECT idImageProcessing FROM commonsense.UsedImageProcessing WHERE idImage = 'ID'
```

The last performed query uses a LEFT JOIN statement of the table containing the image data and the stream. The LEFT JOIN is necessary to get the related stream of an image, which is used to describe the stream and link it to a specific sensor node.

```sql
SELECT Images.id, idNode, idStream, idDataType, Streams.description AS streamDescription, date, creator, Images.description, value, title FROM commonsense.Images LEFT JOIN commonsense.Streams ON commonsense.Images.idStream = commonsense.Streams.id WHERE commonsense.Images.id = 'ID'
```

4.2 Multimedia sensor node

This section focuses on the functionalities of the developed WMSN node, its performance and power consumption. The developed multimedia sensor node is based on the Raspberry Pi platform (3.2.1), extended with a Sleepy Pi board (3.2.5). Initially, the intention was to use an iLive board (3.2.2) to extend the Raspberry Pi and form a MiLive...
node (3.2.4). However, at the time of writing the iLive is in the process of patenting, which made it impossible for the research group to make one available for the project. Therefore the Sleepy Pi board is used to take care of putting the Raspberry Pi to sleep. The Raspberry Pi is equipped with a Logitech HD C615 camera to take pictures, used with the Motion [28] software on Raspbian [29]. The Raspberry Pi makes use of a USB WiFi dongle to connect to the internet.

### 4.2.1 Raspberry Pi and Sleepy Pi

The Raspberry Pi is based on the MWiFi (3.2.3) and runs the Raspbian OS. Raspbian is a Debian based OS optimized for use on the Raspberry Pi. The Raspberry Pi is equipped with a Logitech HD C615 camera and an ALFA USB WiFi dongle. Because the iLive node which can be used to form the MiLive together with the MWiFi is in the process of patenting, it could not be used in this project. This meant a different solution had to be found to take care of the power management of the Raspberry Pi. A Sleepy Pi board is used and programmed to wake and sleep the Raspberry Pi on a regular base to take images, but make sure it is not always running so it can be ran on batteries. The Sleepy Pi is also equipped with a sound sensor, which can be used to trigger an interrupt to wake up the Raspberry Pi.

#### 4.2.1.1 Waking and shutting down the Raspberry Pi by RTC

The Sleepy Pi is programmed to periodically wake the Raspberry Pi. It uses two GPIO pins of the Raspberry Pi to perform the handshaking between the devices. Capturing the data is performed by the Motion [28] software, which can be used to detect motion in the captured image, and/or take periodical snapshots.

![Figure 4.9: Flow of Raspberry Pi](image-url)
When the Sleepy Pi is connected to the power supply, it will start its program and supply power to the Raspberry Pi. The Raspberry Pi then boots up and makes a connection with the Access Point defined in the network configuration file. When the Raspberry Pi is connected to the network, it starts a Python script which sets GPIO pin 24 as input and GPIO pin 25 as output. Firstly, the script starts the Motion service to take an image. If the image is taken, the service is stopped and a Node.js script is started to read, convert and embed the image in RDF/XML. When the conversion is completed the image file is deleted. The Node.js script creates an HTTP POST request containing the XML and sends it to the server. After the Node.js script is finished executing, the Raspberry Pi sets GPIO pin 25 to high. The process of the Raspberry Pi is shown in figure 4.9.

![Figure 4.9: Flow of Sleepy Pi](image)

When GPIO pin 25 is set to high, the Sleepy Pi knows the Raspberry Pi is finished executing its tasks. It sets GPIO pin 24 to high, while checking if GPIO pin 25 is still high. The Python script on the Raspberry Pi periodically checks if GPIO pin 25 is high, if this is the case it issues a command to shut down the Raspberry Pi. If the OS of the Raspberry Pi is shut down, the GPIO pin 25 is low, so the Sleepy Pi knows it can safely cut the power to the Raspberry Pi. After the power is cut the Sleepy Pi puts itself in sleep mode until the next interrupt is triggered by the RTC, and the process starts again. The process of the Sleepy Pi is shown in figure 4.10.

### 4.2.1.2 Waking and shutting down the Raspberry Pi by sound sensor

To wake the Raspberry Pi from a sound sensor, a Panasonic WM-61A analog sensor is used. This sensor gives a maximum resistance of 2.2 kΩ, which increases or decreases depending on the sound it measures. The sensor is put in series with a 2 kΩ resistor. The external voltage source of the Sleepy Pi is used to provide a 3.3V voltage load on the resistors. The A0 pin of the Sleepy Pi measures the voltage over the 2 kΩ resistor.
If the sound increases, the resistance of the sound sensor will decrease, putting a higher voltage load over the 2 kΩ resistor. If the voltage measured at the A0 pin of the Sleepy Pi exceeds a certain threshold, it will boot up the Raspberry Pi and perform the flow described in 4.2.1.1. The schematic of the sensor connected to the Raspberry Pi is shown in Figure 4.11.

![Sound sensor schematic](image)

**Figure 4.11:** Sound sensor schematic

### 4.2.2 Power consumption

Reducing the power consumption of the WMSN node is essential to make the device function with a battery. The power consumption of the Raspberry Pi is too high to run it on a standard battery. Therefore, the Raspberry Pi has to be regularly shutdown to maximize battery life.

#### 4.2.2.1 CoAP vs. HTTP

To send the image, the intention was to compare the use of the CoAP and HTTP protocols and their effect on the power consumption. For CoAP the JCoAP library [32] is used, which is a Node.js implementation of the CoAP protocol. Due to the size of the data the CoAP request had to be separated in blocks. However, the block transfer proved to be not working with the JCoAP library.

For HTTP, a request is created and sent by a Node.js script. The HTTP traffic coming from the Raspberry Pi proved to have similar throughput and packet loss as a regular PC, making the data transmission reliable and not very time consuming. Because the power consumption mainly relies on the time the Raspberry Pi is on, it was decided not to continue the use of CoAP for sending multimedia data, and focus on HTTP.
4.2.2.2 Measurements

Several measurements were taken to make an estimation on how long a battery can last. All measurements consist out of 300 samples taken over approximately one minute, and are performed with the HP Agilent technologies 34401A [33]. The current is measured indirectly by measuring the voltage on a 0.1 Ω resistor put in series with the Sleepy Pi, and applying Ohm’s law. The accuracy of the measurements is approximately 10 mA, and is calculated in Appendix C.

The Raspberry Pi in idle state has an average current of 345 mA. No USB devices were attached, ethernet was not active and the operating system had no activity except it’s normal services to run. The graph of the measurement is shown in figure 4.12.

![Power consumption (mA)](image)

**Figure 4.12: Current in idle state**

The next measurement is taken when the camera is attached, but the motion service is not running. The USB WiFi dongle is attached and connected with an access point. The measurement was necessary because it represents the state when the operating system has booted but has not yet started the motion service. The average current in this situation was 552 mA. The graph of the measurement is shown in figure 4.13.

The next measurement is taken when the WiFi is connected with the motion service running. This represents the situation when the camera is taking an image. This increased the average current with 182 mA to 734 mA. The graph of the measurement is show in figure 4.14.
4.2.2.3 Battery life

To calculate the estimated lifetime of the Raspberry Pi and Sleepy Pi, the current is measured over the time it takes to boot up the Raspberry Pi, connect to WiFi, take and send the picture and shut down again.

The measurements showed nine different active loads of the Raspberry Pi, as can be seen in Figure 4.15 and figure 4.16. The ninth load is the load it uses when the Raspberry Pi is powered down and the Sleepy Pi is in sleep mode, and is used as the load when the WMSN node is not active. Tables 4.2 and 4.3 show the average current and time of the different loads.
Figure 4.15: Current when sending and taking a picture

<table>
<thead>
<tr>
<th>Load</th>
<th>Current (mA)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleepy Pi boot</td>
<td>25,734</td>
<td>2.698</td>
</tr>
<tr>
<td>Raspberry Pi boot</td>
<td>293,551</td>
<td>3.665</td>
</tr>
<tr>
<td>Raspberry Pi starting OS and services</td>
<td>464,839</td>
<td>16.78</td>
</tr>
<tr>
<td>Raspberry Pi connecting to WiFi and start python script</td>
<td>632,670</td>
<td>14.265</td>
</tr>
<tr>
<td>Raspberry Pi starting motion service and taking a picture</td>
<td>716,280</td>
<td>17.933</td>
</tr>
<tr>
<td>Raspberry Pi sending the picture</td>
<td>575,506</td>
<td>10.027</td>
</tr>
<tr>
<td>Sleepy Pi requesting shutdown of Raspberry Pi</td>
<td>494,744</td>
<td>2.891</td>
</tr>
<tr>
<td>Raspberry Pi shutting down</td>
<td>160,303</td>
<td>5.402</td>
</tr>
<tr>
<td>Not active</td>
<td>23,293</td>
<td>3526.339</td>
</tr>
</tbody>
</table>

Table 4.2: Active loads measurement 1

<table>
<thead>
<tr>
<th>Load</th>
<th>Current (mA)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleepy Pi boot</td>
<td>28,2765</td>
<td>2.892</td>
</tr>
<tr>
<td>Raspberry Pi boot</td>
<td>298,2295</td>
<td>3.665</td>
</tr>
<tr>
<td>Raspberry Pi starting OS and services</td>
<td>534,108</td>
<td>16.776</td>
</tr>
<tr>
<td>Raspberry Pi connecting to WiFi and start python script</td>
<td>663,3885</td>
<td>13.127</td>
</tr>
<tr>
<td>Raspberry Pi starting motion service and taking a picture</td>
<td>800,806</td>
<td>17.551</td>
</tr>
<tr>
<td>Raspberry Pi sending the picture</td>
<td>594,12</td>
<td>7.328</td>
</tr>
<tr>
<td>Sleepy Pi requesting shutdown of Raspberry Pi</td>
<td>481,7245</td>
<td>1.736</td>
</tr>
<tr>
<td>Raspberry Pi shutting down</td>
<td>159,49</td>
<td>5.396</td>
</tr>
<tr>
<td>Not active</td>
<td>23,293</td>
<td>3531.53</td>
</tr>
</tbody>
</table>

Table 4.3: Active loads measurement 2

With the formula 3.1 described in section 3.2.8 the WMSN node has an estimated battery life of approximately 297.47 hours (12,40 days), provided the node boots the Raspberry Pi every hour and a 5V battery with a 10000 mAh capacity is used.

Calculated with the same battery capacity and formula the second measurement showed an estimated battery life of 297,01 hours (12,38 days). The measurements show that the
Chapter 4. Development of the proposal / technical / work

Figure 4.16: Current when sending and taking a picture

<table>
<thead>
<tr>
<th>Frequency (m)</th>
<th>Lifetime m1 (h)</th>
<th>Lifetime m2 (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>117.32</td>
<td>116.89</td>
</tr>
<tr>
<td>20</td>
<td>184.28</td>
<td>183.75</td>
</tr>
<tr>
<td>30</td>
<td>227.58</td>
<td>227.04</td>
</tr>
<tr>
<td>120</td>
<td>351.43</td>
<td>351.11</td>
</tr>
<tr>
<td>180</td>
<td>374.05</td>
<td>373.81</td>
</tr>
</tbody>
</table>

Table 4.4: Battery lifetimes

WMSN node has a minimum battery lifetime of approximately 12 days. The battery lifetime changes if the frequency of taking the images is increased or decreased. Table 4.4 shows the estimated battery lifetime with different frequencies of taking images.

4.2.2.4 WiFi vs low-power WiFi

The difference in power consumption between the USB WiFi dongle and a low-power WiFi module is also measured. The average current when using the USB WiFi dongle is 499.30 mA, while the low-power WiFi module has an average current of 411.09 mA, showing the low-power WiFi to be more power efficient. However, it is possible the low-power WiFi would take more time to associate with an access point and/or transmit the data. The Raspberry Pi also has to communicate with the low-power WiFi module over a serial connection, making this a more difficult task, and most likely more time consuming. This means it is important to study the trade-off between using less power over a different amount of time. Due to lack of time, low-power WiFi is not further implemented.

Figure 4.17 provides a graph of the current using regular WiFi, and figure 4.18 provides a graph of the current using low-power WiFi.
Figure 4.17: Current using regular WiFi

Figure 4.18: Current using low-power WiFi
Chapter 5

Evaluation of the proposal / technical / work

5.1 Objectives

This section provides an evaluation of the achieved objectives, as well as the problems occurred in the research. Furthermore, it provides an overview of the objectives which could not be implemented.

5.1.1 Adaption of CommonSense for multimedia traffic

The CommonSense platform is successfully adapted to support multimedia traffic. A wireless multimedia sensor node can send image data to the platform, which is then stored in the database. The platform can be queried for the image metadata in RDF format, as well as provide the actual image the metadata describes. The image is made available through the CommonSense platform via an URL, which can be retrieved from the RDF document describing the image. Since there is no ontology to describe images in a WMSN context, a new ontology is developed providing the necessary classes and properties. The ontology also provides properties to link sensor data to an image, although this is not fully implemented in the CommonSense platform. The server can collect and make the related data available in the image RDF document, however it is not yet possible for the platform to recognize that certain incoming sensor data should be linked to an image.
5.1.2 Creation of a Wireless Multimedia Sensor Node

A Raspberry Pi based wireless multimedia sensor node is developed to capture and send image data to the CommonSense platform. Several measurements are performed to measure the power consumption and calculate the battery lifetime. Due to the use of the Sleepy Pi board, the power consumption of the Raspberry Pi is drastically reduced. This makes it now possible to power the Raspberry Pi from batteries, although the power consumption relies heavily on the frequency of capturing the images. However, the Sleepy Pi has a relatively high power consumption when in sleep mode, draining the battery when it is not active.

The use of CoAP instead of HTTP is attempted, but was not successful due to some functionalities which were not working in the used JCoAP library. Because addressing this issue proved to be time consuming, this approach is not tested. The low-power WiFi module is not implemented because of lack of time, although this might slightly improve the battery lifetime of the sensor node. Furthermore, the sensor node can not be powered by the DC input of the Sleepy Pi, so it is necessary to power it at the 5V USB input. When connecting the DC input to a power source, the fuse of the Sleepy Pi blows, making it impossible to use this input. This issue is still being addressed with the manufacturer, but at the time of writing this is not yet resolved.

5.2 Future work

This section provides an overview of the possible extensions and improvements to the CommonSense platform and the wireless multimedia sensor node.

5.2.1 CommonSense

As explained in 5.1.1, the CommonSense platform can not yet link incoming sensor data to an image. A solution has to be found on how the platform will know data coming from a particular sensor node is related to an image. When sending the multimedia data, a property can be used to provide a link to related data. However, since the ID (and therefore the URL) is chosen when the data is received at the server, the multimedia sensor node has to know this ID to include it in it’s RDF document. A possible solution is to link a sensor node to the multimedia sensor node in the CommonSense platform, so the platform can create the link between the data based on the time of the reading.
5.2.2 Wireless Multimedia Sensor Node

To further improve the battery lifetime of the multimedia sensor node, it can be adapted so it makes use of low-power WiFi instead of regular WiFi, which is more power efficient. Moreover, the use of CoAP can possibly reduce power consumption, although it is unlikely the performance of CoAP would be greater than HTTP on the Raspberry Pi, since the Raspberry Pi handles HTTP very well. The use of a MiLive (see section 3.2.4) can also greatly reduce power consumption, due to the power consumption of the iLive (see section 3.2.2) being far less than the consumption of the Sleepy Pi. Moreover, the multimedia sensor node could be used as a gateway to the internet for other sensor nodes.
Chapter 6

Conclusion

The CommonSense platform is adapted to support images. Images can be sent to the platform in RDF format, where they are stored and made publicly available in RDF format by HTTP queries. The RDF documents contain all the image metadata, as well as the URL to download this image. To describe images in a WSN context, a new ontology is developed and made public on the CommonSense website.

A Raspberry Pi based WMSN node is developed to capture and send an image to the platform. The WMSN node is equipped with a camera to take the images and a WiFi dongle to connect the node to the internet. To make sure the WMSN node can be powered from a battery, a Sleepy Pi is used to take care of the power management. The Sleepy Pi powers down the Raspberry Pi when it is idle, making sure the power consumption is drastically reduced.
Appendix A

Acronyms

**WSN** Wireless Sensor Network

**WMSN** Wireless Multimedia Sensor Network

**RDF** Resource Description Framework

**SSN** Semantic Sensor Network

**DC** Dublin Core

**GPIO** General-purpose input/output

**API** Application Programming Interface

**OWL** Web Ontology Language

**HTTP** Hypertext Transfer Protocol

**HTML** Hypertext Markup Language

**URI** Uniform Resource Identifiers

**CoAP** Constrained Application Protocol

**PMU** Power Management Unit

**PWM** Pulse-Width Modulation

**TCP** Transmission Control Protocol

**UDP** User Datagram Protocol

**REST** Representational State Transfer
Appendix A. *Acronyms*

**SSI** Semantic Sensor Images

**OLD** Open Link Data

**Exif** Exchangeable image file format

**XMP** Extensible Metadata Platform

**IPTC** International Press Telecommunications Council

**XML** Extensible Markup Language

**RTC** Real Time Clock
Appendix B

WMSN node user manual

B.1 Connecting the boards

The Sleepy Pi has to be connected through the Raspberry Pi GPIO pins. Once connected, the jumper on the Sleepy Pi should not connect the two pins marked with 'force on'. If these pins are connected the power to the Raspberry Pi is always on, so the Sleepy Pi cannot turn down the power. Power to the Sleepy Pi has to be supplied through 5V micro USB input of the Sleepy Pi.

Figure B.1 shows how the sound sensor has to be connected to the Sleepy Pi.

![Sound sensor schematic](image)

Figure B.1: Sound sensor schematic
B.2 Setting up the Raspberry Pi

To set up the Raspberry Pi, an ssh connection has to be made. To give the Raspberry Pi ethernet interface an IP address from boot, the following line has to be added to the `cmdline.txt` file on the SD-card of the Raspberry Pi.

```
ip=XXX.XXX.XXX.XXX
```

When the Raspberry Pi is booted, you can connect via ssh with the user `pi` and password `raspberry`. If no screen is available, `tightvnc` can be used to set up a VNC server on the Raspberry Pi with the following command.

```
tightvncserver :1
```

The Python script performing the necessary actions has to be set to run at startup. The following line has to be added to the `/etc/rc.local` file.

```
python /home/pi/bin/button/shutdowncheck.py &
```

On a fresh PI install you will need to set the permissions on `/etc/rc.local` so that it can execute at startup. This can be done by executing the following command.

```
sudo chmod u+x /etc/rc.local
```

To make sure the node can connect to a WiFi network the following lines have to be added to the `/etc/network/interfaces` file.

```
auto wlan0
iface wlan0 inet dhcp
wpa-essid "SSID"
wpa-psk "password"
```

For more complex WiFi setup, WPA supplicant [34] can also be used, although the command to connect to the WiFi network has to be included in the Python script before the command to start the Motion service (see section 4.2.1.1) is executed.
B.3 Setting up the Sleepy Pi

The Sleepy Pi has to be programmed directly from the Raspberry Pi via the Arduino IDE. A full guide on how to install the Arduino library is available at the Sleepy Pi website [22]. It is important to connect the jumper wires on the Sleepy Pi to make sure the Raspberry Pi does not lose power from the Sleepy Pi when it resets. When programming the Sleepy Pi, it is also necessary to disable the Python script at startup (an easy way to do this is by changing the name of the file).

To use the RTC to wake up the Sleepy Pi, the `WakeUpPiOnRTC.ino` file has to be compiled and loaded on to the Sleepy Pi. To change the frequency of taking an image, the variable `RTC_time` in the Arduino scripts can be set (in seconds).

To use the sound sensor to wake up the Raspberry Pi, the `WakeUpPiOnAlarmExt.ino` file has to be compiled and loaded on to the Sleepy Pi. To change the threshold when the Sleepy Pi has to power on the Raspberry Pi, the variable `threshold` can be changed. This should be a number between 0 and 1023.

If the Raspberry Pi is shutdown and the Sleepy Pi is reset, the device will start doing it’s cycles. It is important to delete the line in the `cmdline.txt` file to set the ethernet IP address, because this slows down the process of the Raspberry Pi.
Appendix C

Accuracy calculations of measurement

The accuracy related with current is approximately 10 mA and is calculated with formula C.1. The accuracy is calculated for an average current of 500 mA.

\[ u(I) \approx \frac{\sigma I}{\sigma V} \times u(V) + \frac{\sigma I}{\sigma R} \times u(R) \]  

\( \frac{\sigma I}{\sigma V} \) and \( \frac{\sigma I}{\sigma R} \) are calculated in equations C.2 and C.3

\[ \frac{\sigma I}{\sigma V} = \frac{1}{R} = \frac{1}{0.1\Omega} = 10 \frac{1}{\Omega} \]  

\[ \frac{\sigma I}{\sigma R} = \frac{1 \times V}{R^2} = \frac{0.05V}{(0.1\Omega)^2} = 5 \frac{V}{\Omega^2} \]  

The accuracy of voltage and resistance are calculated in equations C.4 and C.5. The fixed values used in this equations are specified in the datasheet of the HP Agilent technologies 34401A [33].

\[ u(V) = \frac{0.005 \times V + 0.0035 \times R}{100 \times 4} = \frac{0.005 \times 0.05 + 0.0035 \times 0.1}{100 \times 4} = 1.5 \times 10^{-6}V \]  

\[ u(R) = \frac{0.010 \times R + 0.004 \times 100\Omega}{100 \times 4} = \frac{0.010 \times 0.1\Omega + 0.004 \times 100\Omega}{100 \times 4} = 0.001\Omega \]
The standard deviation $u(I)$ is calculated in equation C.6.

$$u(I) = \sqrt{\frac{\sigma_I^2}{\sigma_V^2} \times u(V)^2 + \frac{\sigma_I^2}{\sigma_R^2} \times u(R)^2} = \sqrt{100 \times 2.25 \times 10^{-12} + 25 \times 10^{-6}} \approx 5mA$$  

(C.6)

A standard deviation of $2\sigma$ is used for the accuracy as can be seen in calculation C.7.

$$U(I) = K \times u(I) = 1.96 \times 5mA \approx 10mA$$  

(C.7)
Bibliography


