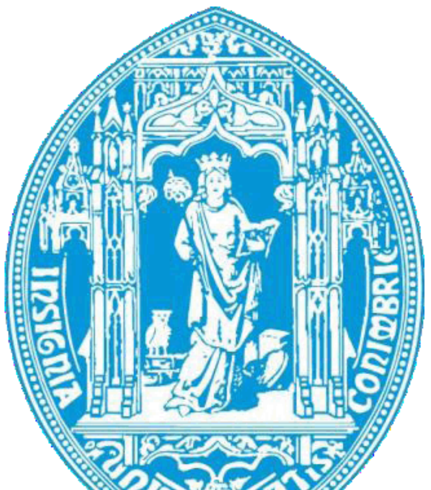
Critical health

Health Monitoring Sensor Suppliers Integration

David Sousa Nunes

Faculty of Sciences and Technology

University of Coimbra 2009/2010



Abstract

Some of the elderly population is unable to live on their own and require assistance and supervision. Assisted Living facilities ensure their health and well being through supervision of their vital conditions, helping with medication routines and assisting them whenever necessary. Wisedome is a platform that enables residents in Assisted Living facilities or Long-Term Care institutions to be continuously monitored by using sensor devices, warning caregivers of potential harmful situations such as elderly patient falls or changes in cardiac heart rate.

In order to provide its customers with the best state-of-the-art sensors, the Wisedome health monitoring platform should be sensor agnostic and integrate multiple types of sensors from different hardware suppliers, that adapt to different types of senior patients with different necessities.

The goal of this project was to contribute to the development and evolution of the Wisedome health monitoring application. This overall goal can be divided into three main tasks: research of industry coalitions that try to achieve interoperability between sensor devices; extension of Wisedome's currently functionality by enabling it to not only use different types of sensors from different vendors, but to also be able to use them simultaneously and dynamically; contributions in the software's quality management system, namely in requirement analysis and system testing.

This document serves as a report on the performed tasks and contributions made to the Wisedome platform during this project's execution.

Resumo

Alguma da população idosa é incapaz de viver de uma maneira independente e requer algum tipo de apoio e supervisão. Instituições que fornecem assistência ou cuidados a longo prazo a idosos servem o propósito de garantir o bem-estar e a saúde de pessoas da terceira idade, tendo em atenção os seus sinais vitais, garantindo a medicação apropriada e ajudando nas suas necessidades diárias. A plataforma Wisedome permite que os idosos nestas instituições tenham os seus sinais vitais monitorizados de forma contínua recorrendo ao uso de sensores, alertando os profissionais prestadores de cuidados de situações potencialmente perigosas como quedas de um idoso ou alterações do seu ritmo cardíaco.

De maneira a garantir que os seus clientes tenham sempre os melhores e mais actuais sensores, a plataforma de monitorização de saúde Wisedome deverá ser independente do tipo de sensores utilizado e permitir integrar vários sensores de diferentes fornecedores, que se adaptem a diferentes tipos de idosos com diferentes necessidades.

O objectivo deste projecto foi o de contribuir para o desenvolvimento e evolução da aplicação de monitorização Wisedome. Este objectivo pode ser dividido em três grandes tarefas: pesquisar as principais alianças que procuram atingir a interoperabilidade entre sensores de saúde; melhorar as funcionalidades da plataforma Wisedome para que esta possa vir a suportar não só o uso de diferentes tipos de sensores, mas que os suporte simultaneamente e de uma forma dinâmica; contribuir para o controlo de qualidade do software, nomeadamente em termos análise de requisitos e testes de sistema.

Este documento serve de relatório das tarefas desempenhadas e das contribuições feitas para o desenvolvimento do Wisedome durante a execução deste projecto.

Acknowledgments

I would like to express my gratitude to all people that directly or indirectly made this project's completion possible.

A big thank you to all of my current and former colleagues at Critical Health, for the fun environment, activities and team spirit by them provided, which made me feel an integrate part of the team.

I would like to express my gratitude in particular for the support of all the members of the Critical Health's Health Monitoring team. I am in debt to Tânia Baptista for her patience and help during my time working on quality management and to Bernardo Raposo for his promptly assistance whenever I required it. A very special thank you is due to André Lemos, for all his patience, invaluable guidance, critical reviews and supervision throughout my internship at Critical. Another special thank you is due to Gregory Stern for being responsible for this project's existence, for the opportunity to work in Critical Health's health monitoring team and for his guidance and supervision throughout my time at Critical.

I am also in great debt to Professor Jorge Sá Silva, for undertaking the task of being my project's mentor, for all his assistance during the project's execution and for the opportunities that he gave to me.

I would also like to express my gratitude to Professor Miguel Morgado, for accepting this project's idea, managing the creation process and making the project happen.

I also own a great deal to my closest friends, who always stood by me and humored my life, as I would certainly not be able to enjoy it as much as I do without them.

For last, but not least, I would like to thank my mother, father and brother for their unconditional support during my whole life and for being the main pillar that maintains my existence in balance.

Index

Acronyms

AFH	Adaptive Frequency Hopping
API	Application Programming Interface
CLH	Cluster Head
CMISE	Common management information service
Continua	Continua Health Alliance
CSMA	Carrier Sense Multiple Access
CVS	Concurrent Versioning System
DSSS	Direct Sequence Spread Spectrum
ECG	Electrocardiogram
FDMA	Frequency Division Multiple Access
GHz	Gigahertz (1GHz = 10^9 Hertz)
HR	Heart Rate
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISM band	industrial, scientific and medical band
ISO	International Organization for Standardization
JAR	Java ARchive
JVM	Java Virtual Machine
Kb	Kilobits (1Kb = 10^3 bits)

Kbps	Kilobits Per Second
LDAP	Lightweight Directory Access Protocol
MAC	Medium Access Control Sub-layer
Mbps	Megabits Per Second (1 Mb = 10^6 bits)
MHz	Megahertz (1MHz = 10^6 Hertz)
ms	Millisecond (1 ms = 10^{-3} seconds)
OSI model	Open Systems Interconnection model
PAN	Personal Area Network
PHY	Physical Sub-layer
POJO	Plain Old Java Object
QMS	Quality Management System
RF	Radio Frequency
SNMP	Simple Network Management Protocol
SOA	Service Oriented Architecture
SSH	Secure Shell
TCP	Transmission Control Protocol
USB	Universal Serial Bus
XHR	XMLHttpRequest
XML	Extensible Markup Language
μ s	Microsecond (1 μ s = 10^{-6} seconds)

List of Figures

1 Introduction

This document is a thesis for a master degree in Biomedical Engineering at the University of Coimbra. It acts as a report of a research project developed in an internship at Critical Health, a spin-off of Critical Software. Its objectives are to present the work there developed and also to serve as a reference for future developments and improvements on Wisedome's health monitoring sensor integration effort.

1.1 Motivation

Advances in the medical field allow human beings to live better for longer periods of time. This trend results in an increase of the elderly people population. While some of the elderly are able to live independently on their homes, there are also many who are unable to live on their own and require assistance and supervision. Assisted Living facilities ensure their health and well being by supervising their health condition; medication routines and making sure that their personal needs are satisfied.

Wisedome is a platform that is being developed by Critical Health that enables residents in Assisted Living facilities to be monitored using wireless wearable sensors to automatically detect falls, capture heart rate and monitor elderly patient's location within the facility. Patients are also able to call for assistance from a nurse by pressing a built-in button on their sensor. It is desirable for a health-monitoring platform such as Wisedome to be able to be sensor agnostic designed to be used with multiple sensors from multiple hardware providers. In order to support their clients with the best state-of-the-art sensors, the Wisedome platform should be able to integrate multiple types of sensors, in order to provide solutions for different types of senior patients: independent, dependent, rehab and dementia seniors.

1.2 Project Overview and Objectives

Critical Health is a spin-off of Critical Software encompassing all the healthcare activities that were being developed since 2006. Critical Health was created to improve the Quality of Life and to reduce the total Healthcare Spending. It aims to achieve this vision by providing critical health information to everyone through the marketing of innovative and accessible technological products.

Wisedome is Critical Health's Health Monitoring Solution for Assisted Living Facilities. At the beginning of the project here documented, Wisedome was unable to properly integrate various health monitoring devices from different vendors, only being able to effectively work with a set of sensor devices (caregiver device, chest-strap and bracelet sensors) from a single hardware vendor. This project was born from an effort to improve the Wisedome platform, allowing it to support different kinds of devices from different vendors, in order to increase the platform's flexibility and increase its value. This project was not limited to work on integration efforts, however, but also included work in other areas of Wisedome development and quality management, namely Requirement Analysis and Software Testing.

The objective of this project was to contribute to the development and evolution of the Wisedome health monitoring application. Its main component is related to the improvement of Wisedome's capability of using multiple sensor devices, firstly with a research effort on the major health monitoring industry coalitions and their hardware integration efforts, and secondly on the development of an integration framework that facilitates the integration of new types of device hardware with the Wisedome system. A secondary component is related to contributions to Wisedome's quality management system, where work on requirement definitions and system testing was performed.

1.3 Audience

This document's target audience is Critical Health's collaborators that will continue to work on the Wisedome platform and this project's juries and supervisors, as well as the biomedical and informatics academic community.

1.4 Document Structure and Organization

This document's structure comprises six major chapters. Chapter 1 serves as an introduction to the project and outlines its context and major objectives. Chapter 2 provides a general description of the Wisedome system, which is the software to where the work described on thesis was directed. Chapter 3 presents the research made on two of the main health industry coalitions and describes the technologies adopted by each one. Chapter 4 is dedicated to the description of the design and implementation process of a device integration system made for the Wisedome platform. Chapter 5 explains the tasks performed in Wisedome's requirement analysis and software testing. Finally, chapter 6 presents the conclusion and personal opinions on project and also a future work analysis.

2 Wisedome Health Monitoring System

Wisedome is the application where most of the work presented in this thesis is directed to. This section will describe the application, its purposes and architecture.

Figure 1 – Wisedome's Logo.

2.1 Wisedome General Description

Wisedome is a Senior Health Monitoring application currently being developed by Critical Health and directed towards Assisted Living Facilities or Long-Term Care institutions. It consists on a system that monitors elderly patient well being and raises alarms for caregiver personnel whenever a potentially hazardous situation occurs, such as a patient's heart beat being too high or a patient's fall. It monitors the elderly by having them wear various sensors that continuously feed information to the health monitoring application. These sensors have hardware capable of monitoring various vital signs, possess a button that can be pressed for requesting a caregiver's help and also have a locking mechanism that straps them on to the patient. Caregivers also possess devices that warn them whenever there are unresolved active alarms. There are also various alarm screens spread throughout the building that show information about currently active alarms. The system also serves as a tool for documentation of a patient's situation, allowing caregivers to register notes, document alarms or edit patient's information and share this data with other caregivers. Wisedome has three primary goals (1):

- To improve quality of care of elderly people in residential living.
- Save caregivers' time by providing them with tools to document and store important information or notes.
- To alert caregivers of patient alarms and potential risk situations and to allow alarm detection parameters to be individually configured for each patient.

Wisedome developed its adopted features based on the functionalities provided by a few reference vendors. One of these reference vendors was Plux, which will be used as an example for this point onwards. Wisedome is capable of monitoring various patient conditions and raise alarms when these conditions stray out of predefined states. Wisedome is currently capable of warning caregivers if a patient has fallen, if a patient's heart rate is higher or lower than normal, if a patient has called for assistance (by pressing a button on his/hers sensor device) or if a patient has entered a restricted area. It will be capable of monitoring additional alarm conditions in the future with the addition of new types of sensor devices. Additionally to the alarms, Wisedome also warns caregivers of several system related events, such as sensor device's battery running low (average battery life for Plux devices is three days), communication with a sensor being lost or if a sensor's locking mechanism has been opened when it wasn't supposed to be (by a dement patient trying to remove the sensor, for instance). Alarms and events can be classified as low priority and high priority: call for help button alarm or low battery events are considered low priority, while a falls or excessive heart rate value alarms are considered high priority. Low priority events can, however, be promoted to high priority, in case they are unattended for a certain period of time.

A device may be in an un-assigned (inactive) or assigned (active) state. A sensor device is always assigned to a given patient, and when in an assigned state, the device should be locked onto the patient in order to start monitoring the patient's condition. Assigned devices are capable of detecting alarm situations and issue alarm events to the Wisedome application, which will then process the event and warn the caregiver personnel. Wisedome currently supports two types of sensor devices: bracelets and chest-straps. Bracelets are the most basic type of device and are worn in the patient's wrist; they simply possess a call button, which may be pressed by the elder to request a caregivers assistance. Chest-straps are worn in a patient's torso and in addition to the call button, they also possess an accelerometer capable of detecting patient's falls and electrodes capable of detecting a patient's heart rate.

Caregivers can also be assigned with devices that are designed to warn them of alarm situations that require their attention. The devices are equipped with a light and beeping mechanisms and emit light signals and beep frequently in order to draw the caregiver's attention. The devices are, however, incapable of giving any additional information about the nature of the alarm. To this purpose, exist the Wisedome's alarm displays and workstations, which are spread throughout the residential care building. The alarm displays are screens that give information about the active alarms' nature, priority and associated patient, as shown in Figure 2. They also emit sounds when there are active alarms, in order to draw caregiver's attention.

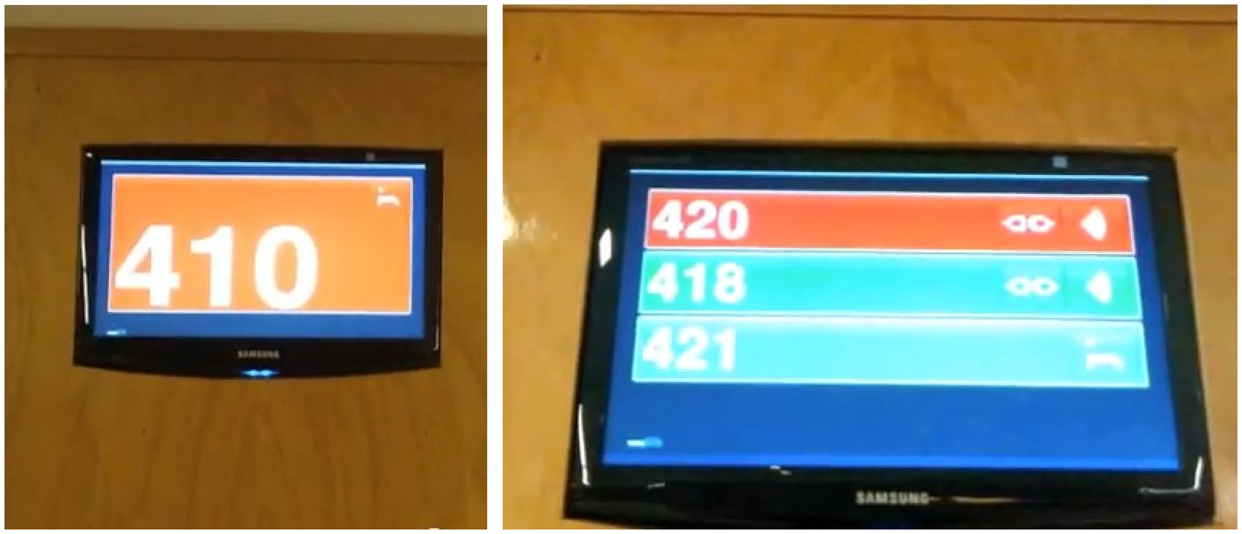


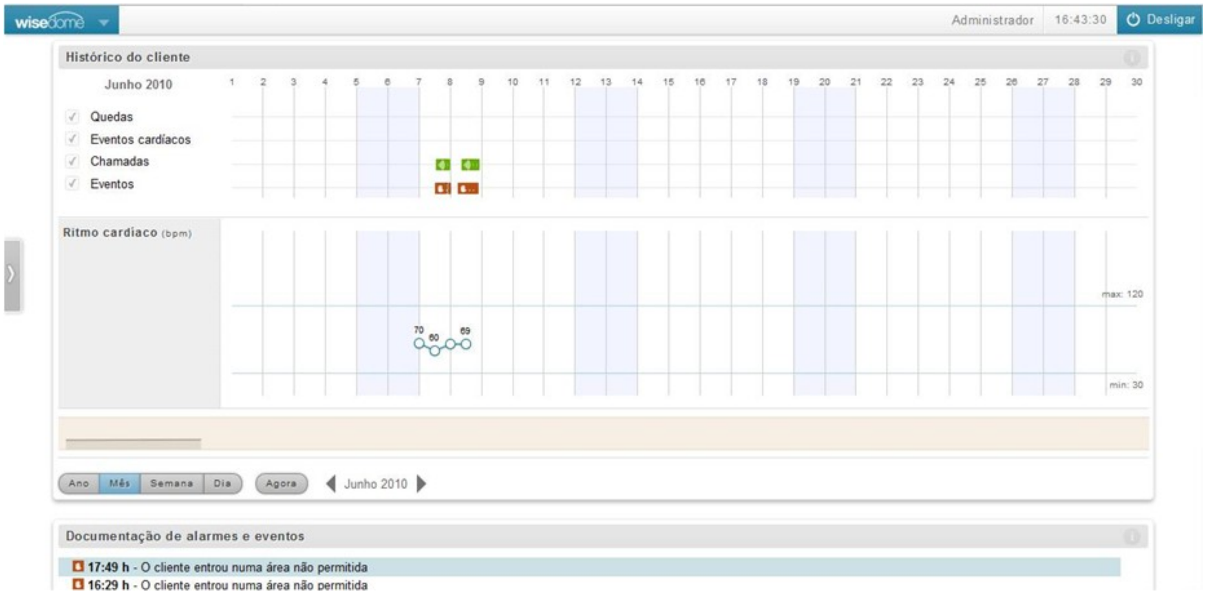
Figure 2 – Wisedome's alarm displays. In the left showing a single high priority alarm and in the right showing two low priority alarms and a high priority alarm

Alarm displays give information about alarms that occur in different floors of the building, with each floor divided in several wings. Alarms and events are separated by the location where they occurred, and the system first tries to notify caregivers assigned to the same location of the alarm. Only if an alarm is not promptly addressed by the local caregivers will the system notify caregivers of a different wing or floor. Caregivers are also divided in two categories: TAPs (*Técnico de Apoio Permanente*) and nurses. TAPs are technicians that attend to the elderly person's needs while nurses are more responsible for health concerns. Low priority events are only sent to TAPs, while high priority events are sent to both TAPs and nurses.

Alarm situations are resolved by having a caregiver acknowledging them. Caregivers can resort to two mechanisms for resolving an alarm: they can either use their device or a workstation. Caregiver devices possess a button that allows them to perform an acknowledgment sequence, which consists on a sequential pressing of their device's button and the patient's device button, thus resolving the active alarms for that patient. Caregiver devices can also be used to initiate a patient's escort sequence, allowing caregivers to escort patients through restricted areas without raising any alarms.

Alternatively, workstations can be used to acknowledge and resolve active alarms, but they are also used to perform many other system management tasks. Workstations are used by nurses to see detailed information about every alarm currently active in the system. Nurses can silence alarms in the workstation, meaning that nurse devices, patient devices and alarm displays will no longer emit sound for a given alarm, although the alarm remains active. The workstation can also be used to, as mentioned above, effectively resolve alarms, allowing nurses to document them, leaving notes that may depict what was the source of the alarm or what actions were taken to resolve the situation, for instance, what medication was given to the patient. It is also possible suspend a patient's device using the workstation so that it won't raise any alarms until it is resumed, allowing nurses to escort patients. It is also through the workstation that nurses can assign or un-assign patient sensors.

The workstation lists all patients currently using the system, allowing nurses to consult and edit each patient's personal information and configuration parameters. Nurses can, for instance, edit the maximum and minimum permitted hear rate values for a given patient, set the patient's fall detection sensitivity (low, medium or high) or configure what areas of the building the patient is allowed in.



17:49 h - O cliente entrou numa área não permitida
16:29 h - O cliente entrou numa área não permitida

Figure 3 - Wisedome's patient chart, where heart rate and event history are displayed.

Nurses can also consult the patient chart page, which contains the history of measured heart rates and raised alarms for a given patient. In this view, nurses can add custom events and notes or commentaries for previously resolved events. This input is stored, allowing nurses to easily share important information with other nurses during shift change. It also allows them to classify alarms as false alarms.

2.2 Wisedome's Technology Architecture

In order to give a greater insight of the technologies employed in the Wisedome system, this section will talk about its technological architecture.

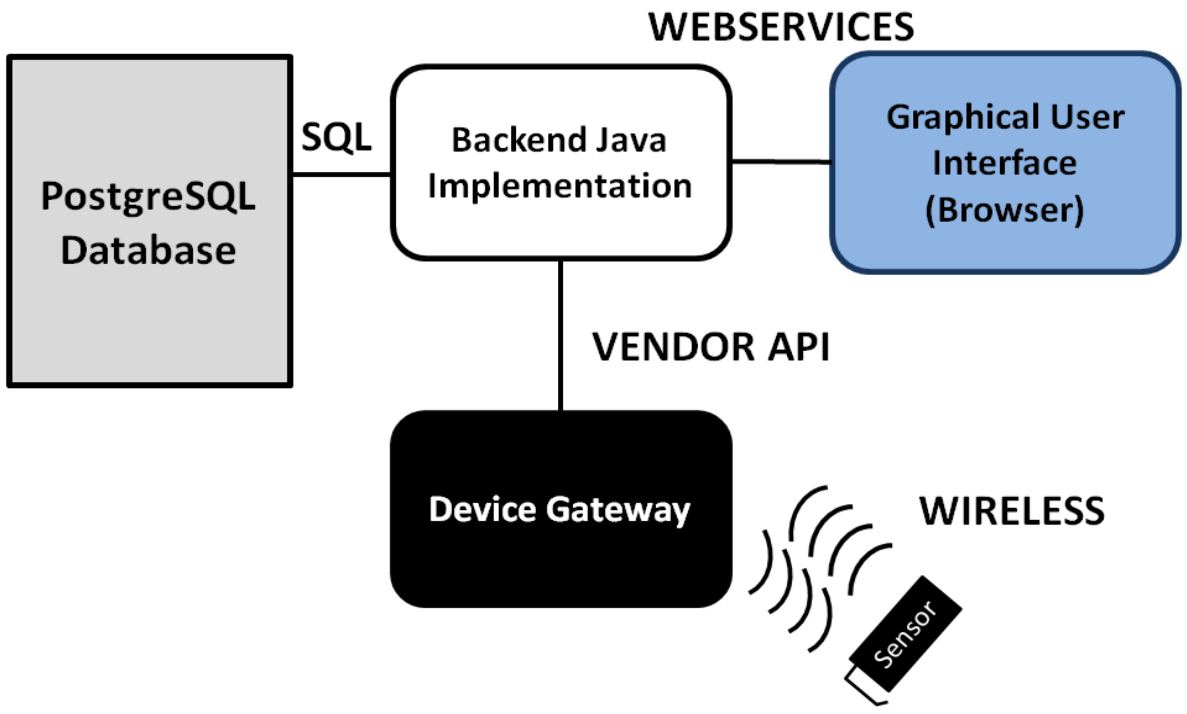


Figure 4 - Wisedome's Technical Architecture.

As seen in Figure 4, Wisedome is comprised of several components, each having its own tasks:

- **PostgreSQL Database –** This is Wisedome's database. The database stores information that is important to the whole system, such as patient personal information, a patient's unique identification number (patient's ID), user credential information, device information (including the device's serial number), device current state (for instance, if it is locked or not), system event history, and configuration values. It communicates with the rest of the system through Structured Query Language (SQL).

- **Backend Java Implementation** - This is Wisedome's "brain", its core. It's where the database information, alarm and device management occurs. The application was developed using Java programming language. It communicates with all the other parts of the system: several classes call upon the database for querying information; two communication classes receive events from and send commands to the device gateway in order to communicate with the sensor devices; the graphical user interface uses webservices to communicate with the backend and process user requests.
- **Graphical User Interface** – This is Wisedome's "face", the part of the system users effectively see and use. It consists on a browser application where users input and request information. It also allows users to manage the whole system. It doesn't perform any processing tasks by its own, so it communicates with the Backend Java Implementation through webservices in order to satisfy user requests.
- **Device Gateway –** The device gateway is a part of the system that bridges Wisedome's communication with the sensor devices. It may exist as a separate application inside Wisedome's machine, be a part of Wisedome or even exist in a separate machine. Its nature depends on the sensor's hardware manufacturer. It offers Wisedome various services, usually through a proprietary set of development tools or communication language, known as the vendor's Application Programming Interface (API). It allows Wisedome to communicate with the devices, in order to issue commands or receive events. The communication with the devices is made through a wireless technology, of the hardware vendor's choice.

These 4 elements compose the Wisedome Health Monitoring system. Most of the work described in this thesis will focus on the device gateway, backend implementation, and on the database components. The next sections will each focus on one aspect of the work dedicated to improving the Wisedome platform during this project's execution.

3 Alliances and Technology Research

One important strategy for Wisedome is to support multiple types of sensor devices. Currently, there exist several health industry alliances or coalitions that are composed by many health device manufacturers. These alliances define how Devices produced within the alliance should communicate with each other. If Critical Health joined one of these alliances, it would have access to the documentation and guidelines that make possible for Wisedome to communicate with every device certificated by that alliance. It is then important to know the most significant alliances on the market, their supported technologies and what variety of sensors do they offer. The health monitoring industry has several on-going trends, each trying to achieve one purpose: system interoperability. Interoperability is achieved when different devices, manufactured by different vendors, are able to communicate and work together. In health monitoring, interoperability is very important since it may involve many different types of devices: heart rate sensors, ECG sensors, thermometers, scales and so on. To establish a standard communication protocol so that both devices and applications are able to work together is one of the primary goals of health care industry coalitions. Joining an established industry coalition and abiding by the agreed standards should ensure interoperability with any device that was certified by the coalition and abides by the same standards. This interoperability between many devices belonging to a single coalition is desirable for an application such as Wisedome. Thus, there was time dedicated in this project to an analysis and comparison of the current trends in the health monitoring sensor market to facilitate the decision of picking the most appropriate industry coalitions to join and technologies to adopt. This chapter's purpose is to present the results of this research and also to comment and present some conclusions in a form of a comparison matrix.

3.1 The Necessity for Standards and Industry Coalitions

Standards are an important part of health care systems. One of today's healthcare industries goals is to drive patient related information to be exchanged freely between the various systems in the continuum care (2). One area of these systems is the health monitoring information, in which various types of sensors gather and send patient data. Integration and data sharing between various types of devices, personal health records or alarm management platforms enables new types of health monitoring applications such as home care and residential care solutions. In today's health monitoring market, devices and software platforms are manufactured and developed by many different entities. In order for a health monitoring system, such as Wisedome, to be able to support a wide range of vital signs, it is necessary to use a wide variety of measurement devices such as blood pressure monitors, glucose meters, heart rate monitors, weighing scales, ECG sensors and fall detection accelerometers. For each type of device there are a number of companies manufacturing them, but no company makes all of these devices. Therefore, it is necessary for systems such as Wisedome to work with many different suppliers in order to provide a complete range of measurement devices to its customers. Unfortunately, unlike other market areas such as the financial industry, healthcare industry is still in its primordial phases when it comes to defining interoperability standards (2). Particularly when it comes to wireless health monitoring sensor networks, there is yet to appear a definitive standard that is adopted by the majority of the industry. Instead, many sensor manufactures use their own proprietary solutions for device communication messages. The proprietary solutions are closed and the majority lacks interoperability, thus making difficult their integration on large scale health monitoring systems. Different device suppliers use different communication technologies (Wi-Fi, Bluetooth, ZigBee, etc.) and even if they all used the same technology for communication they would still use different ways of structuring the messages transmitted over the transport technology (3). Thus, a system that seeks to implement different types of sensor devices is left with the daunting task of integrating different types of messages and technologies, a task that gave birth to Wisedome's Integration Framework, later described in section 4 of this thesis. Without standards that define how to structure messages in order to share information and what technologies to use for communication, it will never be possible to truly achieve system interoperability. Fortunately, there are various attempts at defining what technologies and communication standards are to be used between the various products in the health care industry: industry players gather to develop and agree upon guidelines or profiles that define which standards to use on a given situation and how to combine them in order to achieve interoperability. Collaboration between these entities and the use of global standards is essential for the future of health monitoring systems (2).

In order for Wisedome to benefit from interoperability with devices produced by entities belonging to an industry coalition, it is necessary that Critical Health joins such a coalition. Therefore, it was necessary to determine what technologies and standards are best suited for Wisedome's needs. It was decided that the research would focus primarily on two important health monitoring industry coalitions, the Continua Health Alliance and the ANT+ Alliance, and their respective sensor wireless technologies of choice: ZigBee, Bluetooth and ANT.

3.1.1 Continua Health Alliance

Continua Health Alliance defines itself in its website as:

> *"(…) a non-profit, open industry coalition of the finest healthcare and technology companies joining together in collaboration to improve the quality of personal healthcare. With more than 200 member companies around the world, Continua is dedicated to establishing a system of interoperable personal health solutions with the knowledge that extending those solutions into the home fosters independence, empowers individuals and provides the opportunity for truly personalized health and wellness management (4)."*

Figure 5 – Continua Health Alliance Logo.

The founding members of the group include BodyMedia, Cisco Systems, GE Healthcare, IBM, Intel, Kaiser Permanente, Medtronic, Motorola, Nonin Medical, Omron Healthcare, Panasonic, Partners HealthCare, Polar Electro, Royal Philips Electronics, RMD Networks, Samsung Electronics, Sharp, The Tunstall Group, Welch Allyn and Zensys (5). The Continua Health Alliance has, however, far outgrown its initial members and now bolsters an impressive number of approximately 240 members, which are listed on Continua's website.

By the definition above, one can see that the Continua Health Alliance seeks to achieve the so-called interoperability between different elements of personal health solutions. Rather than develop new standards, their focus is to agree upon already established standards and in what situations they should be used (devices, sensors, application servers). Thus, they define design guidelines that enable vendors to build interoperable sensors, home networks, telehealth platforms and health wellness services (6). The Continua Health Alliance's Design Guidelines is the document that references to the standards and specifications that Continua selected for ensuring interoperability of devices (7). They were specifically written for device manufacturers that intend to go through the Continua Certification process with their devices, companies that integrate Continua devices in their solutions (which would be the case of Critical Health) and test labs that certify compliance to Continua specifications (7).

Continua Health Alliance’s efforts focus primarily on three areas:

- **Health & Wellness –** The focus here is on devices that maximize the effectiveness of fitness programs, perform training progress tracking and allow sharing of workout results. By collecting information about an individual’s fitness, it is possible to improve the results of workout programs and contribute to keep fitness routines fun, interesting and engaging. By promoting and facilitating exercise among adults, the Continua Health Alliance believes that is possible to make people stay healthier for longer periods of time, thus reducing healthcare costs on the long run (8) (9).
- **Disease Management** – According to Continua, 860 million people around the world have at least one chronic disease and the number is rising fast (9). Interoperability amongst health products allows for a better management of chronic conditions at home or at work. Integration between devices and health monitoring systems allow patients, specialized care teams or family members to better keep track of disease’s condition and to intervene as necessary (10).
- **Independent Aging** – Elderly monitoring can improve the quality of life of the aging population. By providing interoperability between different kinds of sensor devices, it is possible to reduce some of the burden in elderly care facilities or improve quality of care at home. Products with the Continua logo collect information about an elderly

person's wellbeing on a daily basis and communicate it to family members and care teams (9). This is the area which holds the most interest to Wisedome.

With the purpose of building trust among costumers and entities, each product within the Continua Health Alliance has to pass a product certification program. Only the products that pass this certification process and present the Continua Certified logo are considered interoperable with other certified products. Certification allows Continua products to become "future proof" as long as the new versions of the design guidelines ensure retro-compatibility. Certification includes two processes: conformance testing towards the Continua standards/specifications and interoperability testing (11).

Regarding wireless device connectivity, the Continua Health Alliance endorses two wireless communication technologies: ZigBee and Bluetooth.

ZigBee technology offers a "build reliable, cost effective, low-power" wireless solution for use in residential, commercial and industrial applications (12). On June 8, 2009, the Continua Health Alliance chose ZigBee as the wireless technology of choice for low power sensors (13).

Bluetooth is a wireless technology standard specialized in exchanging data over short distances (using short length radio waves) between fixed and mobile devices, creating personal area networks with high levels of security (14).

Both these technologies are merely a vessel to transmit information; the way that information is structured in messages is just as important. The ISO/IEEE 11073 family of standards has been adopted by Continua Health Alliance as the protocol to be used in sensor device communication, which guarantees that devices will understand each other (13) (15).

3.1.2 ANT+ Alliance

The ANT+ Alliance defines the ANT+ connectivity solution in its website as:

> *"ANT+ facilitates the collection, automatic transfer and tracking of sensor data for monitoring information anywhere, anytime. The key advantage of this unique managed network is device specific interoperability which enables wireless communication with other ANT+ products. This interoperability function (added to the base ANT protocol) now facilitates the reliable transfer of data between sensors and display devices such as watches, heart rate monitors and bike computers. Applicable in sport, wellness management and home health monitoring, ANT is proven with several million nodes shipped to date. ANT+ provides off –the-shelf interoperability to guarantee seamless digital wireless communication in the 2.4 GHz license-free band (16)."*

The ANT+ Alliance is an industry coalition that endorses the use of ANT Wireless technology as basis for communication and interoperability between devices. Created as a division of the Dynastream Innovations Inc (17), ANT was initially target at the sports sector, particularly in the cycling and fitness areas, and was adopted by many of the industries' significant players such as Garmin, Nike, Suunto and Tacx (17) (18). Since then, ANT+ initially success propelled the group to other areas such as the wellness and home health monitoring markets. By the time of this thesis writing, ANT+ has reached approximately 250 members that cover the sports, wellness and health monitoring areas.

ANT™
the power of less

Figure 6 – The ANT logo.

It is ANT+ Alliance's objective to standardize communication between a wide variety of sports, wellness and health monitoring devices in contrast with the proprietary solutions imposed by many of the manufacturers in these industries (16). Standardization allows interoperability between devices produced by many companies within the ANT+ Alliance, which can then be integrated in health monitoring solutions such as Wisedome. Standardization is achieved by firstly, using a unique wireless technology on all devices, the ANT wireless technology and secondly thorough the ANT Device Profiles. The ANT Device Profiles define the data packet format, in other words, how messages are structured and shared between devices. The Device profiles also define the appropriate wireless channel parameters, thus ensuring interoperability between different devices (19).

After joining the alliance, a company is then capable of acquiring the development tools necessary to ensure that its device or system is compatible with other ANT+ devices and then perform the ANT+ Self Compliance Test Process. The device can then bear ANT+ logo in its packaging and labeling and becomes an official ANT+ Alliance product (19).

Unlike the Continua Health Alliance, the creation of the ANT+ Alliance was not driven by the intent of defining communication standards and technologies, but to promote the use of the ANT Wireless technology instead. The ANT wireless technology focuses on the 2.4Ghz wireless band and is designed for applications that require (20):

- **Ultra low power** – ANT wireless communication has very low energy consumption. An ultra low power device has energy consumption in the range of the milli-watt or the microwatt (21).
- **High resource optimization** – ANT wireless messages are optimized to have as little overhead as possible (21) and the protocol fits into a compact sized memory (20).
- **Network flexibility and scalability –** ANT wireless protocol allows for multiple network topologies including mesh networks (22).
- **Easy to use with low system cost -** Operates independently with a single chip (20).

ANT Wireless technology was engineered for being a reliable, ultra-low power wireless solution.

3.2 Alliances and Technology Comparison

In order to compare the industry coalitions and their supported technologies, it was decided that a comparison should be made that would put describe how each Alliance and their supported technologies fare on the following aspects:

- **Wireless Specification** – In what wireless specification do the supported technologies base themselves on.
- **Energy Consumption** – A qualitative analysis of the technology's energy consumption levels.
- **Network Types** – A comparison on what network topologies are supported.
- **Interoperability Standards** – How are messages structured and how interoperability is ensured between alliance certified devices?
- **Health Monitoring Devices** – A qualitative analysis on the number of health monitoring devices certified in each alliance.

The following section will firstly present the research performed and compare Bluetooth, ZigBee and ANT wireless technologies on each of these areas. Finally, it will culminate on the presentation of the comparison matrix.

3.2.1 Wireless Specification

The Wireless Specification section describes the basis wireless specification used by each of the technologies. The base specification determines most of the wireless technology features and capabilities.

3.2.1.1 ZigBee - IEEE 802.15.4-2006

ZigBee is built upon the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4 standard (23). The ZigBee group wanted to create a low data rate, low power consumption, low cost wireless networking protocol while the IEEE 802.15.4 committee began to work on a low data rate standard shortly after. The ZigBee Alliance and the IEEE group joined forces and ZigBee became the commercial name for the technology (24).

The 802.15.4 standard serves as basis for other wireless technologies, such as 6LoWPAN (25) and MiWi (26). ZigBee draws many of its features from its underlying 802.15.4 standard: the emphasis on low power consumption, low cost, self-organizing network arrangement, network path adaptation, co-existence assurance with other IEEE 802 wireless technologies, secure communication and low data rate. The standard has been designed with applications which cannot handle the power consumption of heavier protocol stacks in mind (24).

The ZigBee technology can be seen as a stack composed of four layers: the top high-level layers are defined in the ZigBee specification while the bottom low-level layers are defined in the IEEE 802.15.4-2006 standard, as shown in Figure 7. Thus, the ZigBee protocol builds upon and acts as an intermediate between the low level 802.15.4 MAC and PHY sub layers and the software that makes use of wireless communication. It defines the network layer specifications and provides functionality for application programming (27).

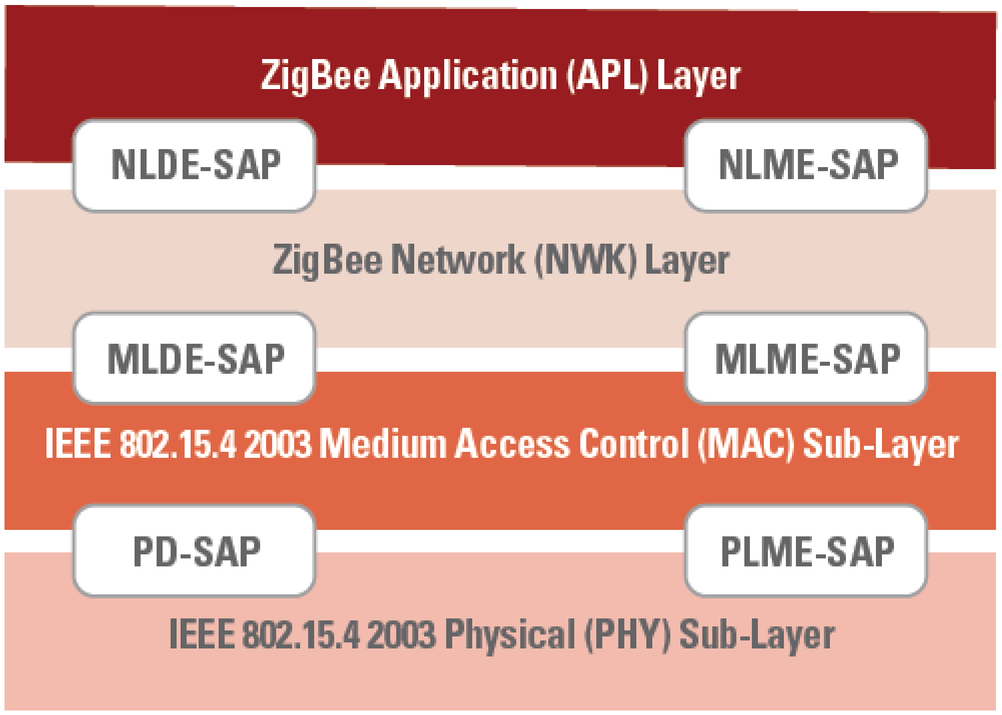

ZigBee Application (APL) Layer
NLDE-SAP
NLME-SAP
ZigBee Network (NWK) Layer
MLDE-SAP
MLME-SAP
IEEE 802.15.4 2003 Medium Access Control (MAC) Sub-Layer
PD-SAP
PLME-SAP
IEEE 802.15.4 2003 Physical (PHY) Sub-Layer

Figure 7 - The ZigBee Stack (23).

The 802.15.4 standard supports several wireless frequency bands. It is capable of achieving speeds of 20 kbps on the 868 MHz band, 40 kbps on the 915 MHz band and 250 kbps on the 2.4 GHz band (27). The maximum communication range between two ZigBee devices is approximately 100 meters (28).

3.2.1.2 Bluetooth - IEEE 802.15.1-2002/2005

Initially, the Bluetooth protocol stack was created by the IEEE group as the IEEE 802.15.1. The original version of this standard was based on portions of the Bluetooth v1.1 Specification. An updated version of the Bluetooth standard, Bluetooth v1.2, was released as IEEE 802.15.1-2005 (29). However, after the release of v1.2, the IEEE Study Group 1b discontinued their relationship with Bluetooth SIG. Thus, later versions of Bluetooth wouldn't become IEEE standards (30).

The 802.15.1 specification was originally designed by Ericsson (31) as a cable replacement, an alternative to RS232 data cables (32). Thus, the basis technology has little consideration over power saving. One particularity of Bluetooth is it's usage of various wireless frequencies. Bluetooth divides the data into several packets and transmits them over 79 frequencies in a pseudo-random pattern. Bluetooth radios "hop" to a new frequency after transmitting or receiving a packet, following the pseudo-random , known to all the devices in a single Bluetooth connection (33). The Bluetooth technology can achieve speeds up to approximately 3 mbps by using exclusively the Bluetooth protocol as of Bluetooth 2.0, but since Bluetooth 3.0 the speed has been increased up to 24 mbps, by also using the 802.11 radio protocol, the basis for Wi-Fi technology (34). The maximum expected range between two Bluetooth devices is approximately 10 meters (28).

3.2.1.3 ANT Wireless

Ant Wireless uses its own proprietary physical layer and it is not based on any other protocols. The ANT protocol stack provides "reliable data communications, flexible and adaptive network operation and cross-talk immunity" (35). The stack is compact and requires minimal microcontroller resources.

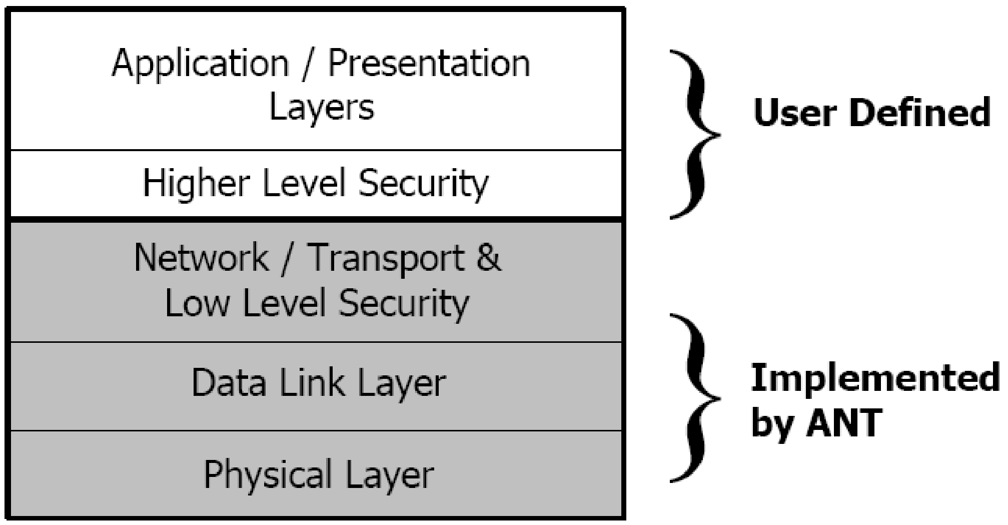


Figure 8 – OSI Layer model of ANT protocol stack (35).

The ANT protocol ensures enough flexibility in terms of user control, allowing, for instance, user-defined network security implementations. However, it also lightens computational burden by abstracting most of the low level functions from the application developer. The interface between ANT and the Host application has been designed with the intent of being as simple as possible, thus allowing for an easy and quick implementation of the ANT protocol in communication boards to be used by applications like Wisedome or sensor device processor chips (35). Aside from being simple to implement, the protocol is also lightweight, has a low handshaking overhead, and provides ultra-low power consumption and low latency (21) (22). It boasts an over the air data rate of 1 mbps for low duty cycle operations (22) and a maximum expected range between nodes of about 30 meters (28).

3.2.2 Energy Consumption

Energy consumption of the wireless communication protocol is one of the most important issues when dealing with low power health monitoring sensor devices. The lower the energy used for data transmission, the longer the device's battery life becomes. Longer battery life means less device charging. The less time nurses or TAPs have to spend charging and replacing devices, the more time they have to focus on patient care, thus improving customer satisfaction.

3.2.2.1 ZigBee Energy Consumption

One of the flagship features of the ZigBee technology is its design for “low-power” applications. The ZigBee has been designed to deliver wireless networking to even the most basic devices, including those that run on batteries lasting for years. ZigBee implements various mechanisms in order to reduce overall power consumption.

ZigBee’s low power consumption is associated not with the RF power of ZigBee’s nodes, but with a sleep mode, specifically designed to extend a device’s battery life (36). Devices can remain asleep most of the time, thus saving battery life. ZigBee radios can switch automatically to sleep mode whenever the device is not transmitting thus achieving very low duty cycles. When a radio is sleeping, the RF power rating is irrelevant, only when transmitting does the radio consume significant amounts of energy (36). Also, the sensor’s processor itself usually consumes very little energy; it’s the radio that consumes most of the power. Thus, with very low duty cycles, ZigBee solutions can effectively achieve a very low average power usage. In order to reduce the amount of time the radio is on, the data is sent through small-duration but high-rate bursts, which is an effective way of saving power (37) (38). Since the active periods of the ZigBee radios can be very short, significant power can be lost if the transceiver warm-up time is long. Fortunately, the ZigBee protocol uses various techniques to lower this warm-up time. The wideband techniques used by ZigBee, such as the Direct Sequence Spread Spectrum, have the advantage of their wide channel filters having inherently short settling times (39).

From these efforts, the ZigBee alliance claims that battery lifetimes from a few months up to several years can be expected (40). ZigBee Alliance’s confidence in their technologie’s low energy consumption is evident in their certification process: individual devices must have a battery life of at least two years to pass ZigBee certification (41).

However, according to the studies performed by Chaitanya S. Misal presented in his thesis, energy consumption on ZigBee is heavily dependent on the network configuration. Device battery life depends on factors such as the size of the packets transmitted or received over the air, the node being a recipient or a transmitter of data (recipients are expected to last approximately 63% more than transmitters) or if the network uses transmission retries when messages are not delivered (retries can bring down battery life by almost 45%) (42).

3.2.2.2 Bluetooth Energy Consumption

Up until very recently, the Bluetooth protocol did not have the same kind of low power consumption features presented by its competitors. As previously mentioned in page 19, Bluetooth was initially developed as a cable replacement technology, thus, not having much concern about power consumption. According to (43), small low-powered devices cannot bear the power consumption and cost associated to Bluetooth. This lead Nokia to create an alternate Bluetooth-based protocol named Wibree, which would work alongside Bluetooth while effectively circumventing Bluetooth's high energy costs (44). In June 2007 it was decided that Wibree would become an integrate part of the Bluetooth specification (45) and was recently released as *Bluetooth Low Energy*, a part of the new Bluetooth 4.0 specification (46). According to Bluetooth SIG director Michael Foley, the new specification reduces energy consumption by nearly 90%. He also mentions that this energy reduction enables the use of Bluetooth more effectively in many areas, one of them being health monitoring sensors (47). Unfortunately, chip manufacturers have yet to disclose power consumption data on data sheets, thus it is difficult to truly ascertain how much power the new Bluetooth protocol consumes (48). Devices that support both Low Energy and Classic *Bluetooth* technology via a dual-mode radio feature will be compatible with Classic *Bluetooth* devices. However, single-mode *Bluetooth* Low Energy devices will not interoperate with existing *Bluetooth* devices (49). Despite consuming less energy, the Low Energy specification has an over the air data rate lower than Classic Bluetooth, transmitting at 1 Mbps. Just like ZigBee, Bluetooth low energy applications are able to achieve ultra low duty cycles. Connection setup time was also greatly reduced in this new version of the protocol, with connections being established, and the data transferred in an amount of time as low as 3 ms. This can be particularly useful in the health monitoring area, as the time it takes to detect an alarm event and send the alarm must be as short as possible (50). The new Bluetooth 4.0 specification is still very recent, but devices using it are expected to appear by the end of 2010 (49).

3.2.2.3 ANT Energy Consumption

ANT Wireless distinguishes itself from its competitors by declaring itself as an "Ultra Low-Power" wireless solution.

"Ultra Low Power is an electronic gadget that has milli-watt or microwatt power consumption." (21).

They believe that they are the most energy-efficient wireless technology currently on the market, claiming to be able to operate for up to several years on coin cell batteries (51).

Figure 9 shows ANT's graphical comparison regarding energy consumption, range and data rate for several wireless technologies on the market while Figure 10 shows their expectations regarding the new Bluetooth Low Energy (the specification hadn't been released at the time these comparisons were presented).

ANT believes that ultra low power wireless can be achieved through efficiency, simplicity and scalability:

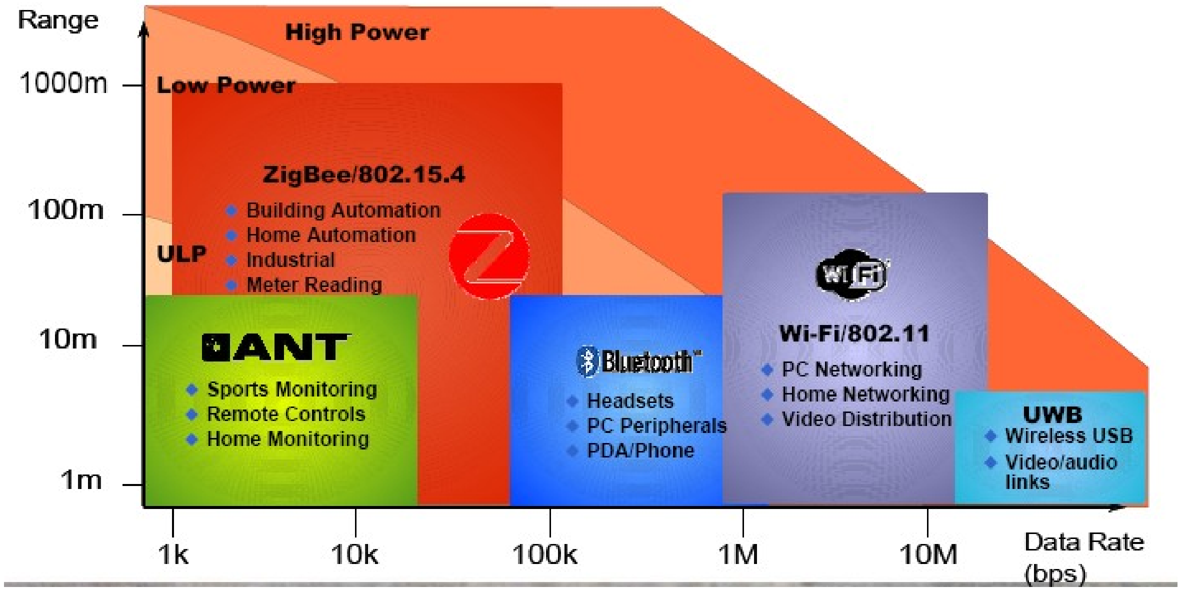


Figure 9 – ANT's comparison of the energy consumption between some of the current wireless technologies **(21)**.

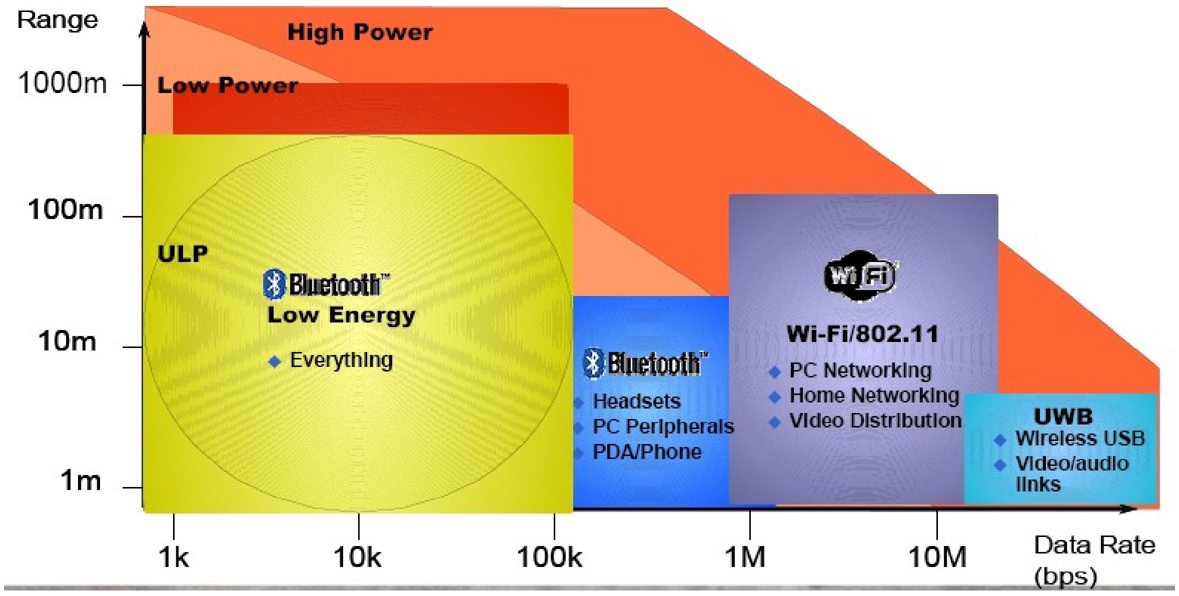


Figure 10 - ANT's expectation of the energy consumption of the new Bluetooth Low Energy protocol **(21)**.

Efficiency consists of communicating for a short period of time in order to save battery power. ANT sought to create a lightweight protocol with as little message overhead as possible: according to them, Bluetooth, Bluetooth Low Energy and ZigBee all have about twice as much message overhead as ANT. ANT's protocol stack size is also much smaller than its competitors (21).

Simplicity consists on creating a clear message protocol that requires as little handshaking or negotiation as possible. ANT decided to completely bypass handshaking, so unlike Bluetooth and ZigBee, ANT does not require handshaking at all (21).

Scalability allows developers to create complex network configurations with simple ones as a base. ANT is capable of scaling from complex inter-node communication methods, like multi-frequency bidirectional communication, to simple methods such as single frequency single direction communication (21).

Brian Macdonald, Director of ANT Networks, believes that an Ultra Low Power wireless solution fares much better in comparison with other wireless protocol solutions. In an article written for Nikkei Electronics Asia in December 2007, he gives an example of a sensor transmitting 8 bytes of data, 24 hours a day with a message period of 0.5Hz. For this use case, a technology that supports Ultra Low Power consumption such as ANT presents a battery life of 7.2 months for the transmitter sensor and 6.3 months for the receiver sensor. According to him, this solution has a much better battery performance than a ZigBee based solution which presents a battery life of only 8 to 10 weeks for both the transmitter and the receiver (52).

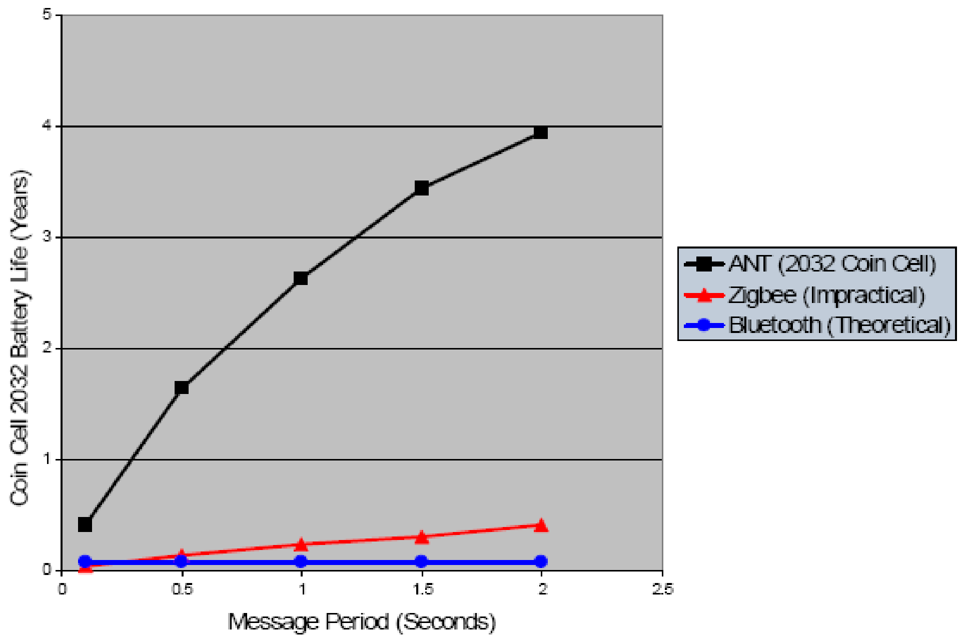


Figure 11 – ANT's comparison of the duration of a coin cell 2032 battery using different wireless technologies **(52)**.

Figure 11 shows a comparison of the battery life between ANT, ZigBee and Bluetooth using a Coin Cell 2032 battery. Although interesting, it should be noted that this comparison is not realistic. Peak current requirement for ZigBee and Bluetooth exceeds coin cell battery capability so coin cell operation is impractical. Since both ZigBee and Bluetooth devices are not capable of using 2032 Coin Cell batteries, the data presented in this graph is only theoretical (28).

3.2.3 Supported Network Topologies

Different network topologies fulfill different kinds of needs. The more types of network topologies a technology supports, the more flexible that technology is. Both ZigBee and ANT are capable of implementing mesh network types. The concept of a Mesh network is powerful and consists on an environment where a given device is able to communicate with any other device on the network. Mesh networks are "self-healing" because the network is able to route traffic by using different device paths: if a device that has been routing traffic between two other devices on the network suddenly starts malfunctioning, the network should be able to use other devices to create an alternate path and keep the communication going. This section presents the topologies supported by ZigBee, Bluetooth and ANT.

3.2.3.1 ZigBee Supported Network Topologies

ZigBee supports various types of network topologies (mesh networks included) with a maximum of 2^{64} nodes per network (28). There are two types of devices on a ZigBee network (27) (53):

- **Reduced Function Devices** (RFD) – is the simplest type of device, only applied as an "end-point" in the network configuration. They have just enough functionality to relay traffic to their parent nodes (a single FFD) and are unable to route traffic coming from other devices.
- **Full Function Devices** (FFD) – these devices present full functionality and perform the most complex network tasks. They are capable of full communication and to relay traffic between other devices

These two types of devices are, in turn, capable of assuming three different roles on a ZigBee network (27) (53):

- **ZigBee PAN Coordinator** – These devices assume role functions and manage the whole network. They are the most functional device and are used to start the network and provide the bridge to other networks. This function is always assumed by FFDs.

- **ZigBee Router** – The ZigBee routers function is to route information between different nodes in the network, serving as “communication bridges”.
- **ZigBee End-Device** – These are the end-nodes on the network, usually the receivers or the senders of network commands. Usually these types of nodes are part of sensors that gather information or devices that perform some function that gather information or receive commands. Each one of these is usually associated with a single ZigBee Router. They are usually RFD, although they can also be FFD acting as simple devices.

Using these types of devices, ZigBee is capable of supporting Star, Cluster and Mesh network topologies.

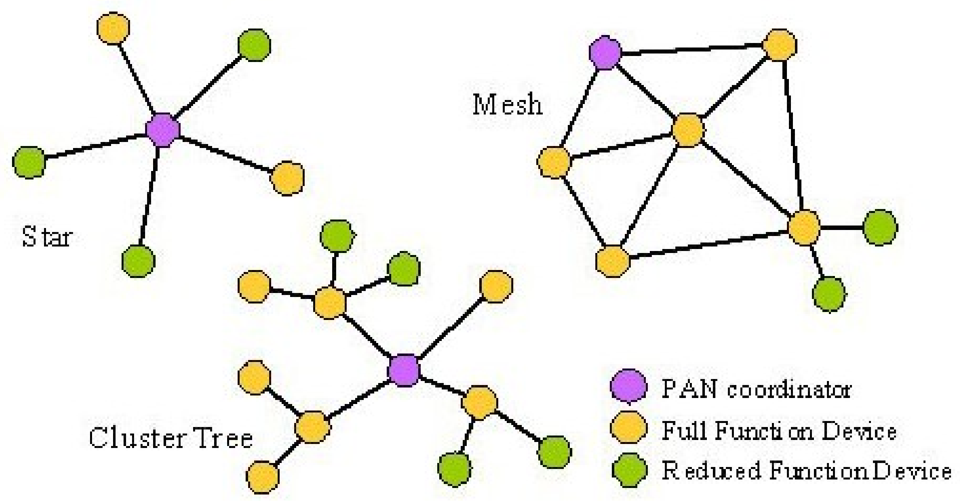


Figure 12 – Network topologies supported by ZigBee **(24)**.

In the Star topology, the PAN coordinator establishes communication between the devices. The PAN coordinator’s tasks are demanding, so it is usually plugged onto a power source while the other nodes are battery powered. Whenever a FFD device is activated, it may establish a new start network and become the coordinator. By choosing a unique PAN identifier, each star network may operate independently. The end-point devices may only communicate with the network’s coordinator (24).

Mesh networks also require a PAN coordinator. Unlike the star topology, a device can communicate with any other device, with messages being routed through the ZigBee routers (24). One interesting feature about ZigBee’s mesh networks is the network’s self-healing properties. As Figure 13 shows, the network is capable of generating alternate paths in case of device failure or external interference (23). Although mesh networks result in increased robustness and flexibility, those advantages come with a price. In order to guarantee multi-path routing, the ZigBee routers must increase their radio's operating time so that they are prepared to receive possibly incoming traffic from the network. Mesh networking requires more complex network protocols and has a higher communications overhead. These factors lead to an increase in overall power consumption and costs (53) (54).

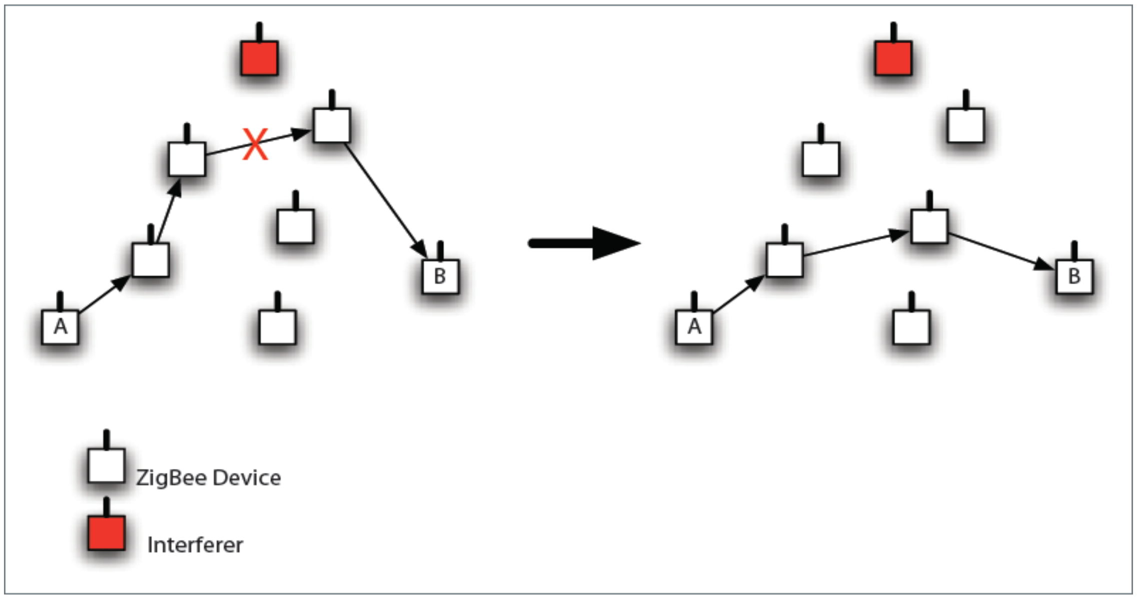


Figure 13 – ZigBee is capable of multipath routing (23).

The Cluster-tree network is a special case of a mesh network. Most of the devices in the network are FFDs and RFDs usually connect only as ends of a branch. The network is composed of various "clusters", with each cluster having a cluster head (CLH). The PAN coordinator forms the first cluster by establishing itself as the CLH and then broadcasting "beacon" messages to neighboring devices. Devices that receive these beacons can request the CLH to join the network. In case the PAN coordinator permits the device to join, it will add this new device as a "child-node" on its list of neighboring nodes. The child node will add the CHL as its "parent-node" and then begins to transmit new periodic beacons so that other devices may then join the network at that device. As soon as application or network requirements are met, the PAN coordinator may instruct one of its child nodes to become the

CLH of a new cluster adjacent to the first one. This topology presents increased coverage area at the cost of increased message latency (24).

3.2.3.2 Bluetooth Supported Network Topologies

Bluetooth wireless links are always established within a piconet. Each piconet has one master node and at least one slave node and can have at most 8 active members. Slave nodes do not directly communicate with each other, but instead rely on the master as a transit node (55). However, the terms "master" and "slave" only apply to a given piconet: a device can assume both roles if it belongs to more than one piconet. A Bluetooth enabled device can only be the master of a single piconet, but it may assume the role of slave for several independent piconets (56). By belonging to several piconets, a Bluetooth device can act as a bridge between them and when two or more piconets are connected, we are in the presence of a scatternet.

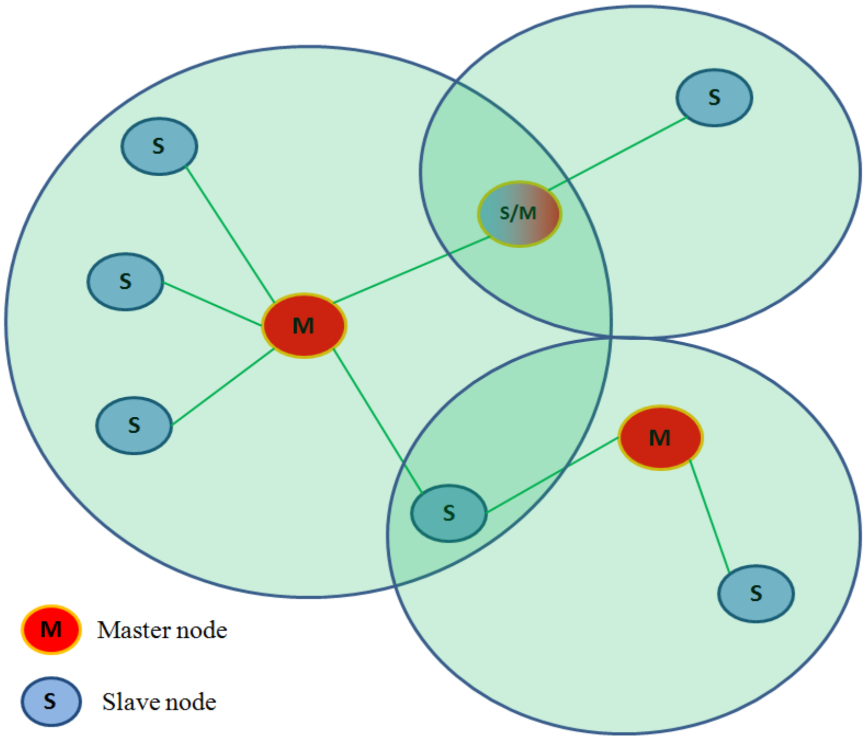


Figure 14 – Example of a Bluetooth scatternet composed of 3 piconets.

The scatternets are not covered by current Bluetooth core protocols. The *Bluetooth* core protocols do not, and are not intended to offer such functionality, thus, it has to be implemented by higher level protocols (56).

3.2.3.3 ANT Supported Wireless Topologies

ANT supports all the wireless topologies supported by ZigBee, but with a different mechanism and with a maximum of 2^{32} nodes (28). In ANT, devices do not have different functions on the network unlike ZigBee's several types of devices (Coordinator, Router and End-Point). When setting up a ZigBee network, one of the limitations is that it isn't easy to extend the network on an ad hoc basis. This is because it is difficult for nodes to casually join or leave the network unless they are of the right type (for instance, end-point devices can only connect to one FFD). In ANT's case, all nodes are identical and capable of acting as "slaves" and "masters", swapping roles whenever necessary. All the nodes can act as transmitters, receivers or transceivers to route traffic and can leave or join the network in an ad hoc fashion. Another interesting feature is that every node is capable of determining the best time to transmit based on the activity of its neighbor nodes, thus, coordinators are not necessary. ANT claims that the technology is capable of supporting this type of network with tens or hundreds of nodes while keeping system resource overhead low (53).

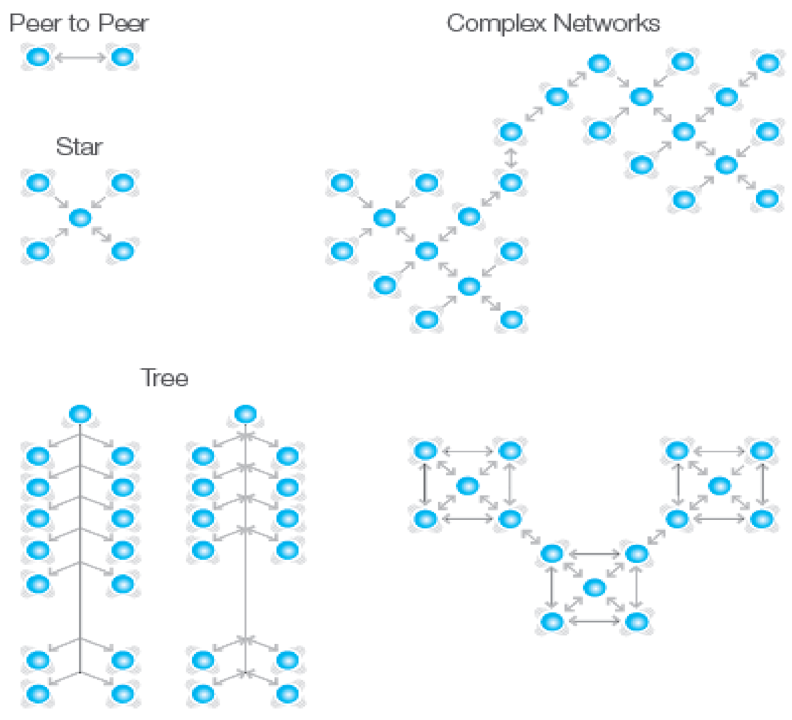


Figure 15 – ANT supported network types: peer to peer, star, tree and complex networks (mesh) **(89)**.

Just like ZigBee, ANT supports star, tree and mesh network topologies. The tree topology is similar to the cluster-tree topology, in that it is like having several star topologies connected to each other, much like the clusters were connected in the cluster-tree topology. Mesh network is supported but, just like ZigBee, using this type of topology introduces complexity and induces in a bigger power consumption and increased system resources. In most cases, problems can be resolved using peer-to-peer, star or tree networks (22).

3.2.4 Interoperability Standards

It is not just enough that devices use the same means to communicate (same wireless technology), they must also speak the same "language" and that's where interoperability standards come in. Interoperability standards make devices able to understand each other by following the same rules for message exchange. The standards define the data types, message payload, communication handshaking, connection states and other overall communication characteristics. This section will present the interoperability standards adopted by each industry coalition and will also include a brief description of how they work and what they offer.

3.2.4.1 IEEE 11073-20601™ - Optimized exchange protocol – PHD

Continua Health Alliance supports the ISO/IEEE 11073 as the standard message protocol for sensor devices. The IEEE 11073 (also known as X73) is a family of medical device communication standards that was defined by the Institute of Electrical and Electronics Engineers in 1982 in order to seek interoperability, with the objective of defining a common way for medical equipment to communicate. The 11073 family of standards has existed for several years, but it used to be limited to intensive care unit devices or medical equipment that is plugged to a continuous power source. The protocol had been criticized for being excessively heavyweight and complex. Thus, Malcolm Clarke et al. (57) proposed an adaptation of the former 11073 standard to fit the demands of personal health data (PHD) devices, such as wireless health monitoring sensors. This resulted in the separation of the IEEE 11073 family of standards into two main branches (58):

- At the hospital level stands the former family of standards, the IEEE 11073-00101 PoC-MDC (Point of Care Medical Device Communication)
- For personal telehealth systems there is the more recent IEEE 11073-20601-2008 PHD (Personal Health Device) family of standards.

The IEEE 11073-20601-2008 PHD is basically a simplified, “lighter” version of the former 11073 POC protocol. Its main differences reside within the OSI layer 7, the communication layer. PHD provides an “Optimized exchange protocol” that offers the same functionally as OSI layer 7, but implemented in a lightweight fashion. The exchange protocol is optimized for low capability devices that have limited energy resources and processing power for communication. The existing nomenclature of the 11073 POC was used but was simplified in the PHD version by constraining the scope of the model to the personal health devices (58). In 11073 PHD, sensor devices are known as “Agents” while the applications that use them are known as “Managers” (59).

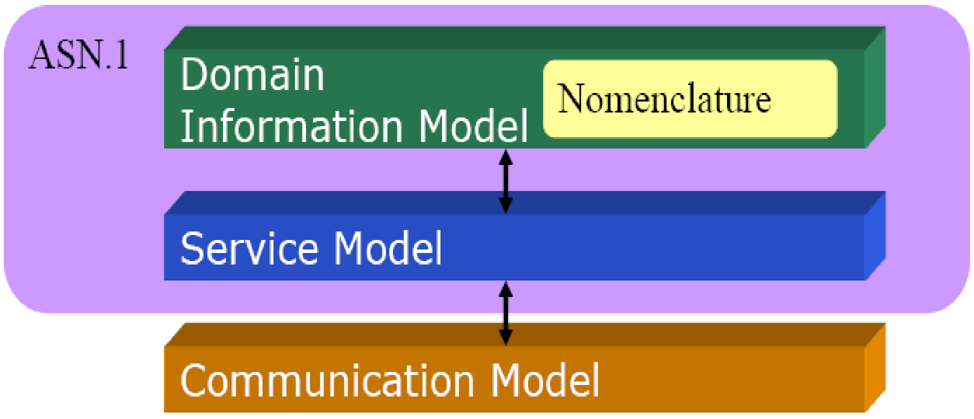


Figure 16 – IEEE Std 11073-20601™ model constitution **(60)**.

As show in Figure 16, the 11073 PHD is composed of three modules, two of which are defined using Abstract Syntax Notation One (ASN.1) language used for the abstract modeling of data and interactions. The Domain Information Model (DIM) describes the sensor device and physiological data. To do this, it uses an object - attribute approach: an “object” is tailored by a series of “attributes”. Objects are created from Classes which are a model that defines the attributes and methods for that object type. Objects can represent sensor devices (Agents), measurement metrics or specific data types and each possess a series of attributes that tailor it. For instance, an object representing a sensor device may possess attributes that define the device manufacturer, the model, the device ID or the battery level while an object representing a weight measurement might include attributes that define a timestamp, the measurement value, the units of the measurement (kg, lb, etc.) among others (59).

The Service Model part of the 11073 PHD defines interactions between devices and data. It works by defining a series of services that are offered by the sensor devices. These services include the Event Reporting Service, Object Access Service, Association Service and Conversion Service. The Event Reporting Service allows Agents to send events, such as updates on a measurement value or changes in device status. It is also used to send Agent configuration during the association process, in which the Agent sends to the Manager what attributes and message formats it supports. Event report message format can be pre-defined in order to omit message headers (which indicate what kind of information is being transmitted), thus optimizing the protocol. The Object Access Service allows the Manager to get and set all kinds of information regarding an object and its attributes. The Manager can, for example, get information on the Agents manufacturer or set the Agents date and time. The Association Service provides various services that allow Agents to “associate” with Managers in order to establish a connection and pass on information. Finally, the Conversion Service allows the conversion of data to various formats (59).

The Communication model describes communication characteristics, connection states and legal interactions on each state. It supports two types of communication: Reliable and ”Best Effort”. The Reliable communication guarantees that the data is delivered in order, free of detectable errors, not duplicated and not missing. However, this kind of reliability comes at a cost of delays and increased power consumption due to retries. In “Best Effort” communication, data may be discarded or lost in its entirety, delivered disordered or duplicated. The communication model also describes how Agents should connect to Managers. This includes defining connection states (disconnected, associated and disassociated) and what tasks should be performed in order to successfully associate an agent to a manager and what kind of interactions can exist in each state. The protocol defines various workflows explaining the steps required for each connection (59).

In order to decrease communication overhead and to simplify the association process, the 11073 PHD families of standards define a series of “Device Specializations”. As previously mentioned, Agents must send their configuration (what kind of attributes they possess and message formats) to the Managers during the association process. These specializations provide standard configurations that are known *a priori* by both Agents and Managers. The Agent merely sends particular Dev-Configuration-Ids that correspond to these specifications and Managers automatically know what kind of Agents they are dealing with. This increases the likelihood of interoperability, by defining specific objects, attributes, ids and services. It also optimizes the configuration phase, since it is not necessary for the Agent to send its full configuration to the Manager. These device specializations focus on describing particular usages of the 11073 PHD. Some examples are the IEEE 11073-10404™ - Pulse Oximeter and the IEEE 11073-10415™ - Weighing Scale (59).

The IEEE 11073-20601™ Application profile - Optimized exchange protocol is **transport independent**. The Continua Health Alliance supports its implementation in different communication technologies, like Bluetooth, ZigBee or USB (15) (60).

3.2.4.2 ANT Device Profiles

ANT+ seeks to ensure interoperability at the message level through “Device Profiles”. Device profiles are keystone of interoperability in the ANT+ Alliance. They define uses cases for a type of sensor, and considering the use case, they also define topology requirements, pairing requirements and data requirements for the sensor. They also define how data should be encoded and decoded for transmission between sensors, channel parameters and the network key to be used. Additionally, implementation code examples on how to perform channel configuration and node pairing are also included to facilitate development of the ANT solution (61).

ANT Device Profiles are not a static documentation; they are designed to respond to the needs of the ANT+ Alliance members. In case the current device profiles are not enough to cover the needs of an application, it is possible for ANT+ members to work with ANT in order to create new device profiles. The device profile is developed by the ANT+ company member and reviewed with ANT+ team on an iterative process until an agreement is reached. When both parties agree on a device profile proposal, a new device profile is created and released for other members of the alliance to use. The ANT+ Alliance ensures its members that backwards compatibility will always be assured in new versions of the device profiles, so that legacy devices won't become obsolete (61).

On an email exchanged with Dallin Doney, an ANT+ Business Development Manager, it was mentioned that ANT device profiles are designed in order to allow the collection devices to map the data from the optimized ANT+ format into possible other XHR formats, with specific reference to IEEE 11073-20601 compatibility. This email can be consulted in Annex B.

3.2.5 Health Monitoring Devices

An important part of this research was to estimate how many health monitoring devices are certified by both Alliances. For Critical Health, it wouldn't be interesting to join an Alliance which didn't certify a considerable number of health monitoring devices that could be used in the Wisedome system. Thus, it was necessary to know the amount and variety of devices certified by each Alliance, before making a decision.

3.2.5.1 Continua Health Alliance Health Monitoring Devices

The number of health monitoring sensors currently certified by the Continua Health Alliance is still poor, but has been growing. The Continua Health Alliance keeps a record of every product currently certified on their website. At the time of this document's writing, there are 14 products certified by the Continua Health Alliance, from which only 6 can be considered wireless health monitoring sensors. There are currently no certified devices using ZigBee technology, and all of the devices use classic Bluetooth technology. The currently Continua Certified wireless health monitoring devices are (62):

- **Nonin Onyx® II 9560 Wireless Fingertip Pulse Oximeter** – This pulse oximeter uses classic Bluetooth technology to allow clinicians to remotely monitor patients with chronic diseases.

- **A&D Medical UA-767PBT-C Blood Pressure Monitor** – This Blood Pressure monitor uses classic Bluetooth and is capable of sending data either in real-time or in batch mode.
- **A&D Medical UC-321PBT-C Weight Scale** – This scale is capable of measuring patient weight and sending data to a telemedicine access point through classic Bluetooth.
- **OMRON Bluetooth® Home Blood Pressure Monitor** – The device has the ability to easily transmit data to electronic health applications by using Bluetooth.
- **Bluetooth® Body Composition Monitor HBF-206IT Weighing Scale** – This scale gives a comprehensive view of a patient's health level through several fitness indicators and provides an ability to easily transmit data to electronic health applications by using Bluetooth.
- **Omron Pedometer with Bluetooth Docking Station** - This pedometer along with the Bluetooth® Docking station allows transmission of data to electronic health applications.

The ZigBee Alliance itself (not directly associated with the Continua Health Alliance) does not currently show any certified ZigBee health monitoring devices in their own webpage (63). There are, however, non-certified health monitoring devices using ZigBee technology produced by independent companies. Since these devices are not a part of the Continua Health Alliance's list of certified products, nothing guarantees that they are using IEEE 11073 PHD as a communication standard, making them no different than any other proprietary solution. Despite ZigBee being used sporadically, Bluetooth and Wi-Fi stand as the most popular wireless technologies of choice for independent health monitoring device solutions.

3.2.5.2 ANT+ Alliance Health Montoring Devices

Unlike Continua Health Alliance, the ANT+ Alliance has an extensive list of certified products in its website. However, most of these products belong to the sports and fitness areas, while devices in the health area are fewer. Even so, there exist some products in the health monitoring area and some of the devices from the sports area could theoretically be used with a health monitoring application, particularly bracelets, watches and chest-straps that measure an individual's heart rate.

During the MEDICA International Trade Fair 2009 health tradeshow, some ANT+ members such as Cosmed, HMM, IDT Technology, Beurer, Spantec and A&D Medical launched various new healthcare sensor devices and demonstrated interoperability with ANT (64) (65).

3.3 Wireless Interference Research

In an eventuality that Wisedome could come to use more than one kind of wireless device simultaneously, it would be interesting to have an idea of the interference between different wireless technologies operating on the same environment. This section gives an overview of the mechanisms against interference employed by ZigBee, ANT and Bluetooth as well as an analysis of a few studies that quantify the performance of these wireless technologies when co-existing on a single environment.

3.3.1 ZigBee Interference Avoidance Techniques

ZigBee uses various techniques to avoid wireless interference, such as **Direct Sequence Spread Spectrum** (DSSS)**, Multiple Channels, relatively high Data Rate, CSMA** and **transmission retries.**

The use of DSSS techniques consists on spreading the wireless signal over a broader bandwidth than the information signal actually requires. This is in contrast with narrow-band signals, that when happen to collide with other narrow-band signals result in great amounts of overlapping and information loss. Signals that use a broader bandwidth can co-exist with narrowband signals because in an event of a collision, the receiver sees it as merely a slight reduction in the signal-to-noise ratio over the larger spectrum being used. Thus, by the use of DSSS, ZigBee can coexist with narrow-band signals on the crowded 2.4Ghz band. This specific spreading technique uses a pseudo-random code sequence called "chipping sequence" that is used to modulate the carrier signal and to encode the data being transmitted. Only the radio receptors that are aware of the "chipping sequence" are able to "de-spread" and reconstruct the original signal. This method is also used in 802.11b and 802.11g WLAN technologies (23) (66).

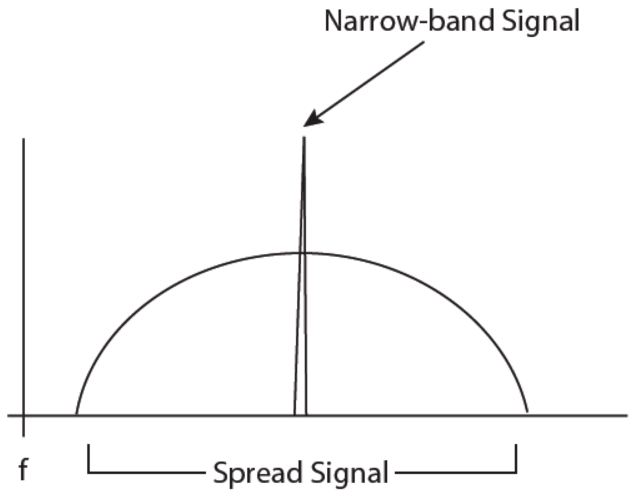


Figure 17 – Collision of a spread Signal with a Narrow Band Signal **(23)**.

The use of multiple frequency channels also allows ZigBee to avoid possible interference. This technique is called frequency division multiple access (FDMA) and in the 802.15.4 standard it divides the 2.4 GHz ISM band into 16 non-overlapping channels, 5 MHz apart from each other. The 802.15.4 standard ensures that devices operating in adjacent channels can coexist comfortably. The wider the space used between channels, the bigger will be the resistance against interference (23). The 802.15.4 standard PHY layer also provides the ability to verify and report whether or not a channel is clear to transmit and measure the interference present in a given channel. This can be used by higher layers in the protocol stack to select the best available channel for operation. ZigBee supports a total of 27 wireless channels over 3 different frequency specters (39):

- 2.4 GHz with 16 channels Bands for global use
- 915 MHz with 10 channels for N. America, Australia and a few additional countries
- 868 MHz with 1 channel for EU countries

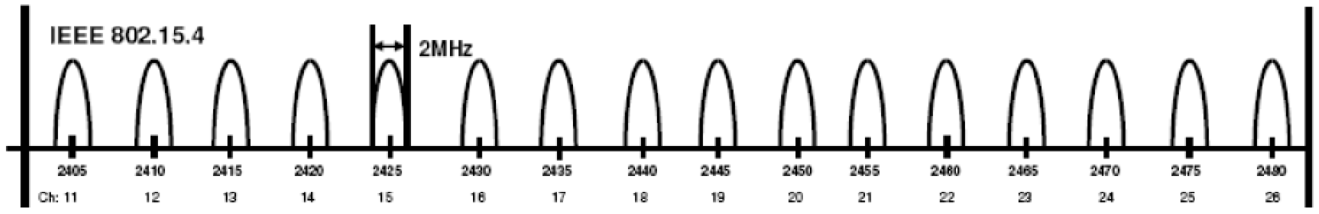


Figure 18 – The 802.15.4 standard 2.4 GHz ISM band channels **(90)**.

Even if ZigBee tries to minimize interference by using multiple channels, it may happen that a device shares the same channel with other ZigBee device. This is particularly important when the 863.3 MHz frequency that only has a single channel is being used. To avoid signal collisions, there has to be some order as to when each device transmits. If a device transmits whenever it wants to, collisions with other devices are prone to arise. IEEE 808.15.4 uses carrier sense multiple access (CSMA) to address this problem. It consists on the strategy of "listening before you talk". The device checks if the channel is busy, and if it is, waits for a random period of time before checking again. When the device finds the channel free, it finally transmits its message (23).

The 802.15.4 designers chosen to use a relatively high data rate of 250 Kbps, even if most of the target applications require a much lower data rate. One of the reasons for this is to reduce channel occupancy and increase coexistence. A radio that transmits its data faster will occupy the wireless channel for shorter amounts of time, thus reducing the probability of collision and allowing others to use the channel more often (23).

Finally, ZigBee also employs a retransmissions system, in case a message that was sent is not successfully received. All communication requires acknowledgement responses in 802.15.4: a device that receives a message has a small time interval to send back to the sending device a short acknowledgment response that confirms the reception of the message. If the sending device does not receive an acknowledgement within the specified time window, it assumes that the message was not delivered and sends it again. The transmitting device keeps re-transmitting the message until it either receives an acknowledgement response or, after a few tries, gives up and reports a failure (23).

3.3.2 Bluetooth Interference Avoidance Techniques

In order to reduce interference and to increase security, Bluetooth uses **Adaptive Frequency Hopping (AFH)** which is a type of Frequency-hopping spread spectrum (FHSS). The FHSS is a spread spectrum modulation technique that uses a large bandwidth (much like ZigBee), divides the data into packets and transmits it over numerous "hop frequencies" with 1MHz of difference between them, in a pseudo-random pattern (Bluetooth uses 79 hop frequencies). This pseudo-random pattern is derived from the clock of the piconet's master node, and each slave must remain synchronized with it (33). The pseudo-random pattern is made known to both the master and the slaves and is used to map the sequence of frequencies that will be used. Bluetooth radio modules after transmitting or receiving a packet follow the sequence and "hop" to a new frequency, performing this task 1600 times per second (67).

3.3.3 ANT Interference Avoidance Techniques

ANT operates in the **2.4 GHz ISM frequency band** which is prone to interference due to its status as the global license-free band and the wide array of network solutions that use it. ANT addresses the interference issues by using an adaptive isochronous scheme that "effectively eliminates the vast majority of interference issues (22)."

ANT divides the 2.4GHz band on several 1 MHz channels and each channel in several timeslots. The adaptive isochronous scheme relies on each node transmitting on a clear timeslot, within a defined 1 Mhz channel. Each radio only transmits for a very short period of time (less than 150 μs per message) thus allowing a single frequency channel to accommodate hundreds of timeslots. Although the system uses a timeslot mechanism, it does not require synchronization with a master clock. Each node transmits at regular intervals, but is able to change its transmission scheme if interference from a neighbor node is detected for a particular timeslot. Besides having specific timeslots for each node, ANT provides yet an additional level of co-existence: in case a particular frequency is suffering from too much interference due, for example, to the existence of another 2.4 GHz wireless network (such as

Bluetooth, Wi-Fi, ZigBee or even another ANT network), ANT is capable of hopping to a different 1MHz frequency channel, thus avoiding the crowded frequency.

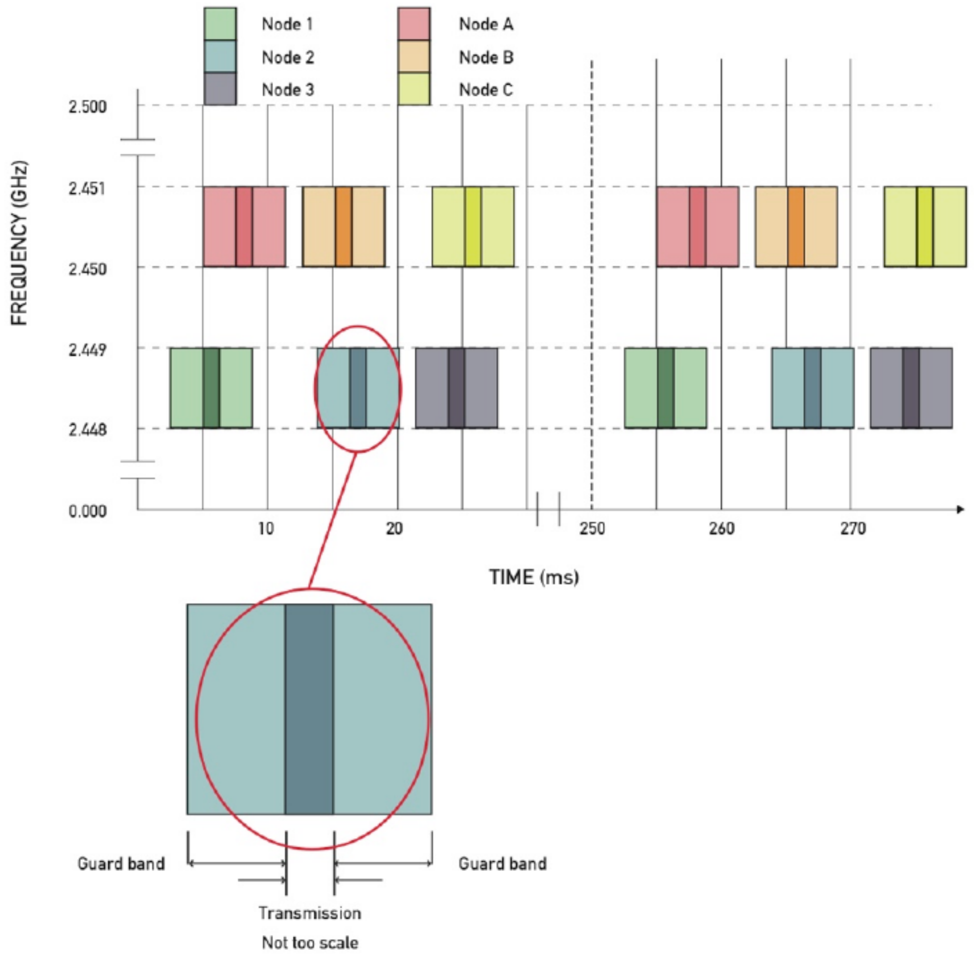


Figure 19 – ANT's adaptive isochronous scheme **(22)**.

As seen in Figure 19, the 2.4GHz frequency band is divided in several 1MHz channels, which are then divided into several timeslots. The timeslots are repeated according to a determined message period (250 ms in the example). A timeslot is comprised of two guard bands used to avoid interference and a transmission. Nodes 1, 2 and 3 share the same channel by using different timeslots while nodes A, B and C use overlapping timeslots with Nodes 1, 2 and 3 but have hopped into a different channel, thus avoiding interference (22).

3.3.4 Wireless Interference Studies

Despite the interference avoidance techniques that each technology employs, there is always a good probability that there will be a certain level of interference when more than one type of wireless network co-exists at a location. So it was important to research information about studies based on real experimentation that effectively measure interference levels between different wireless technologies. This section's purpose is to explain and present a few results from some research studies, while trying to determine potential co-existence issues that might arise if Wisedome were to work with multiple wireless technologies at the same location.

This section base itself mainly on two studies: Co-existence of Zigbee and WLAN, A Performance Study (68) and Coexistence of IEEE 802.15.4 with other Systems in the 2.4 GHz-ISM-Band (69). It is worth pointing out to the lack of independent research on newer wireless technologies such as ANT. It was not possible to find any scientific studies regarding wireless interference on ANT devices. Thus, there are only ANT's claims regarding its high interference immunity. From ANT's website, about a new family of ANT chips produced by Nordic Semiconductor, comes the following statement:

> *"This will make the nRF24P2 family (of ANT wireless chips) highly immune to disturbance from other 2.4GHz radios like Bluetooth® wireless technology and Wi-Fi™ operating in the vicinity. The net result will be fewer retransmits and lost packets (and hence even more power savings) in noisy RF operating environments (70)."*

In the Co-existence of Zigbee and WLAN, A Performance Study, the research team studied the effects of the co-existence between **ZigBee and Wi-Fi** and between **Wi-Fi and Bluetooth**. The tests were performed on a "test-bed" environment in an open indoor cubical office environment area, with no interference from any other radio frequency devices except for the ones intended: a Wi-Fi router, one Wi-Fi laptop, two ZigBee laptops and two Bluetooth laptops.

The team performed various experiments and measured the performance loss for each technologies throughput (the amount of bits transmitted per second). Most of the experiments consisted on measuring the interference between networks of different wireless technologies on a cubical environment, which is interesting since Wisedome is designed for assisted living facilities, which also usually have multiple rooms for the various patients. A Wi-Fi client was placed inside of a reference room communicating with a Wi-Fi router outside of the room. Additionally, two other devices communicating with each other were placed with the Wi-Fi client in between, as shown in Figure 20. The experiments alternate the wireless technologies used on these communicating devices and present the results for both ZigBee and Bluetooth technologies.

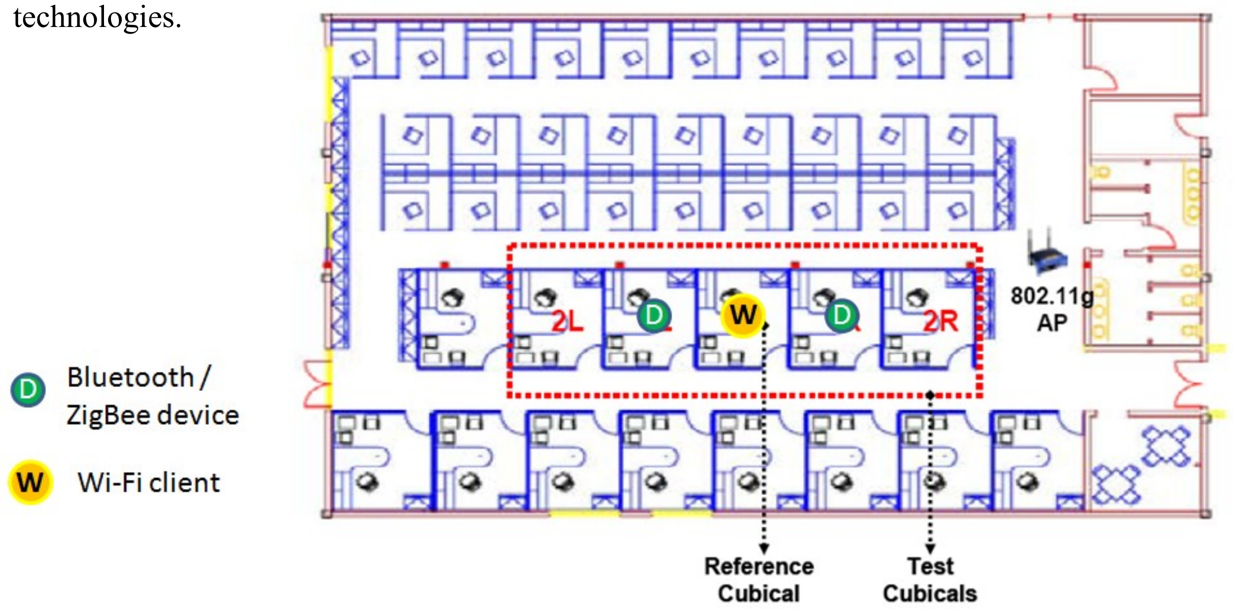


Figure 20 – Representation of the distribution of devices within the "test-bed". **(68)**.

Overall, this study shows that ZigBee has a good co-existence with Wi-Fi when different central carrier frequencies (the frequency that transmits most of the data) are used, but presents a bit of a performance loss when similar central carrier frequencies. Bluetooth also shows low performance loss when using separate central frequencies, but loss is considerably higher in this case when comparing with ZigBee. A possible explanation is that Bluetooth uses frequency hopping, so whenever it hops into a frequency that is close to the one being used by Wi-Fi, there will probably be interference, resulting in a greater performance loss. It is also implied that ZigBee fares better than Bluetooth against interference, even when similar central carrier frequencies are used. It is also show that Wi-Fi communications are suffer from much more intereference when co-existing with Bluetooth rather than ZigBee.

The study **Coexistence of IEEE802.15.4 with other Systems in the 2.4 GHz-ISM-Band**, shows yet some other interesting results regarding co-existence between ZigBee and Wi-Fi and between ZigBee and Bluetooth. The tests were performed **on worst case scenarios**, and therefore, on unrealistic situations of great bandwidth usage. Tests were performed where the frequency channel selection for the 802.15.4 (ZigBee) is kept constant while the Wi-Fi channel varies with time. The results show that the interference level is reduced with an increasing distance of the channels used by ZigBee and Wi-Fi. The interference level on ZigBee is the highest when its channel overlaps with the one being used by the Wi-Fi network, which is consistent with the results presented in the previous study. ZigBee communication suffers greatly from Wi-Fi interference, which is expected, especially as the transmission power of IEEE 802.11 8 (Wi-Fi) is about 30 times larger than the one of IEEE802.15.4 (ZigBee) and the intensity is also about 4 times larger (69). Other tests reveal that the packet loss rate is not monotonous with the distance between the ZigBee antennas for small distances (lower than 2 meters). For distances greater than 2 meters, the packet loss rate increases with larger distances. Thus, one can conclude that the use of different central carrier frequencies allows for ZigBee and Wi-Fi to co-exist without suffering from significant interference.

Another interesting test performed in this study was the interference experiment using Bluetooth devices. One notebook makes an FTP transfer to a PDA by using Bluetooth, achieving a medium data rate of approximately 15 Kbps. Another notebook makes a FTP transfer to a desktop PC using Bluetooth, with a medium data rate of approximately 50 Kbps. At the same time, IEEE802.15.4 (ZigBee) stations are communicating in the area.

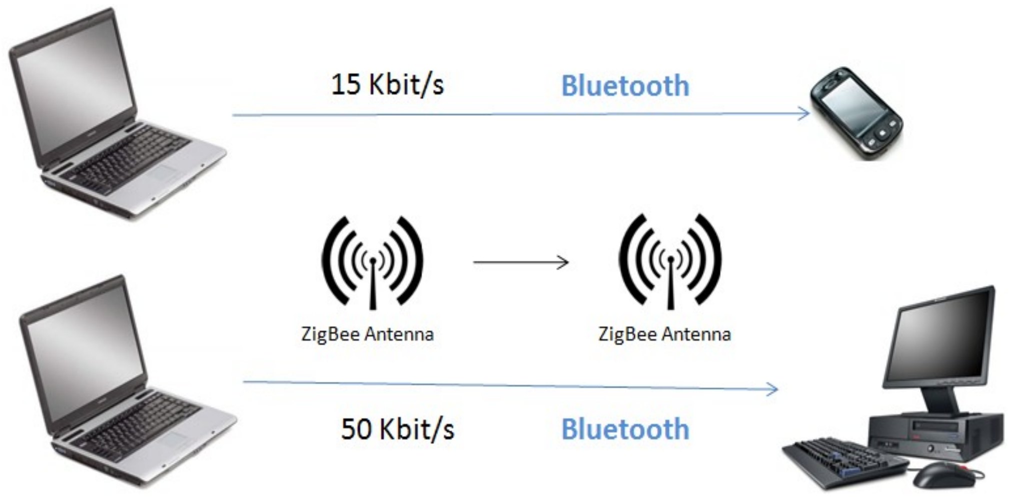


Figure 21 – Test setup used to measure the interference of Bluetooth on ZigBee and vice-versa.

In this test case, only about 10% of ZigBee data packets were lost. IEEE802.15.4 data may be destroyed by a Bluetooth transmission that occurs in the same time slot with the same frequency as ZigBee; in other words, when Bluetooth hops to a frequency similar to the one used by the ZigBee devices. No impact of the ZigBee stations onto the Bluetooth communications was observed (69). Considering that this is a worst case scenario, it is reasonable to assume that, under normal circumstances, ZigBee and Bluetooth can co-exist without major issues.

3.4 Comparison Matrix

The following matrix gives an overall and final perspective of the wireless technologies supported by ANT and Continua.

	ZigBee	Bluetooth	ANT
Max. node size	2^{64}	8 (master included)	2^{32}
Maximum Range	100 meters	10 meters	30 meters
Energy Consumption	ZigBee claims to have low energy consumption and research seems to back up that claim	Energy consumption levels high for Classic Bluetooth technology, making it inappropriate for portable devices. New Bluetooth Low Energy seems to address this problem.	ANT claims that it achieves Ultra Low energy consumption, much lower than its rivals ZigBee and Bluetooth.
Interference	ZigBee seems to have good interference resistance, although it may suffer from interference if it's central carrier frequency is coincides with other networks	Bluetooth frequency hopping gives good protection, although some interference may arise when hopping to frequencies coincident with other networks' frequency.	ANT claims to be highly resistant to interference issues do to its adaptive isochronous scheme.
Interoperability	Assured by IEEE 11073	Assured by IEEE 11073	Assured by Device Profiles
Number of Sensors	No Continua certified devices use ZigBee	Few Continua certified devices	Many certified sports sensors and some health sensors.

4 Integration Framework

At this project's beginning, Wisedome was only able to work with one type of hardware vendor. This section describes the efforts made to create an extension to the Wisedome system: the Integration Framework, which allows Wisedome to use several different hardware vendors, and more than one hardware vendor at a time.

4.1 Earlier Integration Attempt

Wisedome already had some embedded code that resulted from earlier attempts to support different types of devices. This was the result of two proof-of-concepts that used vendors other than the reference vendor; one proof-of-concept involving devices from a vendor that provided an ECG signal and Heart Rate, and another proof-of-concept using devices which monitored Heart Rate and allowed the use of the Internet to report values to caregivers. However, this preliminary code was still very primitive and it hadn't yet evolved past this proof-of-concept phase. The purpose of this earlier code was not to provide full functionality but rather, to prove that it was indeed possible to work with other types of devices and to retrieve and display ECG values on Wisedome.

At this phase, Wisdome's configuration file used to held information regarding the host and port values to be used for TPC/IP communication with each device vendor and, additionally, it held information that indicated whether or not to try and connect with these hosts and ports. During its startup, Wisedome would then read its configuration file and store its information on a configuration class. Integration at this point consisted mainly on a series of "if" verifications that would test the configuration class' contents as to whether the application was using reference vendor devices, ECG devices or the Internet communication devices.

Even so, despite this early attempt at integration, the application would only effectively work with one type of reference vendor. The Wisedome project's next milestone was to provide a working Health Monitoring solution by using this single reference vendor's devices only, so working with other types of devices never went beyond mere proofs-of-concept. Functionalities that supported the use of ECG devices and Internet using devices were very few, mainly consisting on the assignment and un-assignment of devices to a patient and retrieval of ECG or HR values. The support for these other vendors was also implemented in a very ad-hoc way, since the purpose was not to establish a definitive code implementation but rather to just prove that working with other vendors was possible. Also, the application at this phase would only work with one vendor device at a time and the existing device communication code was very coupled to the Wisedome application code. The addition of new types of devices required great effort to implement and adding new functionalities to the application would imply changing most of Wisedome's device communication code.

4.2 Integration Requirements

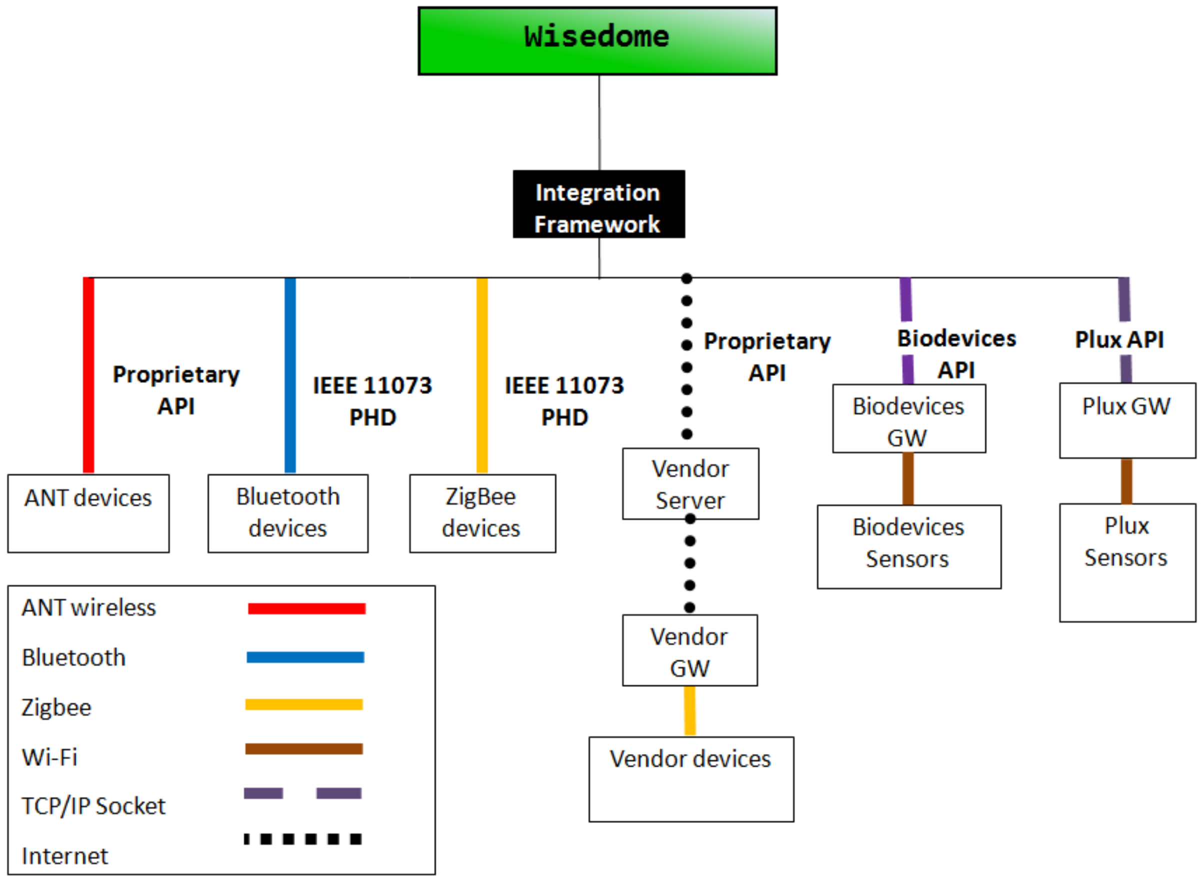


Figure 22 – Initial idea of the Integration Framework's function in the Wisedome System. The Integration Framework would abstract Wisedome from device communication, independently of the device manufacturer or the communication technology.

One of the lingering problems persisting in the Wisedome application was the lack of ability to support different types of sensor suppliers, and use them simultaneously. The earlier proof-of-concepts had proven that it would be productive to use a combination of different types of devices with different functionalities, thus allowing the costumer to choose what kind of device setup he would want to have implemented. Therefore, it was an important step on the application's development not only to support different vendors, but also to support multiple vendors at the same time, thus allowing more flexibility to accommodate different kinds of customer needs. The question was, how to create an integration framework that would allow a developer to easily write code for Wisedome to work with a new type of device but, at the same time, requiring as little amount of integration effort and changes on Wisedome as possible? Additionally, how do we ensure that it is possible to use more than one device vendor simultaneously? From these questions derived one of the most important tasks of this project: the integration of different health monitoring sensor suppliers onto a single health monitoring application.

In order to solve these integration issues, it was necessary to identify and accomplish a series of goals:

- **Wisedome should become sensor agnostic** - Wisedome should not distinguish devices based on their manufacturer. Wisedome shall also become oblivious to the base technology used to communicate with the devices.
- **Maintain current functionality** – It would be the integration framework's responsibility to adapt new devices to work with current Wisedome's functionalities. In other words, the integration framework has to make sure that device behavior, responses and events are in within the boundaries of what Wisedome can handle. This includes, for instance fitting every device into one of the already defined Wisedome basic device types: *Cheststrap*, *Bracelet* or *Caregiver device*. It also implies that the integration framework would have to perform the conversion of vendor specific measurements and thresholds such as fall detection sensibility levels, ECG signal amplitudes or heart rate values into values that are understood by Wisedome.
- **Create a loosely coupled / modular application** – Wisedome's implementation at the time was too tightly coupled with device communication. Although there was some

modularity due to two classes dedicated exclusively to sending and receiving commands from devices, there were still many device commands sent in application workflows. It was necessary to remove all sensor communication commands from Wisedome and move them on to the new integration framework in order to achieve true communication abstraction.

- **Remove device related workflows** – Many of the device configuration workflows were a part of Wisedome's application workflows. For instance, assigning a device to a patient implied not only performing Wisedome application workflow tasks such as updating assignment information on the database, but also performing parameter configuration of the newly assigned Plux devices. Since different vendors would have different device configuration workflows and different parameters to be set, it is easy to conclude that this implementation was not very integration friendly. So, it was necessary to first distinguish between device configuration workflows and Wisedome application workflows, separate them, and then move all the device configuration workflows on to the integration framework, while at the same time making sure that doing these tasks wouldn't alter or break Wisedome functionality.
- **Find a command issuing method common to all devices** – Since the objective was to abstract Wisedome from device communication, it was necessary to relay device commands (such as assignment, parameter configuration or request of vital signs) and handle the respective command responses in a way independent of the vendor manufacturer of the device being used.
- **Find an event reporting method common to all devices** – Issuing device commands is important, but also equally important is the ability to receive events from the devices. These events include not only the alarms that may be sent (for instance, fall alarms or call button alarms) but also battery events (if a device is running low on battery), location events, unreachable events and so on. True device communication abstraction implies that Wisedome does not distinguish between an alarm that is sent from a Device Manufacturer #1 or Device Manufacturer #2. Therefore, a vendor independent method of receiving events had to be devised.
- **Support a Multi-Vendor device environment** – As previously mentioned, one of the features that were desired to be included on Wisedome was the ability to support

different kinds of devices coming from different vendors, at the same time. Thus, any approach at creating an Integration Framework should allow the existence of multiple vendor devices co-existing at the same time.

- **Easy implementation of future devices** – The Integration Framework would have to be written in such a way that would minimize the integration effort when adding new devices to the Wisedome system.

After identifying the goals to be achieved by the Integration Framework, it was time to choose an appropriate technology to implement it.

4.3 Choosing a technology

Now that the integration goals were identified, it was necessary to select which technology would be used to implement the integration framework. Considering the list of goals described above, there are several desired features that were important to be present in the technology of choice:

- **Support a modular architecture -** The integration framework would support several vendors, all offering similar functionalities to Wisedome but with each vendor implementing the functionalities differently, in order to support a different type of hardware. An elegant and effective way of supporting this would be to use a loosely coupled software architecture: each vendor implementation would consist of a "module" and adding support for new vendors would be a matter of adding new modules.
- **Multiple modules running at the same time –** Since one of the goals is to support a multiple-vendor environment, it would be crucial for the integration framework to be able to run multiple vendor modules at the same time. However, it is also crucial that while the modules must be able to run simultaneously, they conflict neither with each other nor with the Wisedome application.
- **Support a Service-Oriented Architecture** – Running the application on a modular environment would mean that the Wisedome application side and the communication

side would run in different modules. Therefore, in order to exchange commands and events, the implementation technology must allow some kind of inter-module communication. This inter-module communication should have good performance so that the latency between an alarm event occurrence and the display of the alarm is not significantly affected by the integration effort. This idea of a modular framework where multiple vendor modules offer similar services to an application, but implemented in different ways fits the idea of a SOA. According to (71), a SOA is basically a collection of services, which communicate with each other.

- **Preference for Java based technologies** – As previously mentioned in section 2.2, the core Wisedome application was implemented using Java, therefore, technologies that use this programming language would be preferred.

Choosing the technology was a very natural process, since Wisedome was already implemented using a standard that includes all of the features mentioned above, and more: an OSGi based Framework.

4.4 The OSGi Specifications

4.4.1 OSGi Technology Usage

"The OSGi specifications define a standardized, component-oriented, computing environment for networked services that is the foundation of an enhanced service-oriented architecture" (72). Although it was initially designed to target residential Internet gateways with Home Automation applications, the standard has achieved considerable success in other areas of development such as smartphones, vehicles, telematics, industrial computers and high-end servers (72). The following section will present a few examples of areas where the OSGi technology is being used.

In today's mobile phone market, mobile application software plays a crucial role. Due to the increasing number of mobile applications filling the market, mobile phone manufacturers need a service platform that is scalable, flexible, reliable, and, most importantly, has a small footprint (72)(73). In response to this necessity, Nokia and Motorola, both being members of the OSGi Alliance, drove an OSGi technology standard for the next generation of smart phones. According to Jon Bostrom, Director of Java Technology, Nokia:

> *"The OSGi Service Platform is an open deployment platform which offers life-cycle management for mobile applications and services. It enables operators, enterprises, and mobile device and application manufacturers to dynamically extend the platforms features after manufacturing. For example, an IT Manager can install new APIs or applications to employees' existing mobile devices over-the-air independent of the mobile phone model." (72)*

This ability to dynamically extend an existing platform can be used to fulfill the need of Wisedome to support new vendor manufacturers in the future. The OSGi specifications allow for application scalability: implementing support for new vendors is a matter of adding new software modules.

The vehicle industry has also adopted the OSGi specifications by making them an intrinsic part of the GST specifications (Global System for Telematics specifications specify a set of services and libraries for the creation of mobile applications), which is supported by many car manufacturers (72). The BMW Group, for instance, uses the OSGi specifications as the base technology for its high-end infotainment platform, the ConnetedDrive (72). BMW's interest in the OSGi specifications comes from the desire to update its system's functionally in an easy way and to reduce costs by sharing one platform instead of having many different ones (72).

The OSGi specifications are also used in the home automation business. Siemens' *serve@home* is a system that includes a complete range of household devices linked via a power-line interface. These devices are coordinated in a central gateway running an OSGi Service Plataform which provides the aggregated services, thus allowing the user to control the various devices and appliances within the house via cellphone or Internet(72)(74). The greatest advantage of using OSGi in this case is also shared with Wisedome: drivers and scenarios can be updated and added after the initial sale, much like additional vendor drivers can be added to Wisedome.

OSGi specifications have also found their way into many other applications. For instance, Eclipse, a popular used open-source Integrated Development Environment (IDE) uses the OSGi specifications in its plugin system. OSGi specifications allow dynamically installing and updating of Eclipse's plugins, without requiring an application restart(72)(75). IBM also uses OSGi technology in many of its products, including the widely used WebSphere Application Server(76).

4.4.2 OSGi Specifications' Features

An OSGi framework is written in Java programming language. The OSGi specifications offer many useful features to be used in software development (72). The most significant features used in the implementation of the integration framework were:

Each software module is compiled onto a package – The OSGi specifications define each software component as a *"Bundle"*. Bundles consist of Java applications (or software components) compiled and packaged onto a standard JAR file. This allows for an easy distribution of new modules or updates for existing modules.

Run multiple bundles at the same time - One of the main reasons why OSGi is so widely accepted in the industry is because the OSGi service platform allows multiple Java-based components to execute together, simultaneously, in a single JVM and cooperate efficiently by using a SOA-based architecture(77).

Manage bundles life-cycles - Also important is ability to manage the life-cycle of each component without compromising the normal operation of the whole framework:

- *Install bundles* – The OSGi-based frameworks support the addition of new bundles to the framework without interrupting framework operation.
- *Start/Stop bundles* – Installed bundles can be started and stopped in an OSGi Framework. Starting a bundle initiates bundle operation and makes whichever resources the bundle provides to become available to other bundles on the framework. Stopping a bundle interrupts bundle operation and cleans up resources that were made available by it. In an OSGi Service Platform, all

applications are started in the same JVM, thereby saving memory, resources, and CPU cycles (72).

- *Update bundles* – Updating a bundle allows developers to update their bundle code, without having to reinstall a bundle completely.
- *Uninstall bundles* – Ends the bundle's life and removes the code and its resources from the system.

Java Package Importing and Exporting – The OSGi platform allows developers to share code between bundles. A bundle can export certain class definitions or libraries to the framework environment, which can then be imported by other bundles. This contribution of code is relevant because it reduces the amount of implementation on each bundle and the amount of total memory used by a system. Take, for instance, the following example: a certain class that occupies 30 Kb is used by six different applications. If these applications were running on a closed environment, each application would have to implement its copy of the class. This would mean that there are 150 Kb of memory wasted in comparison to a scenario where a single application implements the class and the others simply use it. Memory usage is reduced in an OSGi Service Platform because a library can be installed once and then shared to all bundle applications.

Service Oriented Model – A "publish, find and bind" service model is implemented on OSGi-based frameworks. This is an extremely important feature and was extensively used on the integration framework's development. A *Service* is defined as "a mechanism that allows one bundle to provide functionality to other bundles" (72). Please note that this is different from sharing class definitions and libraries: sharing a class definition only allows a bundle to share how to "build" an object based on that class. A Service is actually an actual object that is shared by a bundle in the framework.

Each bundle is, at any time, capable of registering new services on the framework's Service Registry. Registering a service means that that service becomes available for every bundle that wishes and has permission to use it. Bundles can request the service registry for a list of all the services of a desired type and can also use "service listeners". Service listeners listen to events related to services of their interest. The OSGi framework informs service listeners whenever a service of a type being listened to is registered or unregistered, thus allowing an easy management of the services available.

Services are always registered with an interface name (which defines the type of service), and a set of properties that distinguishes the service from others of the same type. For instance, let’s assume that there is an interface called Vendor Communication, and that there are two services (which are in fact, Java objects) implementing that interface. One of these services implements communication for hardware devices produced by “Vendor A” while the other implements the same communication methods but adapted for hardware produced by “Vendor B”. Although these services implement the same interface and, therefore, are of the same type, they implement the same functions in different ways. In order to distinguish these services it’s possible to register them with a property that possesses different values for each one. If we create a property called “Vendor Type” we can distinguish the services by accrediting different property values: we accredit the value “Vendor A” and “Vendor B” to the respective communication services. Now, whenever a bundle seeks services or a listener listens to events from services implementing the Vendor Communication interface, it is possible to filter the results and events to the vendors of interest(78). This filtering is done by using a simple yet powerful filter language named Lightweight Directory Access Protocol (LDAP).

This type emphasis on services allows developers to build loosely coupled software components that are then “glued” together by the Service Registry. One of the greatest advantages of the OSGi framework is that the Service Registry provides a dynamic system, where bundles and their respective services can be added and removed without disrupting the overall environment.

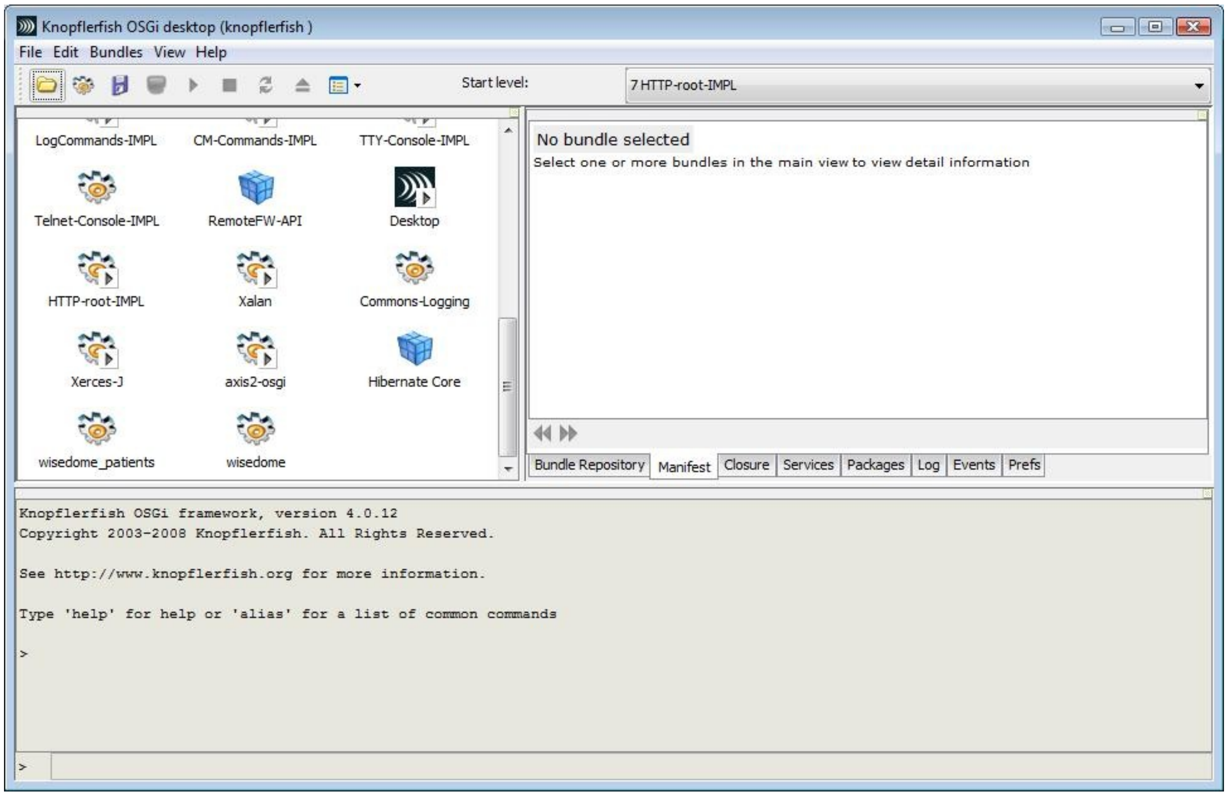
4.5 Implementation Process

The following section will describe the integration framework's implementation process, beginning with an explanation of Wisedome's state before the integration attempt, an overview of the framework's architecture and finally a description of the design and implementation process. Finally, some insight is given on how to extend the current framework to support new types of devices.

4.5.1 Before the Integration Framework

As previously mentioned, before the work on the integration framework begun, Wisedome was already implemented in an OSGi Framework environment.

There are various OSGi Service Platforms available. The one chosen to support Wisedome was Knoplerfish version 2.2.0. The Knopflerfish OSGi is a complete, open source, OSGi R3 and OSGi R4 distribution, led and maintained by Makewave(79).



```
Knopflerfish OSGi framework, version 4.0.12
Copyright 2003-2008 Knopflerfish. All Rights Reserved.

See http://www.knopflerfish.org for more information.

Type 'help' for help or 'alias' for a list of common commands

>
```

Figure 23 – Knoplerfish 2.2.0 UI. This figure displays a pre-integration framework implementation, containing the Wisedome, Wisedome_patients and Hibernate Core bundles.

Figure 23 displays the Knoplerfishe's UI display window implementing a version of Wisedome previous to the Integration Framework. Since it is running on an OSGi Framework, Wisedome is implemented on the form of Bundles. The most important Bundles for Wisedome were:

- *Wisedome* – This bundle contains the main Wisedome application and is responsible for device communication, alarm management, general database management amongst other functions. This Bundle is also responsible for providing the webservices to be used by the UI regarding location management, alarm events and alarm documentation, vital sign values, assignment and un-assignment of devices and device database management.
- *Wisedome_patients* – This bundle is also known as "Person Management Bundle" and is dedicated to managing system users (administers, nurses and TAPs) and patient information. It provides webservices to the UI for patient and user management.
- *Hibernate Core* – The Hibernate Bundle is an OSGi Bundle version of the Hibernate library, which allows an easy way to communicate with the PostgreSQL database by using POJOs. This bundle has been modified to include Hibernate mapping definitions important for the Wisedome's functionality.

At this point, the Wisedome Bundle was responsible for the device communication. Wisedome was designed to work with Plux devices by communicating with a gateway provided by Plux.

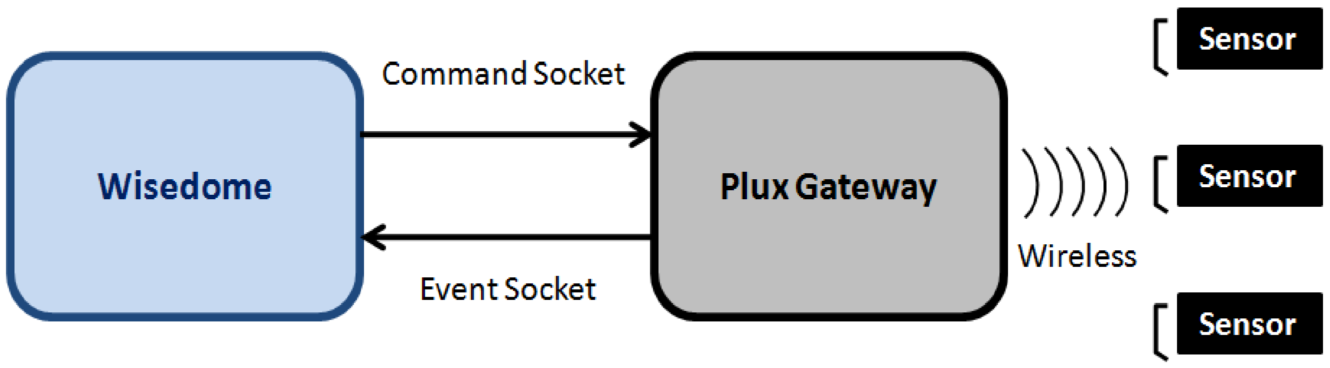


Figure 24- Scheme illustrating how Wisedome uses the Plux Gateway to communicate with the devices.

Device management consisted on sending commands to and receiving events from the gateway, by using network sockets. A *socket* is defined as "(...) one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent"(80). The Plux gateway consists on an application running on the same machine as Wisedome that directly manages wireless communication with the devices, by the means of an USB ZigBee-based router. There are two network sockets used to communicate with the Plux gateway. One socket named "*Command Socket*" is used to send commands to the devices to acknowledge alarms or suspend devices. A second socked named "Event Socket" is used to receive events from the devices, such as battery state changes or alarms. The API (the communication nomenclature) defined by Plux is based on strings with various comma-separated parameters. For every command sent by Wisedome, the Plux gateway returns an acknowledgement response accompanied by any requested values. The event port is only used to receive messages, so it's a one-way communication socket. Figure 25 shows an example of Wisedome requesting a heart rate value for a patient through the Command port and the Plux Gateway sending a call button alarm through the Event port.

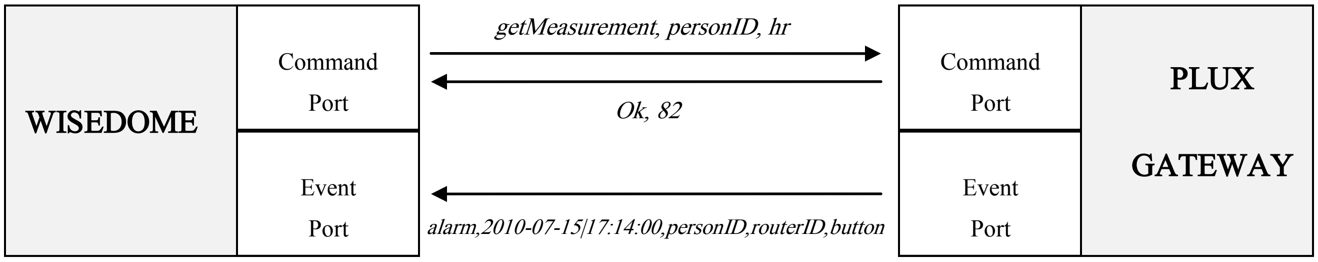

WISEDOME
Command Port
Event Port
getMeasurement, personID, hr
Ok, 82
alarm,2010-07-15|17:14:00,personID,routerID,button
Command Port
Event Port
PLUX GATEWAY

Figure 25 - Network socket communication between the Plux Gateway application and Wisedome.

The Plux API is a document that defines what ports and message nomenclature are to be used to communicate with its gateway. One of the particularities of this API is the fact that the Plux gateway stores the ID numbers of the persons assigned to a particular device. All commands sent to the gateway use the person ID as device identification. In order to initiate device use, it is first necessary to send an assignment command which includes the ID of the person to whom the device will be assigned. The gateway then returns an acknowledgement response, which includes the serial of the device that was assigned to the person. However, in order to issue commands to the newly assigned device, it is not the device serial that is used as an identifier, but the assigned person's ID instead. This is rather useful for the Wisedome application side since it is simpler to just send the ID of person who needs, for instance, a reading of heart rate values, instead of querying the database for the proper device serial each time the person needs his HR values measured. However it is important to keep in mind that not all future device vendors may store this kind of information.

During its startup, Wisedome would verify if it could connect to Plux's Command and Event ports and only then would it become operational. Most of the communication with the Plux gateway was managed in two Java classes named *WriteImplementation* and *ListenImplementation*. The *WriteImplementation* handled the commands to be sent by the devices, while the *ListenImplementation* would listen to incoming messages from the Event port, and then send them to a parser class.

It has been said before that Wisedome would only effectively work with a single reference vendor (Plux is used as an example in this thesis), despite the fact that some of its functions had preliminary code to be used with other device vendors. The system was incapable of using these other device vendors because Wisedome had several Plux-only commands scattered across its application code and also because this preliminary integration attempt was very "incomplete" and was insufficient to make the application work. Wisedome stored in its own configuration file device communication information, namely the IPs and ports used by the vendor gateways and whether or not it should try to connect with them. The preliminary integration code consisted on various *"if"* clauses that would test Wisedome's configuration file as to whether or not a certain vendor configuration was being used and, depending on the vendor, different APIs were used to send messages.

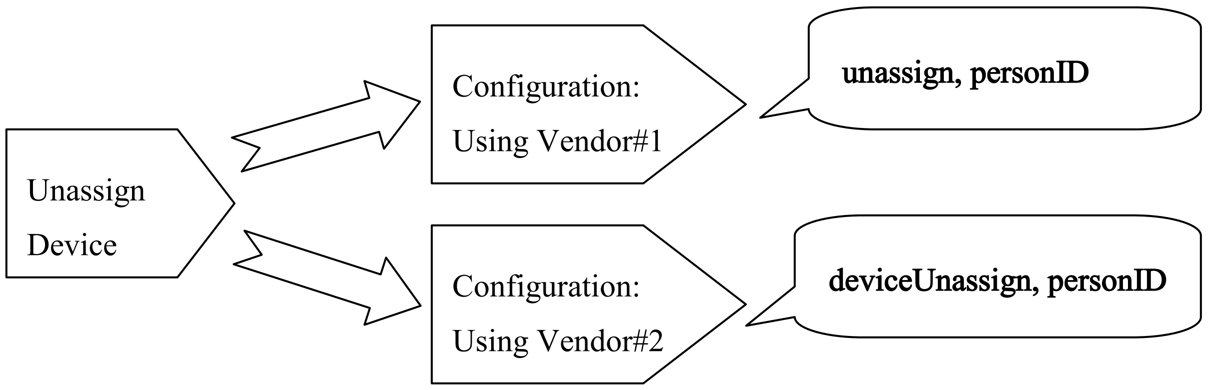


Figure 26 – Figure representing Wisedome's early integration attempt. Depending on the vendor being used, different methods of communication and different messages are sent, since each vendor has its own API.

It has also been mentioned that this integration attempt was written just as a proof-of-concept in order to prove that Wisedome could work with vendors other than the reference one, and was not designed to be a definitive integration solution. This approach at integrating devices was, actually, far from being an ideal solution:

- This type of vendor verification only works on a system designed to use only one type of device hardware vendor at a time. Keeping this implementation on a multi-vendor device environment would be extremely ineffective. In a normal application workflow, when sending commands to a patient's device, the application would have first to test for each device vendor if it was being used or not. For each vendor being used, the application would then have to query the database to find out if it effectively was the vendor of the device worn by the patient. This process would have to be repeated until the correct vendor was found. This, of course, is not an optimized way of solving this issue, since it would require multiple accesses to the database. It would be much more effective to first query for the vendor of the device assigned to a patient, and then use this information to communicate effectively.
- This type of implementation does not entice application modularity, since all of the code for communication with different vendors was in the same class methods, merely separated by the "if" clauses.
- Adding new vendors would require much effort because it would be necessary to search the whole application and then insert new "if" clauses followed by the

appropriate communication code whenever communication with devices was necessary.

Another issue that would difficult integration efforts was how application workflows were tightly coupled with device specific workflows. In several functionalities, Plux device workflows were an integrate part of Wisedome's workflows. One example of this was, for instance, the setting up of several device parameters during assignment, like fall threshold values and minimum and maximum accepted heart rates.

Some Plux related tasks were also implemented as "workarounds" to mold Plux devices behavior towards a desired result or even sometimes to circumvent bugs in certain functionalities. Some examples of these are, sending un-silencing commands to devices before un-assignment or sending both an alarm acknowledgement request and a un-silence command before trying to suspend a device.

It is possible that future device vendors might have the necessity for their own device specific tasks in order to ensure proper system functionality. Maintaining an architecture similar to the one described above would only bloat Wisedome application workflows with multiple "if" clauses for each and every of these device specific workflows. This would further reduce system modularity and would also, most likely, facilitate the introduction of bugs during development and make the process of tracking down and fix them more difficult.

4.5.2 Architecture Overview

One of the first conclusions drawn from studying the Wisedome implementation described above was that it was necessary to devise a completely new system to integrate different vendor APIs. The already implemented efforts presented too many problems and limitations to justify building an integration framework using them as basis. Since the whole system was to be running on an OSGi framework, it would make sense to use the features the OSGi specifications provided. It was then decided that in order to abstract Wisedome from device communication it would be necessary to create new OSGi Bundles specifically designed to handle that task.

The architecture that was agreed upon uses a Main Integration Bundle that manages several Vendor Integration Bundles, each implementing the integration with one device vendor.

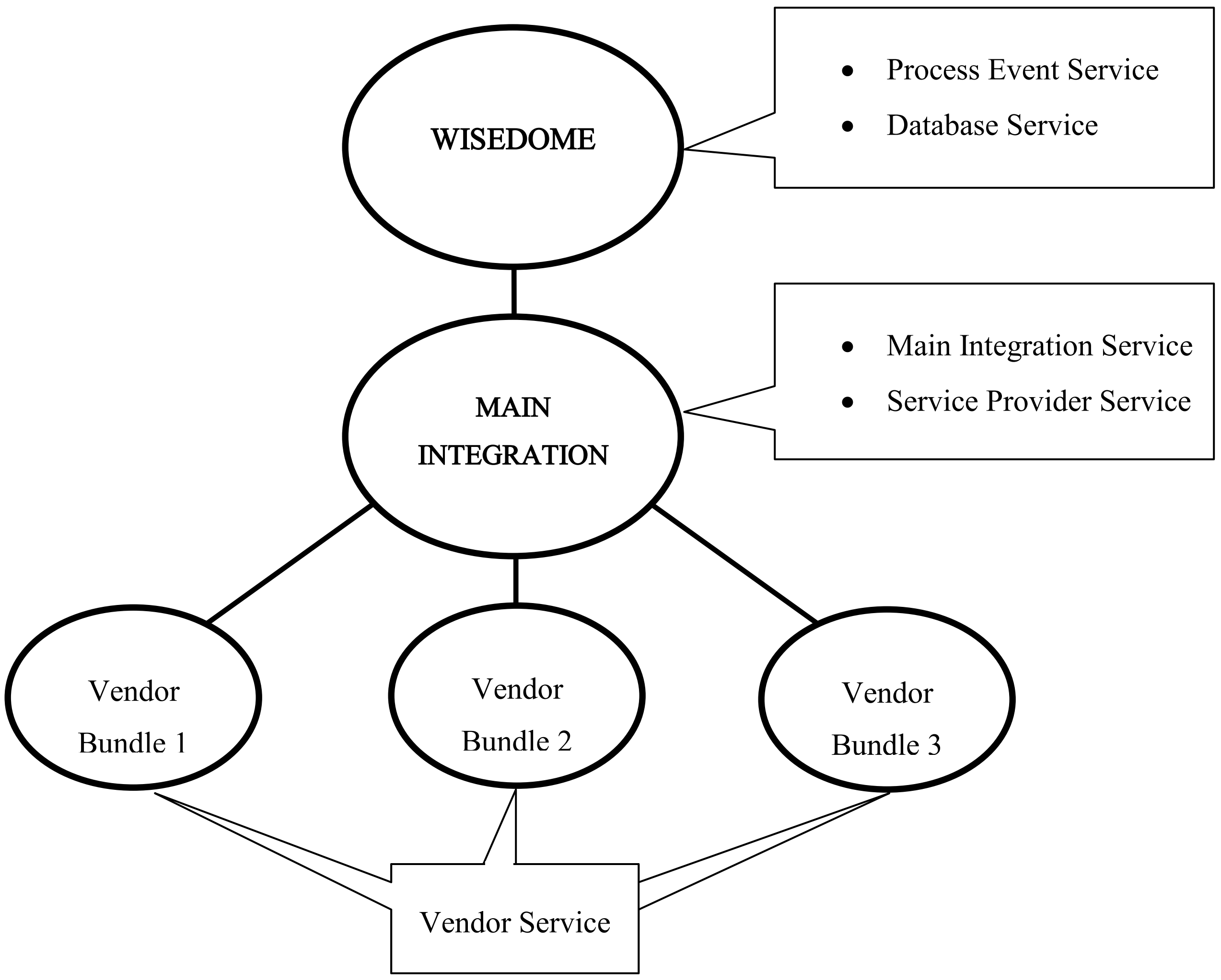


Figure 27 – Representation of the Integration Framework's hierarchy. Bundles are represented by the ellipses and the Services published by each bundle are mentioned inside the speech balloons.

An individual bundle is able to communicate with other parts of the system by registering services on the framework. These services are then used by other bundles, thus maintaining system modularity. The Integration Framework is composed of five types of services that allow Wisedome to communicate with the Main Integration Bundle which then communicates with the Vendor Bundles. This section will explain, in a general way, the purpose of each type of Bundle and Service available on the Integration Framework. A more detailed documentation of the methods provided by each service interface is available the Annex A of this document.

4.5.2.1 Main Integration Bundle

The Main Integration Bundle has five primary functions: to bridge communication between Wisedome and the several Vendor Bundles, to manage the services published by Wisedome so that the Vendor Bundles may access them easily, to manage the Vendor Services that are currently available on the framework, to initiate the necessary Vendor Bundles on application startup and, finally, to store the interfaces which define the methods that must be implemented by each service on the Integration Framework. To accomplish these tasks, the Main bundle publishes two services, the “Main Integration” and the “Service Provider”.

The Main Integration Service serves as the device communication abstraction layer and is used by Wisedome to send commands to devices, such as assignment requests, getting HR values or checking if a device is locked or unlocked. The Main Integration Service does not perform device communication itself, but is capable of calling upon the correct services provided by the Vendor Bundles to answer to Wisedome’s communication demands. The Main Integration Bundle keeps track of every Vendor Service that is currently available in the framework. Whenever a new vendor is registered or unregistered, the Main Integration Bundle sends an updated list of the current available vendors to Wisedome through the Process Event Service. When Wisedome calls upon the Main Integration Service, the Main Integration Bundle seeks the correct Vendor Service to provide a response. If, for some reason, the correct Vendor Service is not available, the Main Integration Service returns an error status, informing Wisedome of the situation. The Vendor Service and Process Event Service are described in sections 4.5.2.2, 4.5.2.3 and Annex A of this thesis.

Most of the methods provided by this service do not require references to the vendor of the device being used. Although it is possible to specifically define what vendor to use when calling a method, in most situations this information is omitted and Wisedome only sends the patient’s identification number (patient’s ID) as an argument. In case the vendor is not defined, the Main Integration driver queries the database to find out who is vendor of the device currently assigned to the indicated person. There are a few exceptions that do not allow the vendor to be omitted. The most notorious one is the **assignDevice** method: in case Wisedome is performing an assignment that means the person does not currently have any device assigned. Therefore, it is not possible to query the database for the vendor of the device assigned to that person. The vendor to be used in assignment must be chosen by the application user: depending on the patient's needs, a nurse chooses the most appropriate device to be assigned. Thus, the **assignDevice** method must know what vendor it should send an assignment request to. Any other method that does not receive a person's ID as an argument should also receive the vendor type information. This is the case for **getRouterState**, which is used to query if a given router is communicable or not. Methods used to check available vendors such as **getAvailableVendors**, **hasVendors** and **checkStartedVendors** do not need to receive neither a person's ID nor a vendor ID.

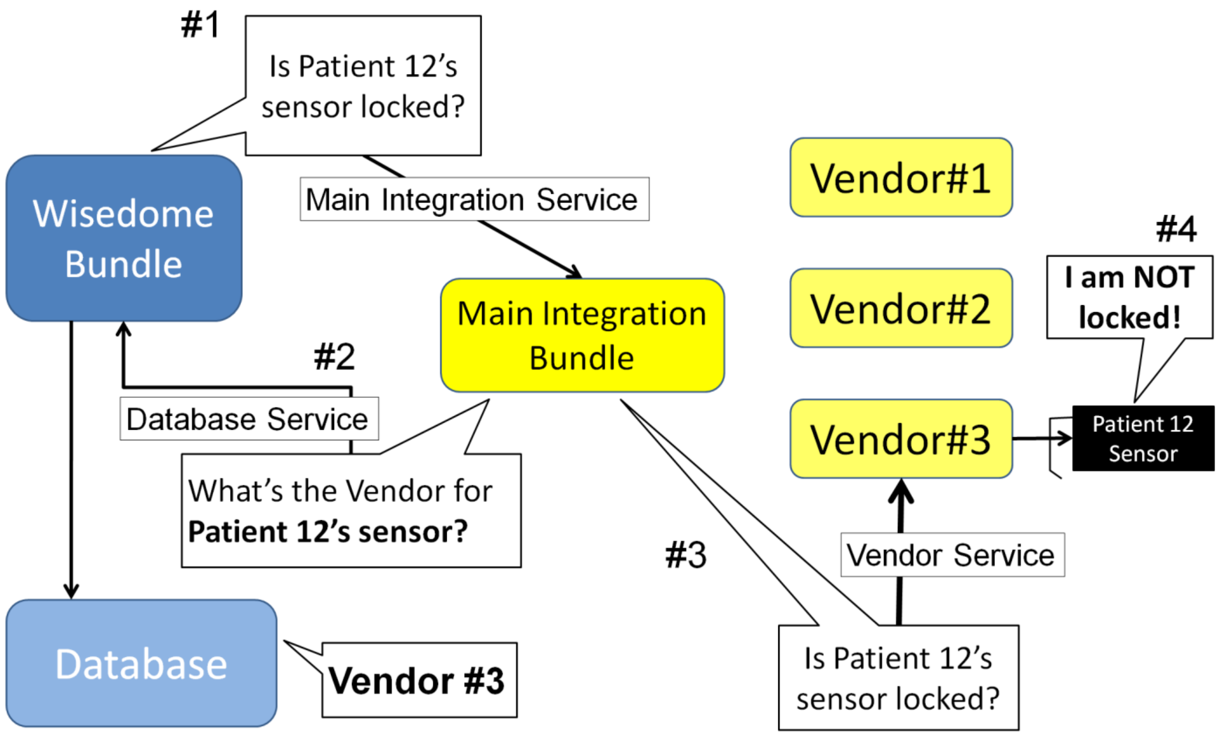


Figure 28 – Representation of how the Main Integration Bundle uses the various services to bridge communication between the Wisedome Bundle and the Vendor Bundles.

The Main Integration Bundle provides yet another service, called "Service Provider". As its name hints, the Service Provider's purpose is to "provide" services that are published by the Wisedome application to the various Vendor Bundles. One might pose the question: since a service is available to any Bundle on the OSGi framework, why do the Vendor Bundles not use the services provided by Wisedome directly? The decision to use a service just to provide other services came from desire to simplify the implementation of Vendor Bundles. The Main Integration Bundle performs the search, keeping and management of the Wisedome services so that the Vendor Bundles only have to retrieve them by using the Service Provider. If this service didn't exist all the Vendor Bundles would have to perform the management of both the Process Event and the Database services. The Service Provider spares Vendor Bundle programmers from this task so that each Vendor Bundle only has to manage a single service: the Service Provider. This reduces both processing power and memory usage and increases modularity. The Service Provider service in itself is rather simple and only has two methods: one method to return the Process Event Service and other method to return the Database Service. These two services will be further explained on section 4.5.2.3 and in Annex A.

This bundle is also responsible for starting pre-defined Vendor Bundles during startup. The whole framework is dynamic; hence, Vendor Services for Wisedome to use may be registered or unregistered at any time without disrupting Wisedome's function. However, it is convenient to define some Vendor Bundles to be automatically started during Wisedome's startup. These are defined on Wisedome's XML configuration file, which is read by the Wisedome Bundle during startup. The Main Integration Bundle then requests the Wisedome Bundle for a list of Vendor Bundles to be started and starts them accordingly.

Services in the Integration Framework are all defined by using Java interfaces. An interface is a group of related methods with empty bodies(81). Interfaces cannot be instantiated - they can only be *implemented* by classes or *extended* by other interfaces(82). As previously explained in page 56, a Service in an OSGi Framework is in fact a Java Object, which is defined by a Java class. In order to declare a service, the class should implement the correct service interface. These interfaces hold information regarding the methods that must be implemented by the service classes, thus guaranteeing that, no matter how the class is implemented, the necessary methods will exist. The Main Integration module contains the interfaces necessary to implement all the services used in the Integration Framework. These interfaces are held in a Java package, which is exported to the OSGi Framework and then imported by other Bundles.

4.5.2.2 Vendor Bundles

The Integration Framework also houses a series of Vendor Bundles. Vendor Bundles represent the part of the system where communication with devices effectively occurs: each one of them provides communication with one type of device hardware vendor. It's their responsibility to handle startup connection and necessary device setup routines. Each Vendor Bundle is also entitled to a XML configuration file where important information such as gateway IPs and ports or default device configuration values may be stored.

A Vendor Bundle is known to the whole Integration Framework by its Vendor Name. The Vendor Name must be unique and constant throughout the whole framework. It is used on several occasions such as the Bundle's JAR filename, the bundle's symbolic name on the framework, the XML Configuration file's filename, in the Vendor Service's "*VendorName*" property tag and in Wisedome's database.

Each Vendor Bundle must implement the "Vendor Service". The Vendor Service is used by the Main Integration Bundle to call upon methods that are used to perform device related tasks such as assigning a device to a patient, silencing a caregiver device or setting new fall detection sensitivity levels. The Vendor Service has many methods that are used to perform device communication with names similar to those of the Main Integration Service. If, for instance, Wisedome needs to know if a patient's device is locked it will use the **isDeviceLocked** method from the Main Integration Service. The Main Integration Bundle will then select the appropriate vendor for that patient and call upon the **isDeviceLocked** method of the respective Vendor Service. In order to register its Vendor Service on the framework, a Vendor Bundle should use a *java.util.Properties* object to set the service's "*VendorName*" property tag. The "*VendorName*" tag should correspond to the vendor's Vendor Name. This is important, as it will allow the Main Integration Bundle to correctly identify the vendor associated with the registered service.

Vendor Bundles can communicate events to Wisedome by using the Service Provider and calling upon the Process Event Service. The Process Event Service will be discussed on the next section.

4.5.2.3 Wisedome Bundle

The Wisedome Bundle is a modified version of original Wisedome Bundle, reworked so that the application would work with the new Integration Framework. This Bundle is the core of the Wisedome Health Monitoring application and still performs all of the underlying alarm management workflows that it did before. It suffered, however, various modifications in many classes and had its former communication classes removed, specifically the *WriteImplementation* and *ListenImplementation* classes. The Wisedome Bundle communicates with the rest of the Integration Framework through its two services: the Process Event Service and the Database Service.

As stated before, it is necessary for the Main Integration Bundle to access the database to, for instance, request sensor related information such as the hardware vendor. Some Vendor Bundles also need to have access to information stored on Wisedome's database, such as what kind of events are currently active, what is the sensor's lock state on the database, what is the serial of the sensor associated with a person and so on. It is for these types of requests that the Database Service exists. The Database Service provides multiple methods to access database information relevant to the integration bundles. The implementation of this service on Wisedome's side handles all the queries to the database, and then returns the results through the Database Service methods. The Database service also offers some non-database related methods, such as the **getConfigFilePath** which returns the path of the Vendor Bundle configuration files and the **getStartVendorsList** that returns the list of vendors to be automatically started by the Main Integration Bundle.

The Process Event Service is particularly important to Wisedome's application workflow. This is the service that Vendor Bundles use (after using Service Provider to retrieve it) in order to report events to Wisedome. Events include all types of alarms, such as fall alarms, call button alarms and heart rate alarms. Events should also be sent whenever a device changes its location, becomes locked or unlocked, has its battery depleted and so on. Vendor Bundles retrieve event information sent by the sensor devices and then use the Process Event Service to inform Wisedome. Wisedome's implementation of the Process Event Service processes the incoming events and integrates them into the overall system workflow.

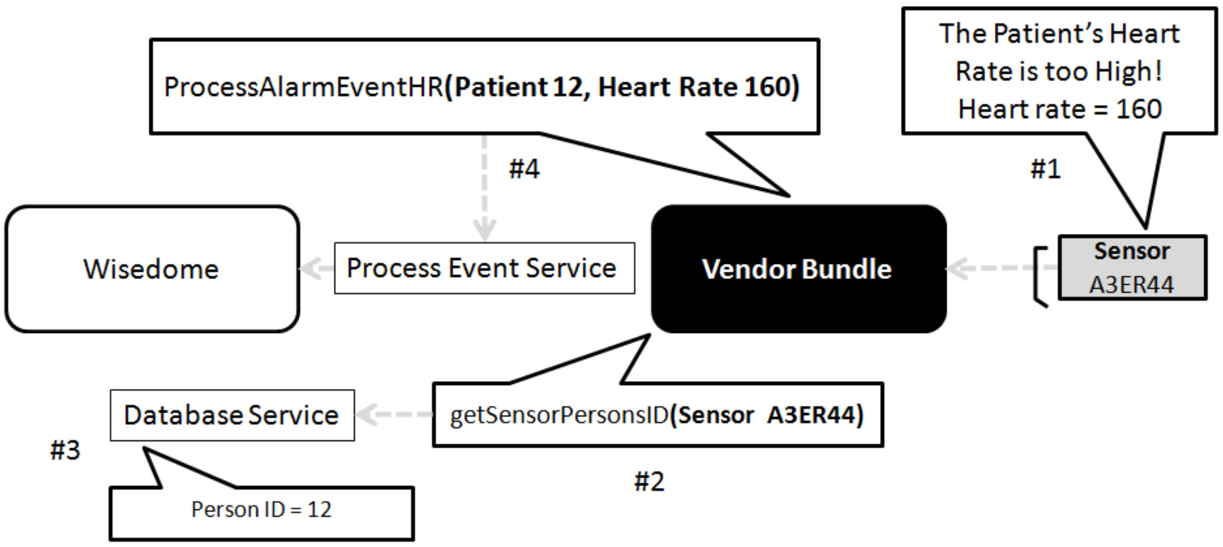

ProcessAlarmEventHR(Patient 12, Heart Rate 160)
The Patient's Heart Rate is too High! Heart rate = 160
#4
#1
Wisedome
Process Event Service
Vendor Bundle
Sensor A3ER44
Database Service
getSensorPersonsID(Sensor A3ER44)
#3
#2
Person ID = 12

Figure 29 - Simplified schematic of the workflow of an event raised by a device. The Process Event Service allows Vendor Bundles to report alarms to Wisedome

4.5.3 Service Methods Overview

This section will give an introductory description of the type of methods belonging to the most relevant services of the Integration Framework implement. Each service is described with more detail on Annex A of this thesis.

4.5.3.1 Database Service Methods

The database service has methods that provide information relevant to the Vendor Bundles and the Main Integration Bundle.

The service provides specific methods, important for other bundles in the Framework. Examples of this are the method **getConfigFilePath** that Vendor Bundles should use in order to retrieve the correct system path to their own XML file, or the **getStartVendorsList** that the Main Integration Bundle uses to know what vendor it should start automatically during system startup. **getVendorName** is also used to retrieve a Vendor Name, based on a vendor's ID.

Other methods are more directed towards retrieving user information. The method **personImp_getPatientSettings** is used to retrieve general information about a person's settings such as the fall alarm sensibility level or the maximum permitted heart rate. These settings might be used for proper device configuration. Also important is to know if a given person is an independent patient, a dement patient or even a nurse or a TAP, and that information is made available by the **personImp_getUserType method.**

The service also hosts various methods that allow for person – vendor – device management. Methods **getPersonSensorVendor** and **getPersonSensorSerial** are used to retrieve the Vendor Name of the hardware manufacturer and the serial of a device associated with a given person. It's also possible to know the ID of the person associated with a device based on the device's serial, by using the **getSensorPersonsID** method.

There are also methods used to create events on Wisedome's database. The method **createVendorDownEvent** creates a "vendor down" event that warns Wisedome that a given hardware vendor stopped being available, while the method **silenceVendorDownEvent** is used to "silence" these "vendor down" events when the vendor come back up. Method changeRouterState can also be used to directly manipulate a router's state on Wisedome's database, informing Wisedome if a given router is communicable or not.

Finally, there are several methods that are used on several workflows to retrieve the current status of a device from Wisedome's database. Such methods are the **isPersonDeviceMuted**, **isPersonDeviceSuspended**, **isPersonDeviceCharging**, **isPersonDeviceDischarged** and **isPersonDeviceUnlocked** which return the current database status for whether or not a device is muted, suspended, charging, discharged or unlocked.

4.5.3.2 Vendor Service Methods

The Vendor Service makes sure that Wisedome is capable of retrieving necessary information from the devices and send necessary comands.

There are methods for assignment and un-assignment workflows named **assignDevice** and **unassignDevice**, respectively.

Several methods allow Wisedome to directly query a device for its status:

- **isDeviceMuted** – Used to know if a device is muted or not.
- **getHrMin** – Used to retrieve the minimum "normal" heart rate value on a device.

- **getHrMax** – Used to retrieve the maximum "normal" heart rate value on a device.
- **getFallThreshold** – Used to retrieve the currently set fall sensitivity level on a device.
- **isButtonAlarmEnabled** – Used to know if button alarms are active on a device.
- **isHrAlarmEnabled** – Used to know if heart rate alarms are active on a device.
- **isFallAlarmEnabled** – Used to know if fall detection is active on a device.
- **getPendingAlarms** – Used to know how many alarms are currently active on a caregiver device.

For each of these statuses, there are other methods that allow Wisedome to set their value:

- **muteDevice** – Used to mute or un-mute a device.
- **setHrMin** – Sets the minimum "normal" heart rate value.
- **setHrMax** – Sets the maximum "normal" heart rate value.
- **setFallThreshold** – Sets the fall sensitivity level on a device.
- **setButtonAlarmEnabled** – Activates or deactivates button alarms on a device.
- **setHrAlarmEnabled** – Activates or deactivates heart rate alarms on a device.
- **setFallAlarmEnabled** – Activates or deactivates fall detection on a device.
- **setPendingAlarms** – Sets the number of pending alarms on a caregiver device.

Other methods are used to retrieve a patient's vital signs from the devices. The method **requestHeartRate** is used to request the current heart rate measure from a device, while **startECGStream**, **stopECGStream**, **getECGStream** and **getECGValues** are used to request ECG information.

There are also several methods used on various application workflows, such as **suspendDevice** and **resumeDevice** that allow Wisedome to suspend (a device no longer raises alarms) or resume a device. Some methods are also used in device configuration, such as the **isDeviceLocked**, which checks if a device has its locking mechanism opened or closed, the **getDeviceType**, which returns if a device is a chest-strap, bracelet or a caregiver device, and the **getBatteryState** which allows Wisedome to estimate how much charge does a device battery have left.

Some methods are used to know more about the devices' and routers' identification, and location and status. The method **getDeviceSensorSerial** returns the serial of a given device while the **locateDeviceRouterSerial** method is used to know the serial of the router that a device is currently communicate with. Also, **getRouterState** allows Wisedome to know if a given router is communicable or not.

Finally, there are the alarm workflow management commands, the **acknowledgeAlarm** and the **cancelAlarm**, which are used to acknowledge or cancel the alarms on a device.

4.5.3.3 Process Event Service

The Process Event Service contains various methods that allow Vendor Bundles and the Main Integration Bundle to "warn" Wisedome in case there is an event coming from the devices.

The Main Integration Bundle uses the method **ProcessVendorStateChange** to inform Wisedome whenever a new vendor becomes available, or a previously available vendor suddenly becomes unavailable.

There are also various methods dedicated to events regarding changes of state within the sensor device. **ProcessBatteryStateChange**, **ProcessLockedStateChange** and **ProcessUnreachableStateChange** are used to inform Wisedome of a change on the device's battery level, locking mechanism and network availability, respectively.

Several methods are used to report alarm events to Wisedome:

- **ProcessAlarmEventButton** – Used to report call button alarms.
- **ProcessAlarmEventFall** – Used to report fall detection alarms.
- **ProcessAlarmEventHR** – Used to report heart rate alarms.
- **ProcessAlarmEventLocation** – Used to report location alarms (when a patient enters a restricted area).

Other methods are used in device workflows, such as the **ProcessUserDeviceReplacement** used when a low battery device is replaced with a charged one, the **ProcessDeviceRouterChange** used to inform Wisedome that a device is communicating with a new router, the **ProcessSuspendDevice** and the **ProcessResumeDevice** used to inform Wisedome that a nurse is trying to suspend or resume a device by using device button sequences, and the **ProcessRouterStateChange**, used to inform that a router has become communicable or incommunicable. The **ProcessHeartRate** is also used in device workflows, and allow to periodically, or by request, communicate the currently measured heart rate value to Wisedome.

There are also two methods reserved for alarm management button sequences, the **ProcessAlarmAckRequest** which processes acknowledgement requests and the **ProcessAlarmEventCancel** which processes alarm cancelation requests.

4.5.3.4 Main Integration and Service Provider Services

The Main Integration Service has many methods with identical names to those of the Vendor Service, used to abstract Wisedome from particular vendors. It is the Main Integration Bundle's responsibility to choose the appropriate Vendor Service to respond to Wisedome's requests, whenever it uses one of the Main Integration Service methods. The Main Integration Service does possess, however, three methods exclusive to it. The **getAvaiableVendors**, **hasVendors** and **checkStartedVendors** methods are used by Wisedome to ascertain what vendors are currently available and to check if the auto-started vendors are active.

The Service Provider Service is very simple and only has two methods, **getDatabaseService** and **getProcessEventService** that return the Database and Process Event services, respectively.

4.5.4 Design and Implementation Process

Although the Integration Framework's architecture and implementation are more stable and consistent, it was not always so. The creation of the integration framework up to this point was an iterative process that will most likely continue as long as Wisedome is endowed with new improvements and features. Development began with the idea of using a Main Integration Bundle and various Vendor Bundles. Some limitations of the current implementation were ascertained at the beginning, while others were discovered during the creation process. The issues and limitations found during development gave birth to many of the functionalities presented in the previous section. This section has the objective of presenting to the reader the most relevant changes that were made to the Wisedome application during the Integration Framework's development, as well as the most significant problems that arose and how they were solved.

The first step was to isolate communication methods from Wisedome, and transcribe them into a new Vendor Bundle. This task also drove the creation of the Main Integration Bundle and Service. Wisedome was analyzed and the communication code used with the Plux gateway was slowly transcribed to what was later known as the "Plux Bundle". The methods were initially named after the Plux API, but later it was realized that it was not easy to discern the method's purpose from its name, so the method's names were changed. The communication classes *WriteImplementation*, *ListenImplementation* and the Parsing class were removed and their tasks transferred to the new Vendor Bundle.

4.5.4.1 ReturnObject Class

Messages exchanged between applications through a socket are nothing more than byte streams representing text strings. The Plux's API uses comma-separated strings to return the responses to command requests, which included both the acknowledgement of the request and, if applicable, the value requested. In case something goes wrong, the acknowledgement responses would return an error status, informing the health monitoring application of why it wasn't possible to perform the request. For instance, the Plux gateway would return "DeviceUnreachable" if devices were out-of-range of router communication or "InvalidName" if the command message was malformed. Wisedome used these acknowledgement returns in its workflows in order to handle the situations where a command was not performed successfully. The same level of functionality can be maintained while using vendor bundles, as long as bundle developers have the means to send acknowledge messages to the Wisedome application.

While Wisedome used to perform the parsing of the text strings received from the Plux gateway, it would no longer make sense for it to continue to do so in an Integration Framework. Each vendor bundle shall parse the messages received from their respective devices and pass these on to Wisedome. However, the vessel used to pass the responses from device commands had to be common to all the vendor bundles because, if Wisedome had to distinguish messages coming from different vendors, there wouldn't be much of a communication abstraction at all. Therefore, it was important to devise a means of passing information coming from different vendors in a standardized and effective way. While working on a Java environment, it would make no sense for vendor bundles to return comma-separated strings like Wisedome was prepared to receive. It would be much more effective to define a specific class to contain the information coming from the vendor bundles, a class which Wisedome would know how to handle. This way, as long as the vendor bundles are able to fit the command responses coming from the devices onto the return class, Wisedome wouldn't be able to tell the difference between responses coming from different hardware vendors. It would be appropriate for this class to allow vendor bundle programmers to pass on different acknowledgement status much like Plux used to do on their API. Also, it should return values, such as heart rates or locked state Booleans, as Java basic data types. This common vessel used as a response to command methods is known as the *ReturnObject* class.

The *ReturnObject* class is capable of returning various data types such as Booleans, Arrays and Doubles and also returns a "Request Status" integer, which describes if the command to the devices was successful or, if not, defines what kind of problem has occurred. The *ReturnObject* also pre-defines values that may be returned for battery levels (ok, low, depleted) or device types (chestband, caregiver or bracelet). One limitation of using a unique return class for every command method is that there has to be consensus on what kind of values are returned on each method. Vendor bundle programmers must know that, for example, the method **isDeviceLocked** returns a *ReturnObject* object with a Boolean value, and that the other possible return values (arrays, strings, and doubles) are null. This limitation can be surpassed with documentation, which can be consulted in Annex A of this thesis.

One of the problems solved by using the ReturnObject class was how to handle situations when a particular function is not supported by a device. Although Wisedome is capable of working with devices that supply patient's heart rate values, some devices might not be able to provide them due to a lack of hardware capabilities for doing so. Whenever a particular device vendor is not capable of performing one of the methods defined by the Main

Integration Service, it may return a *ReturnObject* with the request status defined as "*Not Supported*". Wisedome must then be able to handle this response.

4.5.4.2 Modifying Wisedome Bundle

Making Wisedome work with a new integration framework was far from being just removing its communication classes. The whole application had to be adjusted to start using the new Main Integration Service, instead of the communication class *WriteImplementation* and use the ProcessEventService to receive new events, instead of listening to messages in the *ListenImplementation*. It was necessary to analyze every instance where Wisedome recurred to device communication and adapt the code to use the new services. It was also necessary to adapt Wisedome to use the new *ReturnObject* by replacing the parsing of acknowledgement messages returned by Plux to use the *ReturnObject* values and request status instead.

The Wisedome Bundle startup had to be reworked. The application startup routines were heavily coupled with the establishment of communication with the Plux gateway and this had to be changed. Now, where once the application would verify if it could connect to the Plux gateway, it queries its service listeners instead, to check if the Main Integration Service is available and if any Vendor Services are published. New integration related tasks were also added, such as registering the Database and Process Event Services and starting the Main Integration Bundle in case it is not already running.

More important than just replacing the communication classes, was to separate device specific workflows from Wisedome workflows. As explained in section 4.5.1, page 62, Wisedome workflows were tightly coupled with device specific workflows. It was necessary to abstract the whole application from these device specific tasks, which lead in some cases to realizing the necessity of creating new Main Integration Service methods, beyond those necessary to communicate with the current vendors.

A good example of this situation is the alarm cancelation workflow on Wisedome. It is possible for a user to cancel currently active alarms on the health monitoring application. This workflow involves resolving the alarms on the database and sending alarm acknowledgement requests and mute commands to the Plux devices. Although methods like **acknowledgeAlarm** and **muteDevice** exist on the Main Integration Service that accomplish these tasks, nothing guarantees that other types of devices will require these exact commands to successfully cancel an alarm. They may even have a cancelation command of their own, or require the usage of other types of commands. Clearly in this situation, the alarm cancelation is a command on its own right that needs a new Main Integration Service method which will allow Wisedome to be abstracted from sending the alarm acknowledgement requests and mute commands that may not be necessary for other types of devices. This particular situation led to the creation of the **cancelAlarm** method, on the Main Integration and Vendor Services. There were other situations of device specific workflows that weren't always easy to detect and isolate, spread throughout Wisedome's implementation but for the sake of simplicity they will not be mentioned here.

4.5.4.3 Changing the Database Structure and Creating the Database Service

As mentioned in section 4.5.2.3, the Main Integration Bundle and Vendor Bundles need to query the Wisedome's database for several types of information. It would have been possible to import the libraries necessary to perform database queries directly on each of these integration bundles. However, this approach raises some issues. Wisedome is not a static application and is constantly being modified in order to add new features and to fix issues. In a situation where each integration bundle performs database queries directly, database modifications could possibly affect every bundle in the framework. Thus, with multiple Vendor Bundles and the Main Integration bundle, each database change could require much effort to implement. So, for the sake of a loosely coupled application and to avoid the situation described above, it is preferable that the Wisedome bundles handle all the database communication. This led to the creation of the Database Service.

One of the obstacles that had to be surpassed was how Wisedome was designed to use a person's ID as an argument to sent commands to the correspondently assigned device. As mentioned in page 60, Plux was capable of using person IDs sent by Wisedome to identify the devices to which it should relay the commands to. However, this may not hold true for every hardware vendor to be used with Wisedome. Assuming that every vendor will at least have some sort of "device serial" to identify each individual device, a viable solution would be to allow the Vendor Bundles to know the serial of a device currently assigned to a given patient.

Fortunately, Wisedome maintains a record on its database for every device currently being used on the system and its respective serial number. It also stores information on which device is assigned to each person. Surpassing the problem described above was just a matter of creating a method in the Database Service, **getPersonSensorSerial,** which allows Vendor Bundles to access the serial of the device currently assigned to a person. They may then use this serial to communicate with the correct device, as illustrated in Figure 30.

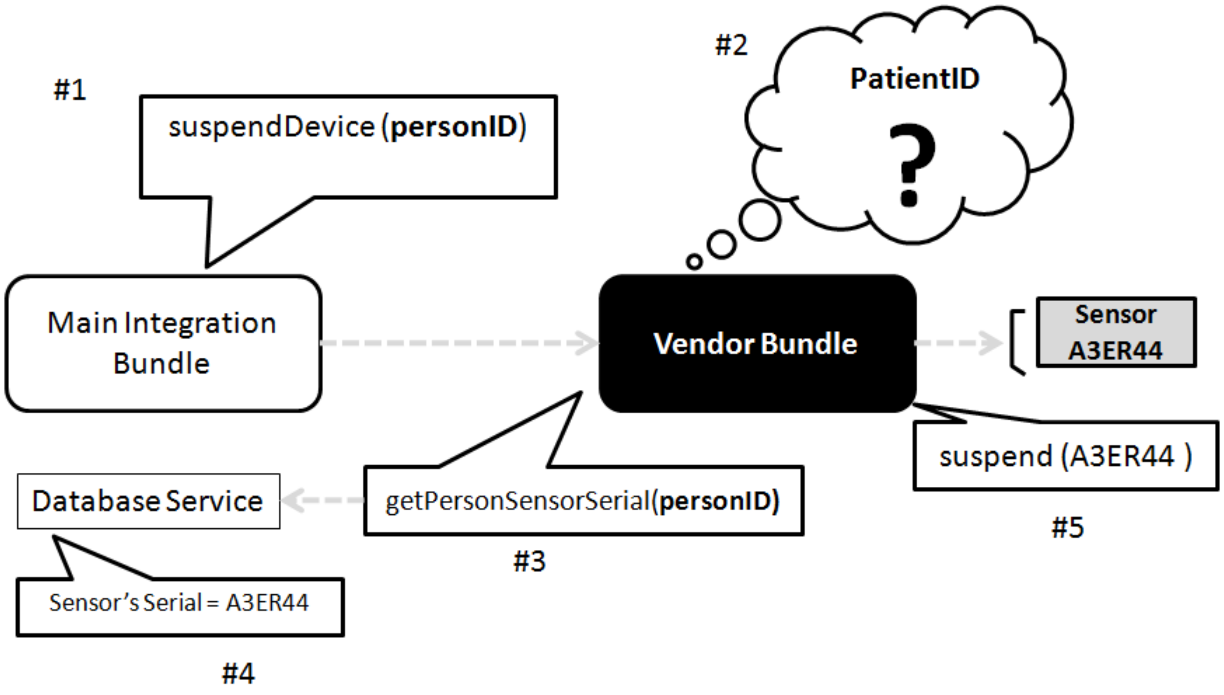


Figure 30 – Illustration of the workflow for the bundles of hardware vendors that do not hold information regarding person ID's.

Another obstacle that had to be surpassed was how to allow the Main Integration Bundle to know which vendor to communicate with when receiving commands from Wisedome. Unfortunately, Wisedome was only designed to work with one type of device, so while it maintained information regarding the serial of a device, it did not maintain any information regarding the vendor of a device. It was necessary to modify the database to include this information. The table that contained information on the devices was modified to include a new column with a vendor ID. This vendor ID was a foreign-key that referenced to a new table containing the Vendor Names of all the vendors currently supported by the system. Since the Vendor Names are unique throughout the application and are also present in the properties of the Vendor Service, the Main Integration Module would then use this information to seek the correct vendor service for each command request, as shown in Figure 28. The method **getPersonSensorVendor** was created on the Database Service, which returns a String containing the Vendor Name of the hardware vendor of the device currently assigned to a person. The webservice methods used to handle device management tasks, such as adding a sensor or editing sensor information were also modified to support the inclusion of the vendor ID.

The vendor ID is also used as an argument in all the Main Integration Service methods that need to know which vendor to be used, as explained in section 4.5.2.1, page 65. The Main Integration Bundle converts the provided vendor ID to a Vendor Name by using the Database Service Method **getVendorName**, and uses this information to seek the correct Vendor Service. This also implied changing the webservice invoked by the UI that is used to assign devices to start providing the vendor ID. This makes sense since it is the user (most likely a nurse) that will chose in the UI which type of device hardware should be assigned to a patient, after considering the patient's necessities.

Another change made to the database was the addition of two new types of database events. Database events are created and stored in various situations such as when a device triggers an alarm, a nurse inserts a note or a new patient is created. They allow Wisedome to keep track of the system's behavior. The new events were specifically created to handle situations with multiple device vendors. The first event is known as "Vendor Down" event, and is created whenever a Vendor Bundle that was configured to startup automatically is unable to do so or when a Vendor Bundle ceases its function due to the loss of communication with the devices. The second event is known as "No Vendors" and is created when there are no Vendor Services available on the framework. This will result in an alarm being displayed on Wisedome's alarms display in order to notify the users of this issue.

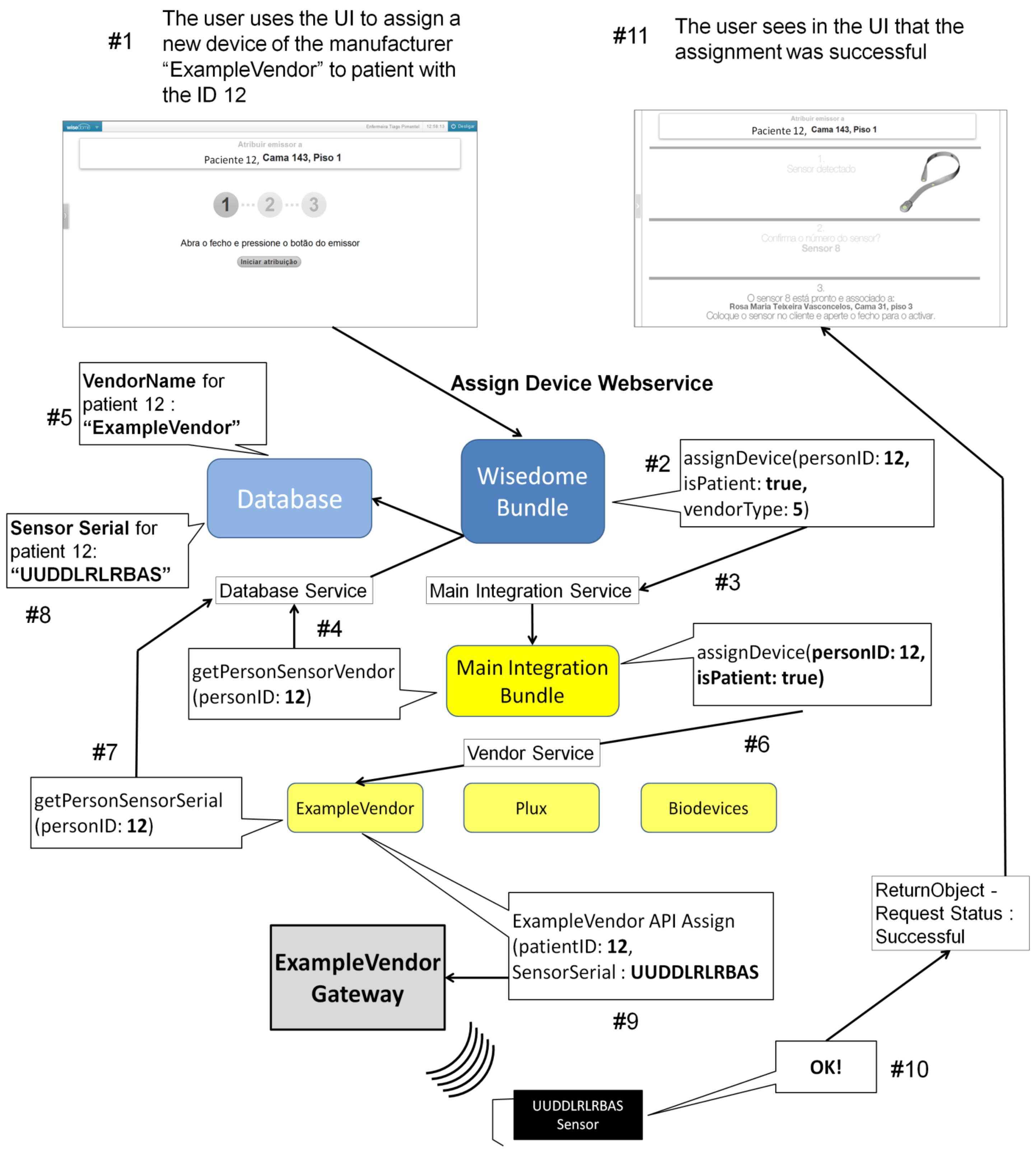

#1
The user uses the UI to assign a new device of the manufacturer "ExampleVendor" to patient with the ID 12
Atribuir emissor a
Paciente 12, Cama 143, Piso 1
1 2 3
Abra o fecho e pressione o botão do emissor
Iniciar atribuição
#11
The user sees in the UI that the assignment was successful
Atribuir emissor a
Paciente 12, Cama 143, Piso 1
1. Sensor detectado
2. Confirma o número do sensor? Sensor 8
3. O sensor 8 está pronto e associado a: Rosa Maria Teixeira Vasconcelos, Cama 31, piso 3
Coloque o sensor no cliente e aperte o fecho para o activar.
Assign Device Webservice
#5
VendorName for patient 12 : "ExampleVendor"
Database
Wisedome Bundle
#2
assignDevice(personID: 12, isPatient: true, vendorType: 5)
Sensor Serial for patient 12: "UUDDLRLRBAS"
#8
Database Service
Main Integration Service
#3
#4
getPersonSensorVendor (personID: 12)
Main Integration Bundle
assignDevice(personID: 12, isPatient: true)
#7
Vendor Service
#6
getPersonSensorSerial (personID: 12)
ExampleVendor
Plux
Biodevices
ExampleVendor API Assign (patientID: 12, SensorSerial : UUDDLRLRBAS
ReturnObject - Request Status : Successful
ExampleVendor Gateway
#9
OK!
#10
UUDDLRLRBAS Sensor

Figure 31 - Representation of a complete workflow for an assignment request.

4.5.4.4 Abstracting Device Configuration, Device Values and Configuration files

It was also necessary to abstract Wisedome from device configuration, since each type of device would have its own characteristics and settings to be configured. Device configuration is an integrate part of some of Wisedome's workflows: for instance, when a user defines new settings for a patient on the UI, such as new minimum and maximum heart rates or new fall detection sensibilities, the UI uses a webservice to update these changes on the sensor devices. It is important to update the patient's sensor device settings to reflect these new values. However, each type of hardware can have different set of functionalities: some devices might support fall detection, others might not. So how could Wisedome know if it should send a **setFallThreshold** command to a device or not?

There were two solutions considered to resolve this problem:

- Wisedome could have become completely abstracted from device configuration and use a generic method for setting device values which would receive every possible setting (heart rates, fall detection levels, alarm cancelation permissions) and let each vendor bundle sort what information would be useful to use for configuring the device
- Wisedome could maintain a basic level of knowledge of types of devices, and only send the appropriate commands for each basic type.

Eventually, it was decided to opt for the second alternative. Wisedome handles three basic types of devices:

- *Caregiver* – This is the device used by nurses and TAPs, which beeps when there are active alarms and allows for alarm acknowledgement routines.
- *Bracelet* – This is the simplest type of device, usually assuming the form of a bracelet and only possesses call button capabilities.
- *Cheststrap* – This is a more advanced type of device that includes call button, fall detection and heart rate detection functionalities.

Every device in the system shall be classified as one of the three basic types of devices. In case a device does not support one of the functionalities defined in the basic type, it should send a “Not Supported” response to commands related to that feature.

The reason why this solution was chosen was because Wisedome was designed to use these basic types and already had various workflows that would depend on them. Some of these workflows were part of core features, such as not allowing patients classified as “dementia” to use chest-straps. However, this solution can be viewed as a limitation of the current integration framework implementation, as it is discussed in section 6.1.1.

Some device configuration values had also to be abstracted. This was mostly related with configuring fall detection sensitivity levels for different Vendor Bundles. Wisedome is designed to support three distinct levels of fall detection sensitivity: high, medium and low. Even if a device supports fall detection, nothing guarantees that it has the same pre-defined values for fall detection thresholds. For example, although Wisedome has three defined levels of fall detection sensitivity, a device hardware vendor might configure its fall detection sensitivity level with a value ranging from 0 to 200. In this case, it is necessary for the vendor bundles to “convert” the vendors fall detection sensitivity level into something that Wisedome would understand. In order to allow some flexibility and room for future improvements, Wisedome might set the fall sensibility as a value between 0 and 40, where:

- 0 is the equivalent to low fall sensibility
- 20 is the normal fall sensibility
- 40 is the high normal sensibility.

These values are consistent with the information stored on Wisedome’s database and it’s up to the vendor bundles to convert these values to the nearest equivalents of their hardware solution. It was only necessary to perform this type of abstraction for fall sensitivity levels, but if new features are added to Wisedome in the future, it may be necessary to define new standardized values for those.

As mentioned in page 60, Wisedome already maintained a configuration XML file where it stored important configuration information such as database TCP/IP communication host and port values or the time it takes to promote an unattended low priority alarm to a high priority alarm. This configuration XML file was also used to maintain device specific configuration values, such as the host and ports of the Plux gateway application, the default fall threshold values for Plux devices, among others. On a multi-vendor environment, it wouldn't make sense to store this type of configuration on Wisedome's XML configuration file, since it would regard many different types of vendors. Therefore, it was necessary to allow each hardware vendor to store configuration information in their own file. Each Vendor Bundle is now entitled to one XML configuration file (stored in the same location as the Wisedome's configuration file) that should be read and handled by the bundle itself. This file may contain any configuration values deemed necessary for the bundle's proper function.

Wisedome's own configuration file was also modified to allow users to define which Vendor Bundles are to be automatically initiated on application startup by inserting the correct Vendor Names. As explained in section 4.5.2.1, page 66, the Main Integration Bundle uses this information to startup bundles automatically; however, bundles may still be started or stopped while Wisedome is running without disrupting the system's normal functionality.

4.5.4.5 Main Branch Application Merging

The Critical Health uses the CVS (Concurrent Versioning System) while developing software. The Integration Framework was developed in a separate CVS *branch* of the project, which means that it was developed in parallel as an *alternate* version of Wisedome. When the development of the Integration Framework begun, the current state of the Wisedome application was branched onto 2 different paths: one of these paths continued Wisedome's development normally, without any integration, while the other resulted on a modified version of Wisedome working with the Integration Framework. Thus, the development of the Integration Framework did not occur on a static implementation of Wisedome: while the Integration Framework was being developed on the *integration branch*, Wisedome was also being changed and improved on the *main branch* with new features and bug-fixes constantly being added. One of the challenges in development was for the *integration branch* to keep on par with the changes of the *main branch* in order to avoid building the Integration Framework onto a completely outdated version of Wisedome. The process of introducing the changes made on the *main branch* into the *integration branch* is called *branch merging* and consists on carefully revising the *main branch* code modifications and passing them on to the *integration* *branch* while making the necessary modifications to support the new integration features and assuring that the new code does not break the *integration branch* functionality. Since a small change can have repercussions on the whole system, merging was a complex and rather tiresome task, but necessary in order to assure the Integration Framework's usefulness.

4.5.5 Developing New Vendor Bundles

Developing new vendor bundles allows Wisedome to work with new types of device hardware vendors. Vendor Bundles act as "drivers": computer programs that allow higher-level computer programs to interact with a hardware device (83). The Integration Framework was designed with the intent of making the process of creating new drivers as simple as possible by offering interfaces that define what methods should be implemented and also by providing services (Process Event and Database Services) that allow driver developers to easily communicate with Wisedome. During the creation of the Integration Framework, two device drivers were developed (for Plux and Biodevices) and documentation on how to create new ones was also written. Driver developers also have access to a "backbone" driver which consists on a basic skeleton from which new drivers may be created, containing various comments explaining what should be implemented and why, what does each resource do and preliminary documentation of each service available to the vendor bundle. This section's purpose is to explain the basic structure of a device driver and to make reference to the resources that allow for the creation of new drivers.

4.5.5.1 Basic Structure of a Vendor Bundle

Next, a suggested driver structure is presented. It should be noted that this is merely a suggestion: the system does not bound driver developers to any pre-defined structure. As a long as the Vendor Service is correctly implemented and registered on the OSGi Framework, Wisedome will be able to use it effectively.

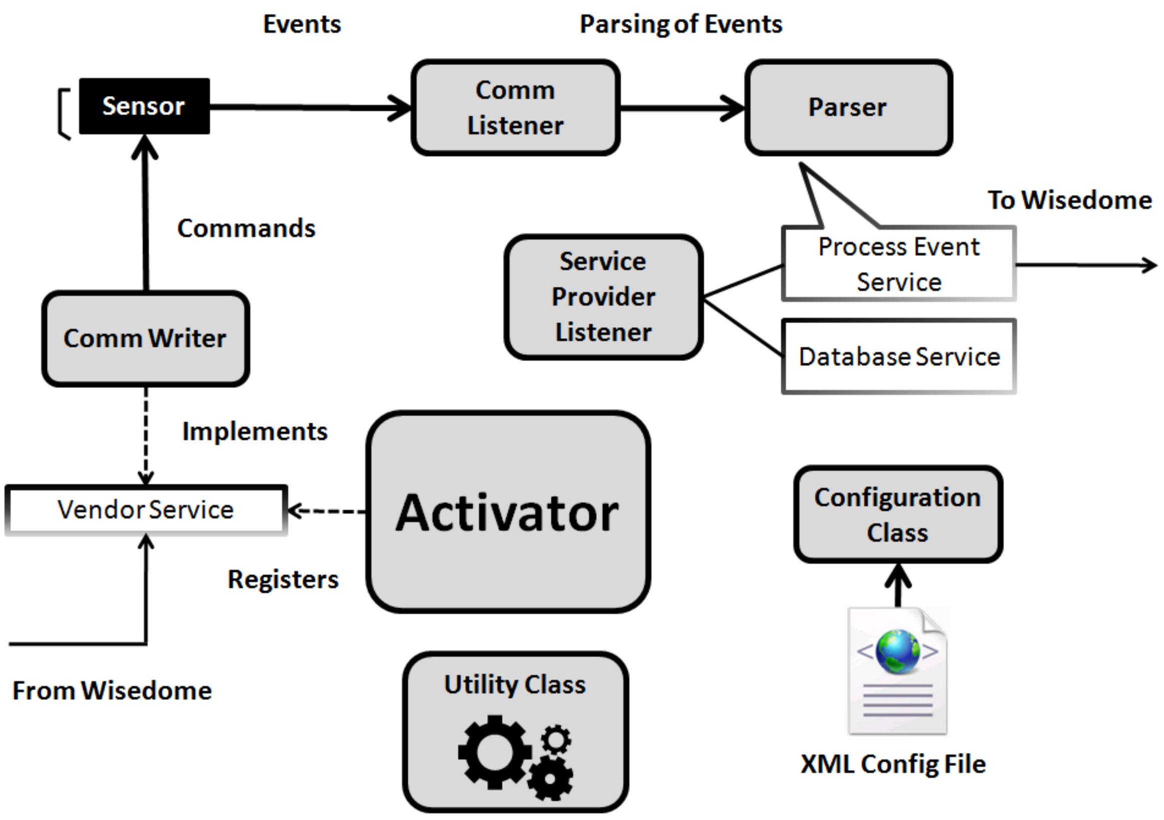


Figure 32 – Scheme illustrating various components of the Vendor Bundle.

- **Activator** – The activator class implements the OSGi's *BundleActivator* interface and is an integrate part of every OSGi bundle, functioning as the bundle's main class. It defines what a bundle should do when it is started and how it should be stopped(84). For Vendor Bundles, this class usually executes any startup workflows such as establishing communication with the devices and setting up initial parameters. This class is also usually responsible for registering the Vendor Service and setting up the Bundle's service listeners.
- **Service Provider Listener** – This class usually implements the OSGi's *ServiceListener* interface and seeks the Service Provider provided by the Main Integration Bundle. It is responsible for managing the Service Provider object and handling what happens when the service becomes unavailable.
- **Configuration class** – The configuration class acts as repository for the information defined in the vendor's XML configuration file. It holds this information in memory for easy access by other parts of the bundle.

- **Communication Listener class** – The communication listener is responsible for listening to incoming events from the devices, such as alarms or device state changes. Whenever a new message arrives, it should be sent to the Parser class.
- **Parser class** – The parser receives messages from the communication listener and converts the information within so that it may be used as arguments for the Process Event Service methods. After parsing the information, the Parser class calls the Process Event Service to notify Wisedome of the event.
- **Communication Writer class** – The communication writer is responsible for sending commands to the devices, such as setting up parameters or silencing devices. Usually, it is this class who implements the Vendor Service interface and all the methods it defines, such as **assignDevice**, **isButtonAlarmEnabled** and so on.
- **Utility class** – This is a helper class that provides several methods that are used by other classes. Examples of methods usually included on the utility classes are date format conversion methods or device parameter conversion methods that convert the information sent by the devices to a format that is understood by Wisedome.

4.5.5.2 Backbone Example and Service Documentation

The Backbone bundle serves as a skeleton driver that may be used as basis to write new drivers. The Backbone driver example respects the structure described above and has already some basic tasks implemented, such as registering the Vendor Service and implementing the Service Provider Listener. The driver serves as a tutorial and contains many comments explaining why is it necessary to include a certain line of code and some preliminary documentation on the methods that must be implemented and resources that are available to the vendor bundle.

It is important for the driver developer to know, for instance, what type of tasks should each of the Vendor Service methods implement, what kind of ReturnObjects he should return for each method, what kind of Process Event Service methods he should call upon for each type of alarm and what functionalities does the Database Service method offer. This information is present in the Integration Framework documentation, which is in Annex A.

5 Wisedome's Quality Management Processes

Additionally to wireless technology research and software development, the work described in this thesis also involved working on Wisedome's quality management process. It is worth noting that the work described in this section is independent from the work described in other sections of this thesis. This work is related to the quality management of the Wisedome application and not with integration efforts.

As a spinoff of Critical Software and part of the Critical Group, Critical Health's quality management system (QMS) is heavily based on the original Critical Software's QMS. The development of software at Critical follows the processes defined by the QMS, which consist on a set of procedures, guidelines, checklists, templates and other documents (85). As described in (85), Critical defines its QMS as a system to improve product development time and overall quality:

> *"The QMS is designed to enable projects and other activities to achieve their objectives efficiently and effectively, by the use of proven defined processes and practices. Where problems arise or errors are found, the QMS enables changes to be made and lessons to be learned. It also ensures that the focus of Critical's QMS is satisfying customers and improving products and services efficiency and effectiveness."*

Critical's QMS is comprised of a series of processes that define and guide the initiation, development, delivery and support of Critical's products and services. The product development phase is mostly guided by the engineering processes. The engineering processes define and describe how projects are managed and developed throughout their lifecycle, from requirements analysis and specification, passing through design and construction and finishing with product validation through customer acceptance and product maintenance (85). Within Critical's QMS exist five engineering processes:

- **Requirements Analysis Process -** This process guarantees that high level customer requirements are analyzed and detailed in a way that can efficiently be translated to the project. This is important to guarantee that the final product will conform to the costumer's needs.

- **Software Design** – This process ensures that software design is produced with a high level of quality and that the design will be able to support the Software Construction process, assuring that software requirements are fulfilled.
- **Software Construction** – The objective is that software development is conducted with a high level of quality, responds to all software requirements, promotes code reuse and allows requirement traceability. This process also guarantees that the software implementation properly reflects software design.
- **Software Testing** – This process is driven by the necessity of determining if the software is compliant with the defined requirements and to ensure software's stability and quality.
- **Maintenance** - The purpose of this process is to modify a system/software product after delivery to correct faults, improve performance or other attributes, or to adapt to a changed environment.

Besides the work on the Software Construction process (with the development of the Integration Framework described in section 4 of this thesis) there was also work done in the Requirement Analysis and Software Testing processes. This section's purpose is to describe what kind of work was performed in these areas, which contributed the improvement of Wisedome's quality and allow further participation on Wisedome's development process.

5.1 Requirements Analysis

Prior to the beginning of this project to obtain the Master's degree in Biomedical Engineering by Coimbra University, there was some work developed during an internship on Critical Health, mainly consisting on requirement analysis, which was further continued during the course of the project here described. The objective of requirement's analysis is to guarantee that the customer's expectations of the software's functionality are met in full. This involves specifically defining every single feature that the software should support in order to meet customer's needs and expectations. These expectations and needs come in the form of requirements that the software must meet.

Requirements definition is a critical step towards the success of a project. It is not an easy task, and requires proper documentation. Requirements should be written in way that they are non-ambiguous, easy to understand, testable, accurate, non-excessive or repeating and with a sufficient level of detail to allow proper software and hardware implementation (86) (87). Requirement analysis often goes further than just describing costumer's needs, but also describe every necessary feature in order to produce a fully functional product, including safety, usability and design attributes. The analysis comprises different levels of detail, starting with high-level business requirements that state the overall customer needs such as "the software should only allow authorized users to have access to certain functionalities", following to deeper, lower-levels of detail that include specific software implementation necessities such as "the software shall have a username field in the login menu".

In the case of Wisedome, it was necessary to read several documents such as project proposals, costumer meeting minutes and other internal documentation. It was also necessary to use earlier prototype versions of the software in order to grasp a complete idea of the project and figure its necessities in order to write proper requirements. There were literally hundreds of requirements written, that fulfilled different levels of detail and referenced different parts of the system. It was necessary to write business level requirements, system requirements and software requirements.

Requirements also played an important part when handling with device manufacturers, as it was necessary to define specifically what kind of features a sensor device should possess. Features such as its maximum permitted weight (since the target population is elderly people, it is not convenient for the sensors to be heavy), minimum caregiver device beep sound intensity, device button configuration or device wireless interference resistance had to be taken into account.

Wisedome's own software requirements were equally important to define. One of the main reasons for this is that, requirements are not merely static definitions of what a system must to, that are written just as a keepsake. Requirements drive other parts of the quality management process, particularly system testing. While writing requirements, it's very important that they are kept unambiguous and simple, because a requirement will then be used to create system tests that are used in order to verify if the software is behaving accordingly to what has been defined. Thus, each test is designed to verify if a given requirement has been fulfilled and always traces back to this same requirement that originally led to its creation. Software features such as the maximum permitted time between the detection and display of an alarm event, the necessity of security for the database information, options available in menus or the ability to add and remove patients from the system need to be properly defined in requirements. Design features are also important, such as the need to have different alarm display colors for different alarm priorities and different icons for each type of alarm.

The Requirement's analysis is not static, but an iterative process: requirement definitions must be reviewed with the project's team and continuously evolve as the project itself evolves. As the project advances, requirements are updated to conform to new project needs. The requirements written as part of the work developed during this project's execution merely served as a basis for later writing a more complete set of requirements. Such task was not a part of the work that led to the writing of this thesis.

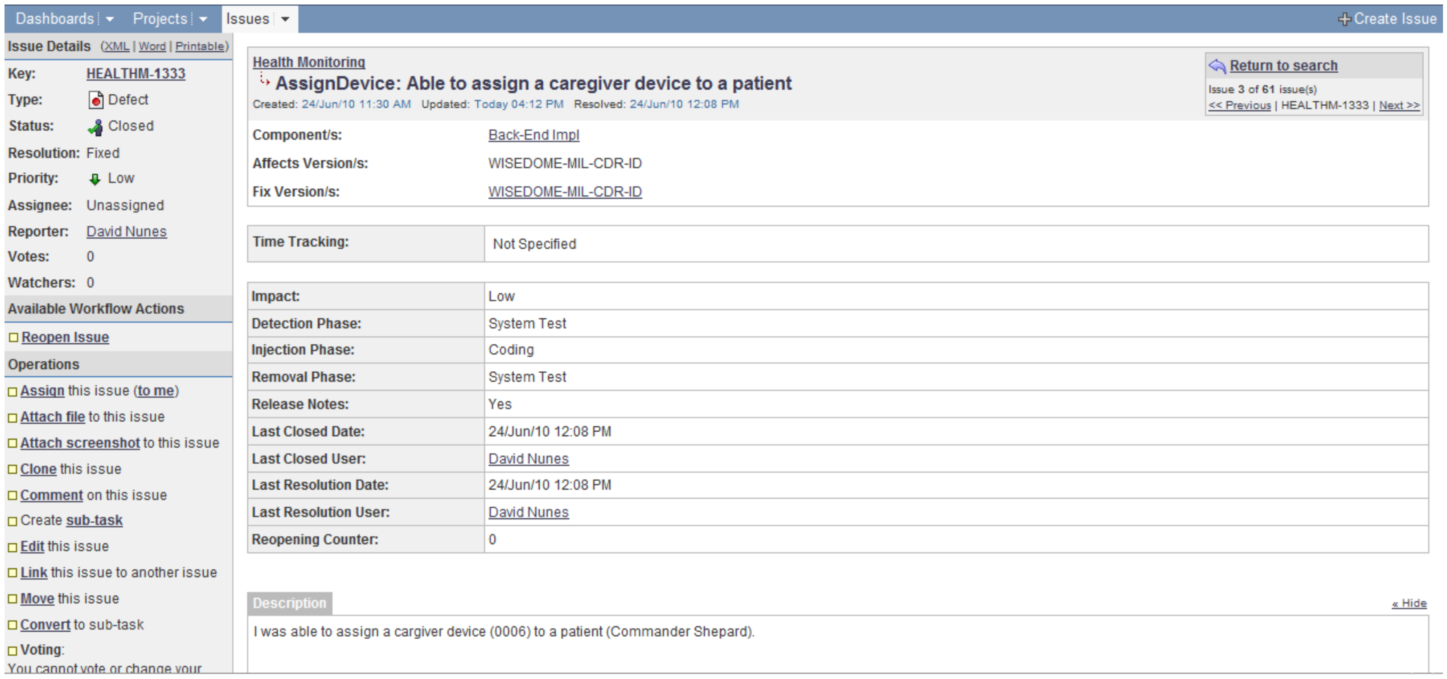
5.2 Software Testing

There was also work performed on the Software Testing Process. Previous to a major Wisedome release, where new features would be included (this release did not include the Integration Framework developed in this project), it was necessary to thoughtfully test the system before implementing it on the client's assisted living facilities. Thus, a month was dedicated to extensively test the system in order to guarantee that it was performing with the desired quality level and in compliance with the specified requirements. The purpose of software testing is to (88):

- Increase the confidence in the quality of the software product.
- Show that the software does what it is supposed to do, and doesn't do what it not supposed to (positive and negative testing).
- Detect software problems as soon as possible, in order to facilitate their resolution.

The tests were performed were driven from the system's requirements and were compiled on a testing matrix, where the test output could be classified as either "Passed" or "Not Passed". Tests ranged from basic functions like being able to login or access a patient's chart, to more complex workflows such as silencing devices or escorting patient's while going out of router range. These were particularly important for the sensor devices, since they allowed to verify if the device's behavior was as expected. Many issues were found during this test phase: device charging mechanism problems, application typos, non-working pop-ups and non-working functionalities are some examples.

The testing process was an iterative process where the testing members would run tests and create issues on an issue tracking software, while development members would fix the found issues. The final result was a much more stable and functional version than the one at the beginning of the testing phase.



Issue Details (XML | Word | Printable)

Key:	HEALTHM-1333
Type:	Defect
Status:	Closed
Resolution:	Fixed
Priority:	Low
Assignee:	Unassigned
Reporter:	David Nunes
Votes:	0
Watchers:	0

Available Workflow Actions

- Reopen Issue

Operations

- Assign this issue (to me)
- Attach file to this issue
- Attach screenshot to this issue
- Clone this issue
- Comment on this issue
- Create sub-task
- Edit this issue
- Link this issue to another issue
- Move this issue
- Convert to sub-task
- Voting:

You cannot vote or change your

Health Monitoring

AssignDevice: Able to assign a caregiver device to a patient

Created: 24/Jun/10 11:30 AM Updated: Today 04:12 PM Resolved: 24/Jun/10 12:08 PM

Return to search

Issue 3 of 61 issue(s)

<< Previous | HEALTHM-1333 | Next >>

Component/s:	Back-End Impl
Affects Version/s:	WISEDOME-MIL-CDR-ID
Fix Version/s:	WISEDOME-MIL-CDR-ID

Time Tracking:	Not Specified

Impact:	Low
Detection Phase:	System Test
Injection Phase:	Coding
Removal Phase:	System Test
Release Notes:	Yes
Last Closed Date:	24/Jun/10 12:08 PM
Last Closed User:	David Nunes
Last Resolution Date:	24/Jun/10 12:08 PM
Last Resolution User:	David Nunes
Reopening Counter:	0

Description « Hide

I was able to assign a cargiver device (0006) to a patient (Commander Shepard).

Figure 33 – Example of an issue detected during the test phases and reported on the issue tracking software JIRA.

The overall testing phase experience was very stimulating, allowing a deeper understanding of the importance of proper system testing and furthering personal relationship with Wisedome's development.

6 Conclusion and Future Work

By the time of this project's conclusion, Wisedome, a platform being developed by Critical Health that enables monitoring of residents in Assisted Living facilities, has already successfully achieved a second major software release, leaving the prototype phase and about to go live at a client's facilities. To this successful release, the efforts made by the team during the test phase were invaluable, efforts that were part of this project's objectives.

Although it has left prototype phase, Wisedome is still being developed and improved. It has now reached a crucial time on its development, where the system's capabilities will be expanded with the introduction of several new types of devices coming from different vendors, which will use different communication technologies and different API's. The developed Integration Framework solution described on this thesis is already being used as a means to facilitate development and integrate the several types of sensor devices onto a single health monitoring solution. The integration efforts with system modularity and expansion in mind will contribute to the Wisedome platform's adoption of new state-of-the-art sensors that include new features which will result in a more flexible and valuable application when compared with other alternatives on the market.

With the conclusion of the health monitoring industry coalitions and wireless technology research, the work on testing that lead to a successful Wisedome's milestone release and development of an already in-use Integration Framework that will most likely become an integrate part of Wisedome's next releases, it can be said that this project has achieved its primary goals which were to develop a health monitoring sensor integration solution and contribute to the development and evolution of the Wisedome application.

6.1 Current Limitations and Future Developments

The Integration Framework is part of the Wisedome system and as Wisedome evolves the Integration Framework must also evolve in order to support the new features and modifications introduced. At the time of this document's writing, the Integration Framework is already being used as an integral part of the Wisedome's system. The *integration branch* mentioned in section 4.5.4.5, page 83, has effectively replaced the *main branch* and new vendor bundles are being developed to support new kinds of hardware vendors. Due to its complex nature and the fact that it effectively tries to serve as a bridge between two different systems (the Wisedome Health Monitoring application and the several hardware devices) it is very possible that the Integration Framework's current implementation still retains some minor bugs, which will probably be discovered in the future, as the framework is used.

Although the current integration effort works, that doesn't mean that it does not have room for future improvements. This section presents some perspective on what could be done to improve the current integration solution and describes some of its current limitations.

6.1.1 Functionality Announcement Feature

Perhaps the most important feature lacking in the current integration implementation is a functionality announcement feature. This feature would surpass most of the limitations of the current implementation and would be very useful to further reduce communication abstraction. The idea is that each vendor bundle announces what kind of features it offers: heart rate measurement, ECG, fall detection, button alarms, localization and so on.

This type of information could be used by the Wisedome bundle on several workflows. A good example would be during the device configuration mentioned in section 4.5.4.4. If each vendor had a way of announcing their supported functionalities, Wisedome would no longer have to assume that chest-band devices support heart rate and bracelets only support button calls. Instead, it could ask the Main Integration Bundle what kind of functionalities does a vendor support and use this information for device configuration or even to determine if it can display heart rate values for a given patient or not. This type of feature would also render the use of the "Not Supported" request status useless, since either Wisedome or the Main Integration Bundle could verify if a vendor supports, for instance, fall detection before trying to issue an enable fall detection command. It would even be possible, after discussing the matter with the Wisedome development team, to simplify the device types to "caregiver" devices and "patient" devices only and remove the distinction between chest-bands and bracelets since it would be now possible to know what kind of features a given device supports.

Implementing this functionality would require some effort in modifying Wisedome, since the application depends on the distinction between bracelets and chest-straps on several workflows. Each one of these workflows would have to be analyzed and discussed in order to discover how to use the functionality announcement feature to its full potential. As an example, we can consider the situation mentioned in page 82 where a dementia patient cannot use chest-straps. It would be necessary to discuss with the development team what device functionalities are patients classified as “dementia” not allowed to use (as opposed to not being able to use chest-straps). It would also be necessary to choose how to implement this function: the announcement of services could be an intrinsic part of the workflow for vendor devices or the Main Integration Bundle could request each vendor bundle for what features does it offer instead. The information on the available features could be stored either in memory or on a table in the database.

6.1.2 New Features Limitation

Another limitation of the current Integration Implementation is how new features implemented in Wisedome can affect all of the vendor bundles. In a situation where a device offers features different than those currently used, which Wisedome will adopt (for example, measuring a person’s body temperature) it is necessary to change Wisedome’s code to support these new features. Since we are adding a new feature to Wisedome (temperature reading) it is also necessary to add new methods to the Vendor Service, so that Wisedome may communicate with the devices that have this functionality. However, changing the Vendor Service interface affects all vendor bundles, because every single one of them implements it. This increases the effort of trying to implement new features, since every device driver has to be updated. However, this limitation is understandable and also manifests itself in other software or hardware applications: if the software or hardware suffers changes, then most of the time drivers also have to be updated in order to conform to the new features.

6.2 Final Appreciation

Working on a commercial Assisted Living solution comes with a sense of fulfillment and with the knowledge that the work developed will contribute to the elderly population’s well-being and to the improvement of elderly care.

It is my understanding that the tasks performed during my internship on Critical Health greatly contributed to my growth as an engineer. I feel that the tasks performed greatly enhanced my knowledge of the Java programming language, my understanding on how wireless technology works and the importance of interoperability between different systems. I also came to understand the importance of quality management and how significant requirement definition and testing are during software development. The time spent on a group as big as Critical Software’s gave me a good insight on what is it like to work in a development team and the challenges associated with the enterprise world. My internship as a part of Wisedome’s team allowed me to acquire technical knowledge and soft-skills that will certainly be useful in future projects that I might participate.

Bibliography

1. Wisedome Quick Guide phase 1. s.l. : Critical Health.

2. **Boucher, Alan, et al.** The Importance of Standards and Its Use in Healthcare. [Online] [Cited: July 24, 2010.] http://www.itsc.org.sg/pdf/5_Healthcare.pdf.

3. *Continua: The Impact of a Personal Telehealth Ecosystem.* **Wartena, Frank, et al.** International Conference on eHealth, Telemedicine, and Social Medicine.

4. About the Alliance. *Continua Health Alliance.* [Online] Continua Health Alliance. [Cited: July 24, 2010.] http://www.continuaalliance.org/about-the-alliance.html.

5. FAQs. *Continua Health Alliance.* [Online] Continua Health Alliance. [Cited: July 25, 2010.] http://www.continuaalliance.org/faqs.html.

6. Mission and Objectives. *Continua Health Alliance.* [Online] Continua Health Alliance. [Cited: July 24, 2010.] http://www.continuaalliance.org/about-the-alliance/mission-and-objectives.html.

7. Design Guideines. *Continua Health Alliance.* [Online] Continua Health Alliance. [Cited: July 24, 2010.] http://www.continuaalliance.org/products/design-guidelines.html.

8. Health & Wellness. *Continua Health Alliance.* [Online] Continua Health Alliance. [Cited: July 25, 2010.] http://www.continuaalliance.org/connected-health-vision/health-and-wellness.html.

9. Continua Use Case Flyers. *Continua Health Alliance.* [Online] Continua Health Alliance. [Cited: July 25, 2010.] http://www.continuaalliance.org/static/binary/cms_workspace/5.04-CH1-CC-401-Continua_flyers.pdf.

10. Chronic Disease Management. *Continua Health Alliance.* [Online] Continua Health Alliance. [Cited: July 25, 2010.] http://www.continuaalliance.org/connected-health-vision/disease-management.html.

11. Continua Certification. [Online] 1.0, Continua Health Alliance, February 23, 2009. [Cited: July 24, 2010.] http://www.continuaalliance.org/static/cms_workspace/Continua_Certification_Public.pdf.

12. Products & Certification Overview. *ZigBee Alliance.* [Online] ZigBee Alliance. [Cited: August 4, 2010.] http://www.zigbee.org/Products/Overview.aspx.

13. **Ramon, San.** ZIGBEE SELECTED BY CONTINUA HEALTH ALLIANCE FOR NEXT GENERATION GUIDELINES. [Online] ZigBee Alliance, June 8, 2009. [Cited: December 18, 2009.] http://zigbee.org/imwp/idms/popups/pop_download.asp?contentID=16015.

14. Bluetooth. *Wikipedia, the free encyclopedia.* [Online] [Cited: August 4, 2010.] http://en.wikipedia.org/wiki/Bluetooth.

15. Continua Overview Presentation. [Online] 19.3, Continua Health Alliance. [Cited: August 1, 2010.] http://www.continuaalliance.org/static/cms_workspace/Continua_Overview_Presentation_v19.3.pdf.

16. ANT+ Connecting Sensors for Life! *This is ANT.* [Online] ANT+ Alliance. [Cited: August 2, 2010.] http://www.thisisant.com/ant/ant-interoperability.

17. Company. *This is ANT.* [Online] ANT+ Alliance. [Cited: August 1, 2010.] http://www.thisisant.com/company.

18. **Horikiri, Chikashi, et al.** Inside 'Nike+iPod' -- Diaphragm is Used as Sensor. *Tech On!* [Online] July 24, 2006. [Cited: August 1, 2010.] http://techon.nikkeibp.co.jp/english/NEWS_EN/20060724/119373/.

19. ANT+ Self Compliance Test Process. *This is ANT.* [Online] ANT+ Alliance. [Cited: August 2, 2010.] http://www.thisisant.com/pages/ant/self-compliance-test-process.

20. Technology. *This is ANT.* [Online] ANT+ Alliance. [Cited: August 2, 2010.] http://www.thisisant.com/technology.

21. ULP – Advantage ANT. [Online] ANT+ Alliance, October 2008. [Cited: August 3, 2010.] http://www.thisisant.com/images/Resources/PDF/ULP-Advantage%20ANT%20Oct%2008.pdf.

22. ANT Q&A's. [Online] ANT+ Alliance, June 2008. [Cited: December 15, 2010.] http://www.thisisant.com/images/Resources/PDF/ant_qandas.pdf.

23. ZigBee and Wireless Frequency Coexistances - ZigBee White Paper. [Online] ZigBee Alliance, June 2007. [Cited: December 15, 2010.] www.zigbee.org/imwp/download.asp?ContentID=11745.

24. **Ergen, Sinem.** ZigBee/IEEE 802.15.4 Summary. [Online] September 10, 2004. [Cited: August 6, 2010.] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.6333&rep=rep1&type=pdf.

25. *6LowPAN.org.* [Online] [Cited: August 3, 2010.] http://www.6lowpan.org.

26. Microchip MiWi P2P Wireless Protocol. *Microchip Technology.* [Online] Microchip Technology Inc., July 2010. [Cited: August 3, 2010.] http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1824&appnote=en536181.

27. **Stevanovic, Dusan.** Zigbee / IEEE 802.15.4 Standard. [Online] June 20, 2007. [Cited: August 5, 2010.] http://www.cse.yorku.ca/~dusan/Zigbee-Standard-Talk.pdf.

28. How ANT Compares. *This is ANT.* [Online] ANT+ Alliance. [Cited: January 10, 2010.] http://www.thisisant.com/why-ant/how-ant-compares.

29. IEEE 802.15.1. [Online] IEEE. [Cited: August 6, 2010.] http://www.ieee802.org/15/pub/TG1a.html.

30. IEEE 802.15. *Wikipedia, the free encyclopedia.* [Online] [Cited: August 7, 2010.] http://en.wikipedia.org/wiki/IEEE_802.15.

31. Bluetooth On The Road. *Hoovers.* [Online] [Cited: August 7, 2010.] http://www.hoovers.com/business-information/--pageid__13751--/global-hoov-index.xhtml.

32. **James, Baldwin.** Know More Knowledge Of Bluetooth. *ArticleSnatch.com.* [Online] [Cited: August 7, 2010.] http://www.articlesnatch.com/Article/Know-More-Knowledge-Of-Bluetooth/889308.

33. **Woodings, Ryan Winfield and Gerrior, Mark.** Avoiding Interference in the 2.4-GHz ISM Band. *EE Times.* [Online] July 1, 2006. [Cited: August 16, 2010.] http://www.eetimes.com/design/automotive-design/4012556/Avoiding-Interference-in-the-2-4-GHz-ISM-Band.

34. **Meyer, David.** Bluetooth 3.0 released without ultrawideband. *ZDNet UK.* [Online] April 22, 2009. [Cited: August 7, 2010.] http://www.zdnet.co.uk/news/networking/2009/04/22/bluetooth-30-released-without-ultrawideband-39643174/.

35. ANT Message Protocol and Usage. [Online] 4.1, ANT+ Alliance. [Cited: August 8, 2010.] http://www.thisisant.com/images/Resources/PDF/1204662412_ant%20message%20protocol%20and%20usage.pdf.

36. RF Power Options in ZigBee™ Solutions - White Paper. [Online] Cirronet, Inc., 2005. [Cited: August 7, 2010.] http://www.cirronet.com/pdf/wp_ZigBeePowerOptions.pdf.

37. **Reggiani, L. and Maggio, G.** Orthogonal convolutional modulation for UWB-impulse radio systems: performance analysis and adaptive schemes. *IEEE Transactions on Wireless Communications.* September 2009, Vol. 8, 9, pp. 4550 - 4560.

38. ZigBee. *John' s Specifications.* [Online] [Cited: August 9, 2010.] http://www.specifications.nl/zigbee/zigbee_UK.php.

39. **Callaway, Ed.** Low Power Consumption Features of the IEEE 802.15.4/ZigBee LR-WPAN Standard. [Online] November 6, 2003. [Cited: December 20, 2009.] http://www.cens.ucla.edu/sensys03/sensys03-callaway.pdf.

40. **Kinney, Patrick.** ZigBee Technology: Wireless Control that Simply Works. [Online] October 2, 2003. [Cited: August 8, 2010.] http://www.zigbee.org/imwp/idms/popups/pop_download.asp?contentID=5162.

41. **Giovino, Bill.** New Atmel Microcontrollers Target Low-Power ZigBee. *microcontroller.com.* [Online] March 20, 2006. [Cited: August 8, 2010.] http://www.microcontroller.com/news/atmel_microcontrollers_avr.asp.

42. **Misal, Chaitanya S.** Analysis of power comsumption of an end device in a ZigBee mesh network. 2007.

43. *Low end extension for Bluetooth.* **Honkanen, M., et al.** Nokia Res. Center, Tampere, Finland : s.n., 2004. Radio and Wireless Conference, 2004 IEEE. pp. 199 - 202.

44. Bluetooth rival unveiled by Nokia. *BBC News.* [Online] October 4, 2006. [Cited: August 10, 2010.] http://news.bbc.co.uk/2/hi/technology/5403564.stm.

45. **Reynolds, Melanie.** Wibree becomes ULP Bluetooth. *ElectronicsWeekly.com.* [Online] June 12, 2007. [Cited: August 10, 2010.] http://www.electronicsweekly.com/Articles/2007/06/12/41582/wibree-becomes-ulp-bluetooth.htm.

46. Bluetooth Low Energy – WiBree. *BuddeBlog.* [Online] April 14, 2009. http://www.buddeblog.com.au/news-and-views/bluetooth-low-energy-wibree/.

47. Bluetooth 4 spec reduces energy consumption by 90%. *TGDaily.* [Online] July 7, 2010. [Cited: August 10, 2010.] http://www.tgdaily.com/mobility-features/50548-bluetooth-4-spec-reduces-energy-consumption-by-90.

48. Bluetooth Low Energy. *Wikipedia, the free encyclopedia.* [Online] [Cited: August 9, 2010.] http://en.wikipedia.org/wiki/Bluetooth_low_energy.

49. Bluetooth Low Energy Technology FAQ. [Online] Bluetooth SIG, December 2009. [Cited: August 10, 2010.]

http://bluetooth.com/SiteCollectionDocuments/Low_Energy_FAQ_External_General_Public.pdf.

50. Bluetooth Low Energy Technology - Technical Info. *Bluetooth.* [Online] Bluetooth SIG. [Cited: August 10, 2010.] http://www.bluetooth.com/English/Products/Pages/Bluetooth_Low_Energy_Technology__Technical_Info.aspx.

51. **Macdonald, Brian.** 2008 ANT+ SYMPOSIUM - The Opportunities. [Online] 2008. [Cited: December 17, 2010.] http://www.thisisant.com/images/Resources/PDF/ANT%2B%20%20The%20Opportunities%20Oct%2008.pdf.

52. —. Building a Practical Wireless Sensor Network. *Nikkei Electronics Asia.* [Online] November 27, 2007. [Cited: December 16, 2009.] http://techon.nikkeibp.co.jp/article/HONSHI/20071127/143123/.

53. **Morris, Rod.** ANT VS Zigbee. [Online] March 31, 2008. [Cited: December 10, 2009.] http://hi.baidu.com/vvfang/blog/item/eef9b1affcbf4ec87dd92a53.html.

54. What's so good about mesh networks? [Online] Daintree Networks, January 2007. [Cited: August 12, 2010.] http://www.daintree.net/downloads/whitepapers/mesh-networking.pdf.

55. *Forming Connected Topologies in Bluetooth Ad-hoc Networks - An Algorithmic Perspective.* **Guerin, R., et al.** 2002. International Teletraffic Congress.

56. Communication Topology. *Bluetooth.* [Online] Bluetooth SIG. [Cited: August 12, 2010.] http://www.bluetooth.com/English/Technology/Works/Pages/Communications_Topology.aspx.

57. *Developing a Standard for Personal Health Devices based on 11073.* **Clarke, M., et al.** 2007. Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE. pp. 6174 - 6176.

58. **Carvalho, Paulo de and Henriques, Jorge.** Telemedicina. Departamento de Engenharia Informática, Universidade de Coimbra : s.n.

59. **Bogia, Douglas.** ISO/IEEE 11073 Personal Health Data Tutorial. [Online] December 21, 2007. [Cited: November 29, 2009.] http://www.lampreynetworks.com/assets/documents/2007-12-21-IEEE-PHD-tutorial.pdf.

60. **Clarke, Malcolm.** ISO/IEEE 11073 Personal Health Devices. [Online] May 26, 2008. [Cited: November 29, 2009.] http://person.hst.aau.dk/ska/MIE2008/ParalleSessions/PresentationsForDownloads/Mon-1530/Sta-30_Clarke.pdf.

61. **Doney, Dallin.** ANT+ Device Profiles. [Online] ANT+ Alliance. [Cited: January 20, 2010.] http://www.thisisant.com/images/Resources/PDF/ANT+%20Device%20Profiles%20Oct%2008.pdf.

62. Certified Products. *Continua Health Alliance.* [Online] Continua Health Alliance. [Cited: August 15, 2010.] http://www.continuaalliance.org/products/certified-products.html.

63. Certified Products Overview. *ZigBee Alliance.* [Online] ZigBee Alliance. [Cited: August 15, 2010.] http://www.zigbee.org/Products/CertifiedProducts/CertifiedProductsOverview.aspx.

64. *Medica.de.* [Online] http://www.medica-tradefair.com/.

65. Leading Health Brands Demonstrate ANT+ Interoperability at MEDICA. *This is ANT.* [Online] ANT+ Alliance, November 19, 2009. [Cited: August 12, 2010.] http://www.thisisant.com/news/stories/leading-health-brands-demonstrate-ant-interoperability-at-medica.

66. Direct-sequence spread spectrum. *Wikipedia, the free encyclopedia.* [Online] [Cited: August 12, 2010.] http://en.wikipedia.org/wiki/Direct-sequence_spread_spectrum.

67. Bluetooth: What´s the advantage of frequency-hopping? *SwedeTrack System.* [Online] Johnson Consulting. [Cited: August 16, 2010.] http://www.swedetrack.com/images/bluet11.htm.

68. *Co-existence of Zigbee and WLAN, A Performance Study.* **Shuaib, K., et al.** 2006. Wireless Telecommunications Symposium. pp. 1 - 6.

69. *Coexistence of IEEE802.15.4 with other Systems in the 2.4 GHz-ISM-Band.* **Sikora, Axel and Groza, Voicu F.** 2005. Instrumentation and Measurement Technology Conference. pp. 1786 - 1791.

70. Nordic Semiconductor previews next generation ANT chips that will redefine industry benchmarks for ultra low power wireless sensoring. *This is ANT.* [Online] ANT+ Alliance, March 25, 2009. [Cited: August 16, 2010.] http://www.thisisant.com/news/stories/nordic-semiconductor-previews-next-generation.

71. **Barry, Douglas K.** Service-oriented architecture (SOA) definition. *Web Services and Service-Oriented Architectures.* [Online] [Cited: July 18, 2010.] http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html.

72. About the OSGi Service Platform - Technical Whitepaper. *OSGi Alliance.* [Online] 4.1, OSGi Alliance, June 7, 2007. [Cited: July 20, 2010.] http://www.osgi.org/wiki/uploads/Links/OSGiTechnicalWhitePaper.pdf.

73. Mobile Market. *OSGi Alliance.* [Online] OSGi Alliance. [Cited: July 21, 2010.] http://www.osgi.org/Markets/Mobile.

74. Siemens Serve@Home home management system. *Appliancist.* [Online] February 23, 2007. [Cited: July 21, 2010.] http://www.appliancist.com/accessories/siemens-home-management-system.html.

75. OSGi. *Eclipse.* [Online] Eclipse Foundation. [Cited: July 21, 2010.] http://www.eclipse.org/osgi/.

76. Enterprise Market. *OSGi Alliance.* [Online] OSGi Alliance. [Cited: July 21, 2010.] http://www.osgi.org/Markets/Enterprise.

77. **Vergara, Jorge E. López de, et al.** An autonomic approach to offer services in OSGi-based home gateways. *Computer Communications.* 2008, Vol. 31, 13, pp. 3049-3058.

78. Apache Felix OSGi Tutorial. *Apache Felix.* [Online] [Cited: March 15, 2010.] http://felix.apache.org/site/apache-felix-osgi-tutorial.html.

79. *Knopflerfish.* [Online] [Cited: July 22, 2010.] http://www.knopflerfish.org/.

80. What Is a Socket? *The Java Tutorials.* [Online] Oracle. [Cited: July 22, 2010.] http://download-llnw.oracle.com/javase/tutorial/networking/sockets/definition.html.

81. What Is an Interface? *The Java Tutorials.* [Online] Oracle. http://download-llnw.oracle.com/javase/tutorial/java/concepts/interface.html.

82. Interfaces. *The Java Tutorials.* [Online] Oracle. http://download-llnw.oracle.com/javase/tutorial/java/IandI/createinterface.html.

83. Device driver. *Wikipedia, the free encyclopedia.* [Online] [Cited: July 23, 2010.] http://en.wikipedia.org/wiki/Device_driver.

84. Interface BundleActivator. *OSGi Service Platform Release 4 Version 4.2.* [Online] OSGi Alliance. [Cited: July 23, 2010.] http://www.osgi.org/javadoc/r4v42/org/osgi/framework/BundleActivator.html.

85. Quality Manual. s.l. : Critical Software, S.A.

86. ENG.1 Requirements Analysis. s.l. : Critical Software S.A.

87. Requirements analysis. *Wikipedia, the free encyclopedia.* [Online] [Cited: August 17, 2010.] http://en.wikipedia.org/wiki/Requirements_analysis.

88. ENG.4 Software Testing Process. s.l. : Critical Software S.A.

89. Product Brief - 1-channel nRF24AP2. [Online] 1.1, Nordic Semiconductor. [Cited: December 16, 2010.] http://www.nordicsemi.com/files/Product/brochures_presentations/PB_nRF24AP2_1ch_v1_1.pdf.

90. **Thonet, Gilles, et al.** ZigBee – WiFi Coexistence - White Paper and Test Report. [Online] Schneider Electric, April 15, 2008. [Cited: January 18, 2010.] http://www.zigbee.org/imwp/idms/popups/pop_download.asp?contentID=13184.

Annex A Documentation

This section documents the ReturnObject class and the Vendor, Database and Process Event services that were introduced by the Integration Framework which are relevant to Vendor Bundle's development. Standard Java notation is used to describe method returns and arguments. This documentation is not meant to be very extensive and avoids going into much detail regarding implementation. It should be viewed as more of a general introduction and assumes that the Vendor Bundle developer has some knowledge of Wisedome's workflows, which are not mentioned on this thesis.

A.I ReturnObject Class

A.I.i Request Status

The request status is used to describe the status of a request made by the Wisedome application.

The class methods that handle the request status are:

- **void setRequestStatus(int requestStatus)** – used to set the request status of a ReturnObject.
- **int getRequestStatus()** – used to get the request status of a ReturnObject.
- **String getRequestString()** – returns a String whose content illustrates the current Request Status of a ReturnObject.
- **boolean isSuccessful() –** this method returns "true" in case the request status has been set to "successful".

The request statuses are defined by integers that are declared within the ReturnObject class. The following request statuses are supported:

- RS_SUCCESSFUL - The Request was handled successfully.

- RS_UnknownID - The PatientID provided is not known.
- RS_DeviceNotReady - The device is not capable of performing the request at this moment.
- RS_InvalidParameter - The command provided is not valid.
- RS_InvalidValue - The argument value provided is not valid.
- RS_NetworkDown – Communication Failure.
- RS_NoDevices - No devices are available.
- RS_InvalidResponse - The gateway / device returned an unexpected response.
- RS_FailedSetupDevice - Failed to perform device setup workflows necessary to satisfy the request.
- RS_NotSupported - the method is not supported by the current vendor.
- RS_InvalidID - the provided ID is known, but not valid at this point.

A.I.ii Return Data

The ReturnObject class supports the returning of following types of data:

- Double – Double is a numerical type of data, supported by a variable belonging to the fundamental Java data type “double”.
- String – A string of text.
- Boolean – A boolean value.
- State – A pre-defined integer that give information about a particular state. The supported states are:
    - STATE_BATTERY_OK – Reports that the device’s battery level is normal.
    - STATE_BATTERY_LOW – Reports that the device’s battery level is low.
    - STATE_BATTERY_DISCHARGED – Reports that the device’s battery is discharged.
    - STATE_BATTERY_CHARGING – Reports that a device in a charging state.
    - STATE_DEVTYPE_CAREGIVER – Reports that the device is a caregiver device

  - STATE_DEVTYPE_CHESTBAND- Reports that the device is a sensor of the type chest-strap.
  - STATE_DEVTYPE_BRACELET – Reports that the device is a sensor of the type bracelet.
  - STATE_EMPTY – This should be used whenever there are no states to report.
- Array – This type of data is used to return an array of other types of data. It is supported by the *java.util.List* class.
- Hashtable – This type of data is used to return a *hashtable* and is supported by the *java.util.Hashtable* class.

Each of the types of data described above is supported by *get* and *set* types of methods, which allow Vendor Bundle developers to manipulate the contents of the ReturnObject. There are also **hasDataType** methods for each data type that allow developers to test if a ReturnObject contains a given data type (**hasDouble** or **hasArray** are examples of these).

A.II Vendor Service

This section describes the Vendor Service interface and explains what is expected from each method that should be implemented. Every method shall return a ReturnObject with the appropriate request status. In case a method is not supported by a given Vendor Bundle, the request status shall be returned as “RS_NotSupported”. In case a Vendor Bundle is incapable of maintaining the state of the currently assigned patient for every device, it should use the Database service to seek that information.

A.II.i ReturnObject assignDevice(int personID, boolean isPatient)

This method handles device assignment. A device is assigned to a person with the ID personID, which is a patient if the Boolean “isPatient” is true or a caregiver if it is false. An assigned device should be active and ready to sent events to Wisedome.

This method shall return a ReturnObject containing a String data type with the serial of the device that was assigned to the person, which should match the one's defined in the sensor's table of Wisedome's database.

A.II.ii ReturnObject unassignDevice(int personID, boolean isPatient)

This method handles device un-assignment. An unassigned device shall not send any events to Wisedome. The arguments personID and isPatient perform similar roles to the ones described in the method "assignDevice". This method's ReturnObject does not need to return any kind of data.

A.II.iii ReturnObject isDeviceMuted(int personID)

This method returns a ReturnObject with Boolean type of data. In case a device assigned to the person with the ID "personID" has its sound disabled, the Boolean should be set as "true". Otherwise, the Boolean shall be set as "false".

A.II.iv ReturnObject getHrMin(int personID)

This method shall return a ReturnObject with numerical type of data, containing the current setting for the minimum permitted heart rate for the device assigned to the patient with the ID "personID".

A.II.v ReturnObject getHrMax(int personID)

This method shall return a ReturnObject with numerical type of data, containing the current setting for the maximum permitted heart rate for the device assigned to the patient with the ID "personID".

A.II.vi ReturnObject getFallThreshold(int personID)

This method shall return a ReturnObject with numerical type of data, containing the current setting for the fall threshold for the device assigned to the patient with the ID “personID”.

It returns an integer between 0 (Low) and 40 (High). The normal values are:

0 → Low

0 → Medium

40 → High

It is possible to return any integer value between 0 and 40. The Wisedome application should be able to handle values in-between.

A.II.vii ReturnObject isButtonAlarmEnabled(int personID)

This method returns a ReturnObject with Boolean type of data. In case a device assigned to the person with the ID “personID” has been set to be capable of raising Button alarms, the Boolean should be set as “true”. Otherwise, the Boolean shall be set as “false”.

A.II.viii ReturnObject isHrAlarmEnabled(int personID)

This method returns a ReturnObject with Boolean type of data. In case a device assigned to the person with the ID “personID” has been set to be capable of raising Heat Rate alarms, the Boolean should be set as “true”. Otherwise, the Boolean shall be set as “false”.

A.II.ix ReturnObject isFallAlarmEnabled(int personID)

This method returns a ReturnObject with Boolean type of data. In case a device assigned to the person with the ID “personID” has been set to be capable of raising fall alarms, the Boolean should be set as “true”. Otherwise, the Boolean shall be set as “false”.

A.II.x ReturnObject getPendingAlarms(int personID)

This method shall return a ReturnObject with numerical type of data, the number of pending alarms that are set on a caregiver device assigned to the person with the ID “personID”.

A.II.xi ReturnObject muteDevice(int personID, boolean muteDevice)

If the Boolean muteDevice is set to be true, this method should silence the device assigned to the person with the ID “personID”. Otherwise, the method should un-silence the device. This method is not required to return any type of data.

A.II.xii ReturnObject setHrMin(int personID, int hrMin)

This method should set the minimum permitted heart rate on the device assigned to the patient with the ID “personID” to the value of the argument” hrMin”. This method is not required to return any type of data.

A.II.xiii ReturnObject setHrMax(int personID, int hrMax)

This method should set the maximum permitted heart rate on the device assigned to the patient with the ID “personID” to the value of the argument” hrMax”. This method is not required to return any type of data.

A.II.xiv ReturnObject setFallThreshold(int personID, int fdTh)

This method should set the maximum permitted fall threshold on the device assigned to the patient with the ID “personID” to a setting equivalent to the value of the argument ”fdTh”. The normal values are:

0 → Low

20 → Medium

40 → High

It is possible to set any integer value between 0 and 40. The Vendor Bundle application should be able to handle values in-between. This method is not required to return any type of data.

A.II.xv ReturnObject setHrAlarmEnabled(int personID, boolean hrAlarm)

If the Boolean “hrAlarm” is set to be true, this method should enable heart rate alarms to be sent by the device assigned to the patient with the ID “personID”. Otherwise, the method should unable the device to send such alarms. This method is not required to return any type of data.

A.II.xvi ReturnObject setFallAlarmEnabled(int personID, boolean fdAlarm)

If the Boolean "fdAlarm" is set to be true, this method should enable fall alarms to be sent by the device assigned to the patient with the ID "personID". Otherwise, the method should unable the device to send such alarms. This method is not required to return any type of data.

A.II.xvii ReturnObject setPendingAlarms(int personID, int pendingAlarms)

This method should set the number of alarms pending on a caregiver device assigned to the person with the ID "personID" to the value of the argument" pendingAlarms". This method is not required to return any type of data.

A.II.xviii ReturnObject requestHeartRate(int personID)

This method is used by Wisedome to request the current Heart Rate measured by the device assigned to the patient with the ID "personID". The heart rate value shall be returned through a ProcessEvent service method: **ProcessHeartRate (int ID, int value, Date timestamp)**.

A.II.xix ReturnObject getBatteryState(int personID)

This method is used by Wisedome to request the current Battery level of the device assigned to the person with the ID "personID". It should return a ReturnObject with the appropriate state, as defined in the ReturnObject section.

A.II.xx ReturnObject getDeviceSensorSerial(int personID)

This method is used by Wisedome to request the serial of the device assigned to the person with the ID “personID”. It should return a ReturnObject with a string containing the serial, in accordance to the device information stored in Wisedome’s database.

A.II.xxi ReturnObject getDeviceType(int personID)

This method is used by Wisedome to request the type of the device assigned to the person with the ID “personID”. It should return a ReturnObject with the appropriate state, as defined in the ReturnObject section.

A.II.xxii ReturnObject isDeviceLocked(int personID)

This method is used by Wisedome to query the state of the lock mechanism of the device assigned to the person with the ID “personID”. It should return a ReturnObject with a Boolean, which value is “true” if the device is locked and “false” if otherwise.

A.II.xxiii LocateDeviceRouterSerial(int personID)

This method should return a ReturnObject with a string containing the serial of the router to which a given device is currently paired with. The serial should be in accord with the information on Wisedome’s database.

A.II.xxiv ReturnObject acknowledgeAlarm(int personID)

Wisedome uses this method to acknowledge all alarms currently active on the device assigned to the patient with the ID “personID”. It is not required to return any type of data.

A.II.xxv ReturnObject cancelAlarm(int personID)

Wisedome uses this method to cancel all alarms currently active on the device assigned to the patient with the ID “personID”. It is not required to return any type of data.

A.II.xxvi ReturnObject getRouterState(String routerSerial)

This method is used to know if the router with the serial “routerSerial” is currently communicable or not. It should return a ReturnObject with a Boolean set to “true” in case the router is communicable and “false” if otherwise.

A.II.xxvii ReturnObject suspendDevice(int personID)

This method shall suspend the device assigned to the person with the ID “personID”. A suspended device shall not raise any alarms but should continue to be assigned.

A.II.xxviii ReturnObject resumeDevice(int personID)

This method shall resume a previously suspended device assigned to the person with the ID “personID”.

A.II.xxix ReturnObject startECGStream(int personID)

This method is used to start acquiring an ECG stream for a patient, and store the values in memory. The acquired values are returned through the method **getECGStream**. Wisedome may not make multiple calls for a **startECGStream**, for the same patient: calling the **startECGStream** more than once without calling the **stopECGStream** in between, will cause the method to return a ReturnObject with InvalidID request status. A *hashtable* shall be used to associate each ECG value with a timestamp.

A.II.xxx ReturnObject stopECGStream(int personID)

This method is used to inform the Integration driver to stop acquiring and storing ECG values from the device assigned to the person with the ID “personID”. Calling this method will return all the values available in memory for the provided patient, through a ReturnObject returning a *hashtable* that associates each value with a timestamp. Invoking this method without previously calling **startECGStream** will return an InvalidID request status.

A.II.xxxi ReturnObject getECGStream(int personID)

This method is used to retrieve the ECG values currently stored in memory after calling the **startECGStream** method. Values are returned through a ReturnObject containing a hashtable that associates each value with a timestamp. Invoking this method without previously invoking the method **startECGStream** will result in a ReturnObject with InvalidID request status.

A.II.xxxii ReturnObject getECGValues(int personID, int numberOfValues)

This method returns a ReturnObject with a List containing a total of "*numberOfValues*" ECG values for a given patient with an ID "personID". It's possible to invoke this method after calling startECGStream but it is not necessary since they are independent methods. If this method is called while there is a request for ECG Stream values (initiated with a startECGStream method), the ECG Stream will contain the values returned by this method, but associated with a timestamp.

A.III Service Provider

The service provider merely offers two methods, **getDatabaseService** and **getProcessEventService**, which should be used to retrieve the Database and Process Event Services.

A.IV Database Service

A.IV.i String getConfigFilePath()

This method returns the system path where the configuration files are stored, and should be used to retrieve the Vendor Bundle's configuration file.

A.IV.ii RetPatientSettings personImp_getPatientSettings(int personID)

This method returns a RetPatientSettings object, exported by the Wisedome_patients bundle, which contains relevant patient information stored in the Wisedome's database. The object contains information like the currently set fall sensibility and heart rate values.

A.IV.iii int personImp_getUserType(int personID)

This method returns an integer that indicates the type of the user with the ID personID. The integer might identify an independent, dementia or dependent patient, as well as a nurse, TAP or administrator. The possible values of the returned integer are defined in the *PersonsImplementation* class of the Wisedome Bundle.

A.IV.iv String getVendorName(int vendorType)

This is used to retrieve a String containing the Vendor Name correspondent to the provided vendorType, as defined in Wisedome's database.

A.IV.v List<Integer> getSensorPersonsID (String sensorSerial)

This method returns a list with the ID's of the persons associated with the device that has the serial "sensorSerial".

A.IV.vi String getPersonSensorVendor(int personID)

This method returns a String containing the Vendor Name for the vendor of the device associated to the person that possesses the ID "personID".

A.IV.vii String getPersonSensorSerial(int personID)

This returns a String containing the Serial of the device currently associated to the person with the ID "personID".

A.IV.viii void changeRouterState(String routerSerial, boolean state)

This method is used to directly change the database state of the router with the serial "routerSerial". The Boolean "state" is used to define if the router is communicable (true), or if the router is incommunicable (false).

A.IV.ix void createVendorDownEvent(String vendor)

Invoking this method will create a "vendor down" event on Wisedome's database. This is used by the Main Integration Bundle to inform Wisedome that a vendor that was supposed to auto-start did not initiate properly, or that a vendor that was available suddenly became unavailable. The String "vendor" corresponds to the Vendor Name.

A.IV.x void silenceVendorDownEvent(String vendor)

This method should be invoked to "clean" a previously created "vendor down" event, if the vendor in question has become available. The String "vendor" corresponds to the Vendor Name.

A.IV.xi boolean isPersonDeviceMuted(int personID)

This method checks the database for the current muted status of a device associated with the person with the ID "personID". The Boolean returned has the value "true" in case the device is muted or "false" if the device is not muted. Please note that this is the value existent on Wisedome's database and might not be in sync with the actual state of the device.

A.IV.xii boolean isPersonDeviceSuspended(int personID)

This method checks the database and returns if the device associated with the person with the ID “personID” is suspended or not. The Boolean returned has the value “true” in case the device is suspended or “false” if the device is active. Please note that this is the value existent on Wisedome’s database and might not be in sync with the actual state of the device.

A.IV.xiii public boolean isPersonDeviceCharging(int personID)

This method checks the database and returns if the device associated with the person with the ID “personID” is currently charging or not. The Boolean returned has the value “true” in case the device is charging or “false” if the device is not charging. Please note that this is the value existent on Wisedome’s database and might not be in sync with the actual state of the device.

A.IV.xiv boolean isPersonDeviceDischarged(int personID)

This method checks the database and returns if the battery of the device associated with the person with the ID “personID” is discharged or not. The Boolean returned has the value “true” in case the battery is discharged or “false” if the battery is not discharged. Please note that this is the value existent on Wisedome’s database and might not be in sync with the actual state of the battery.

A.IV.xv boolean isPersonDeviceUnlocked(int personID)

This method checks the database and returns if the device associated with the person with the ID “personID” has its locking mechanism opened or not. The Boolean returned has the value “true” in case the locking mechanism is opened or “false” if the locking mechanism is closed. Please note that this is the value existent on Wisedome's database and might not be in sync with the actual state of the locking mechanism.

A.V Process Event Service

A.V.i void ProcessVendorStateChange(List<String> vendorlist)

This method is used by the Main Integration Bundle, in order to update the list of available vendors to Wisedome.

A.V.ii void ProcessBatteryStateChange(int ID, int stateValue, Date timestamp)

This method should be used whenever the state of the battery of a device changes. The device shall be identified by its associated person's ID, "ID" and the state of the battery shall be described by an integer, "stateValue", defined in the *ProcessEventService* interface. The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.iii void ProcessLockedStateChange(int ID, boolean isLocked, Date timestamp)

This method should be used whenever the state of the locking mechanism of a device changes. The device shall be identified by its associated person's ID, "ID" and the state of the locking mechanism shall be described by a Boolean, "isLocked", that should have the value "true" if the device has been locked and false if the device has been unlocked. The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.iv void ProcessUnreachableStateChange(int ID, boolean isUnreachable, Date timestamp)

This method should be used whenever the state of the network availability of a device changes. The device shall be identified by its associated person's ID, "ID" and the state of the network avaiability shall be described by a Boolean, "isUnreachable", that should have the value "true" if the device became not communicable and false if the device became communicable. The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.v void ProcessUserDeviceReplacement(int ID, String newDeviceID, Date timestamp)

This method shall be used to notify Wisedome whenever a caregiver tries to change a person's currently associated device to a new one, by performing a specific button sequence. The device shall be identified by its associated person's ID, "ID" and serial of the new device shall be passed on as a String, "newDeviceID". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.vi void ProcessRouterStateChange(String ID, boolean routerState, Date timestamp)

This method should be used whenever the state of a network router changes. The router shall be identified by its serial, in the String "ID", and the state of the router availability shall be described by a Boolean, "routerState", that should have the value "true" if the router is active and communicable and false if the router became is inactive or incommunicable. The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.vii void ProcessAlarmEventButton(int ID, String routerSerial, Date timestamp)

This method is used to report a button alarm, generated by a sensor device, to Wisedome. The device shall be identified by its associated person's ID, "ID" and the router the device is currently communicating with shall be identified by its serial, in the String "routerSerial". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.viii void ProcessAlarmEventFall(int ID, String routerSerial, Date timestamp)

This method is used to report a fall detection alarm, generated by a sensor device, to Wisedome. The device shall be identified by its associated person's ID, "ID" and the router the device is currently communicating with shall be identified by its serial, in the String "routerSerial". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.ix void ProcessAlarmEventHR(int ID, String routerSerial, Date timestamp, int value)

This method is used to report a heart rate alarm, generated by a sensor device, to Wisedome. The device shall be identified by its associated person's ID, "ID" and the router the device is currently communicating with shall be identified by its serial, in the String "routerSerial". The heart rate value that was in the origin of the alarm shall be passed as an integer, "value". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.x void ProcessAlarmEventLocation(int ID, String routerSerial, Date timestamp)

This method is used to report a location alarm, generated by a sensor device, to Wisedome. The device shall be identified by its associated person's ID, "ID" and the router the device is currently communicating with shall be identified by its serial, in the String "routerSerial". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.xi void ProcessAlarmEventCancel(int ID, String routerSerial, Date timestamp)

This method is used to report a cancelation of an alarm to Wisedome. The device in question shall be identified by its associated person's ID, "ID" and the router the device is currently communicating with shall be identified by its serial, in the String "routerSerial". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.xii boolean ProcessAlarmAckRequest(int patientID, int careTakerID, Date timestamp)

This method is used to report an alarm acknowledgement request to Wisedome. The device in question shall be identified by its associated person's ID, "ID" and the router the device is currently communicating with shall be identified by its serial, in the String "routerSerial". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object. In case the processing of the acknowledgement is successful, a Boolean is returned with the value "true", otherwise, the Boolean returned has the value "false".

A.V.xiii void ProcessDeviceRouterChange(int ID, String routerSerial, Date timestamp)

This method should be used whenever a device changes the router that is currently using for communication. The device shall be identified by its associated person's ID, "ID" and the router the device is now using shall be identified by its serial, in the String "routerSerial". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.xiv public void ProcessHeartRate(int ID, int value, Date timestamp)

This method is used to report a heart rate value to Wisedome. This is true for both heart rate values requested by the requestHeartRate method and periodic values. The device that measured the heart rate shall be identified by its associated person's ID, "ID" and the heart rate value shall be passed on as an integer, "value". The event shall also be accompanied by a timestamp, which is comprised of a *java.util.Date* object.

A.V.xv void ProcessSuspendDevice(int patientID, int careTakerID)

This method is used to report a device suspension request to Wisedome. The device to be suspended shall be identified by its associated person's ID, "ID" and the caregiver device in the origin of the request shall be identified by its associated person's ID, "careTakerID".

A.V.xvi void ProcessResumeDevice(int patientID, int careTakerID)

This method is used to report a device resume request to Wisedome. The device to be resumed shall be identified by its associated person's ID, "ID" and the caregiver device in the origin of the request shall be identified by its associated person's ID, "careTakerID".

Annex B Emails

B.I Exchanged Email Regarding ANT Device Profiles

B.I.i Email Sent to ANT

From: David Nunes

Sent: December 4, 2009 7:32 AM

Subject: Questions about ANT+ Profiles

Greetings,

My name is David Nunes, and I am researching ANT wireless technology as a possible integration solution for a HealthCare Monitoring solution.

So far, I have found ANT a very interesting solution, but I would like to know more about the device profiles specified by the ANT+ Alliance. By researching your website, I have found that currently, ANT+ has defined profiles that cover Heart Rate Monitors, Temperatures sensors, Weight Scale and others.I am interested in profiles that could prove usefull in a HealthCare monitoring solution. Is there any work being done in profiles regarding ECG monitors and Fall detection.

I would also like to know how interoperability between sensors is achieved. How well defined the data payload between the devices is? Do these profiles guarantee interoperability at the same level as other possible solutions, such as ZigBee with IEEE 11073?

Finally, I would be interested in knowing if there are any sensor suppliers within the ANT+ Alliance that use common health device integration standards such as the IEEE 11073-

20601-2008 -Personal health device communication, implemented using an ANT+ Wireless solution.

Thank you for your time,

David Nunes

B.I.ii Response from ANT

Hello David,

Thank you for your interest in ANT technology and the ANT+ Managed Network as solutions to your Health Care Monitoring solution.

There are many device profiles that are defined by ANT+ for health sensors as well as some that are in development, these are:

Heart Rate – Gives BPM and R-R values, ECG is in early development at the moment

Temperature Sensor – Gives temperature and other environmental parameters

Weight Scale – This includes body composition parameters

Blood Pressure – Completing development soon

Activity Monitor – In active development

Blood Glucose – In early development

The device profiles that are created by the ANT+ Design Team are driven by the needs of the members of the ANT+ Alliance. So if you were to join the ANT+ Alliance and required a profile to be created that did not currently exist, a process could be started to create the necessary profile to help you meet your goals.

For fall detection specifically, we have a member of the ANT+ Alliance that has created a fall detection sensor and uses ANT as its radio technology. They are in the process of commercializing their product and we will create a profile for this sensor when they are ready for commercial implementation.

In answer to your question "I would also like to know how interoperability between sensors is achieved": The ANT+ device profiles define how sensors operate, the topology, channelization, and data payload of the wireless connection is for each specific sensor. This allows collection devices (PDA, cell phones, watches, laptops, etc.) to communicate with these devices by adhering to the profiles regardless of the manufacturer of the specific sensor.

All of the ANT+ Device Profiles that deal with health and related data sets have been carefully developed to ensure that the necessary data fields are present to allow for the collection devices to map the data from the optimized ANT+ format into possible other XHR formats, with specific reference to 11073-20601 compatibility. This relieves the sensor manufacturers from the additional overhead of the 11073 standard which cannot be run on an ULP (Ultra Low Power) device, but maintains compatibility at the collection device level.

I hope that this answers your immediate questions. I have attached the ANT+ license agreements for your convenience. There are two licenses, the Product Developer License gives you access to the ANT+ Device profiles and other documentation necessary for device development. The Commercial Product License provides the proper branding and other guidelines to allow for commercially deployed products.

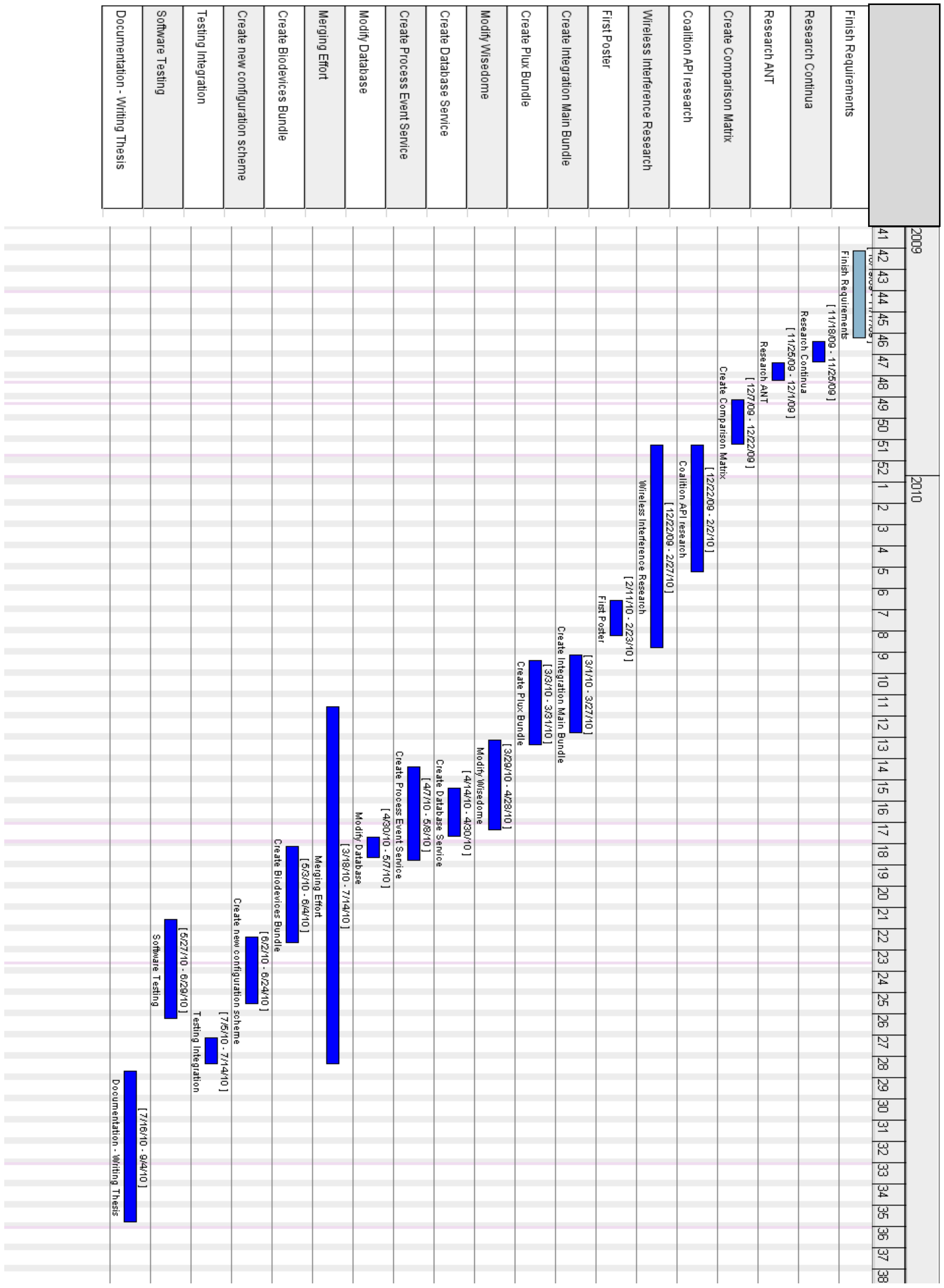
If you have any further questions about ANT, ANT+, or the licensing, please do not hesitate to contact me.

Thanks,

Dallin Doney

Business Development Manager – Wireless

Annex C Project's Gantt Diagram


Finish Requirements
Research Continua
Research ANT
Create Comparison Matrix
Coalition API research
Wireless Interference Research
First Poster
Create Integration Main Bundle
Create Plux Bundle
Modify Wisedome
Create Database Service
Create Process Event Service
Modify Database
Merging Effort
Create Biodevices Bundle
Create new configuration scheme
Testing Integration
Software Testing
Documentation - Writing Thesis
Finish Requirements [11/18/09 - 11/25/09]
Research Continua [11/25/09 - 12/1/09]
Research ANT [12/7/09 - 12/22/09]
Create Comparison Matrix [12/22/09 - 2/2/10]
Coalition API research [12/22/09 - 2/27/10]
Wireless Interference Research [2/11/10 - 2/23/10]
First Poster [3/1/10 - 3/27/10]
Create Integration Main Bundle [3/3/10 - 3/31/10]
Create Plux Bundle [3/29/10 - 4/28/10]
Modify Wisedome [4/14/10 - 4/30/10]
Create Database Service [4/7/10 - 5/8/10]
Create Process Event Service [4/30/10 - 5/7/10]
Modify Database [3/18/10 - 7/14/10]
Merging Effort [5/3/10 - 6/4/10]
Create Biodevices Bundle [6/2/10 - 6/24/10]
Create new configuration scheme [7/5/10 - 7/14/10]
Testing Integration
Software Testing [5/27/10 - 6/29/10]
Documentation - Writing Thesis [7/16/10 - 9/4/10]