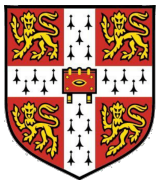
Abstracting Information on Body Area Networks

Pedro Brandão



University of Cambridge
Computer Laboratory
Magdalene College

July 2011

This dissertation is submitted for
the degree of Doctor of Philosophy

Declaration

The dissertation is not substantially the same as any I have submitted for a degree or diploma or any other qualification at any other university. Further, no part of the dissertation has already been or is being concurrently submitted for any such degree, diploma or other qualification.

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation does not exceed the regulation length of $60\,000$ words, including figures, tables and footnotes.

Throughout the text the first person of the plural is used as a matter of linguistic style.

This work was partially supported by Fundação da Ciência e Tecnologia from Portugal.

Pedro Brandão, July 2011

Abstract

Healthcare is changing, correction... healthcare is in need of change. The population ageing, the increase in chronic and heart diseases and just the increase in population size will overwhelm the current hospital-centric healthcare.

There is a growing interest by individuals to monitor their own physiology. Not only for sport activities, but also to control their own diseases. They are changing from the passive *healthcare receiver* to a proactive *self-healthcare taker*. The focus is shifting from hospital centred treatment to a patient-centric healthcare monitoring.

Continuous, everyday, wearable monitoring and actuating is part of this change. In this setting, sensors that monitor the heart, blood pressure, movement, brain activity, dopamine levels, and actuators that pump insulin, "pump" the heart, deliver drugs to specific organs, stimulate the brain are needed as pervasive components in and on the body. They will tend for people's need of self-monitoring and facilitate healthcare delivery.

These components around a human body that communicate to sense and act in a coordinated fashion make a Body Area Network (BAN). In most cases, and in our view, a central, more powerful component will act as the coordinator of this network. These networks aim to augment the power to monitor the human body and react to problems discovered with this observation. One key advantage of this system is their overarching view of the whole network. That is, the central component can have an understanding of all the monitored signals and correlate them to better evaluate and react to problems. This is the focus of our thesis.

In this document we argue that this multi-parameter correlation of the heterogeneous sensed information is not being handled in BANs. The current view depends exclusively on the application that is using the network and its understanding of the parameters. This means that every application will oversee the BAN's heterogeneous resources managing them directly without taking into consideration other applications, their needs and knowledge.

There are several physiological correlations already known by the medical field. Correlating blood pressure and cross sectional area of blood vessels to calculate blood velocity, estimating oxygen delivery from cardiac output and oxygen saturation, are such examples. This knowledge should be available in a BAN and shared by the several applications that make use of the network. This architecture implies a central component that manages the knowledge and the resources. And this is, in our view, missing in BANs.

Our proposal is a middleware layer that abstracts the underlying BAN's resources to the application, providing instead an information model to be queried. The model describes the correlations for producing new information that the middleware knows about. Naturally, the raw sensed data

is also part of the model. The middleware hides the specificities of the nodes that constitute the BAN, by making available their *sensed production*. Applications are able to query for information attaching requirements to these requests. The middleware is then responsible for satisfying the requests while optimising the resource usage of the BAN.

Our architecture proposal is divided in two corresponding layers, one that abstracts the nodes' hardware (hiding node's particularities) and the information layer that describes information available and how it is correlated. A prototype implementation of the architecture was done to illustrate the concept.

Acknowledgements

First I would like to thank the reason for all (not just the thesis, really *all*): my family. My wife Sara and my two kids Inês and Daniel were the ones that stood the most and supported me the most. They make it *all* worthwhile. My mother bared a lot of the burden (starting by having me) and gave me (and still gives) a lot of strength to keep going. My father also tried to put the pressure, ringing and saying "Well, when?". My mother and father in law were also of invaluable help. They made things seem a lot easier than they were, with their tireless support on looking after all of us.

A deep thanks to my supervisor Prof. Jean Bacon, first for taking me in into a rewarding experience. Secondly for providing all the support so that I would carry through this research trip. I am in debt also to Dr. Robert Harle, my advisor, for trying to guide me in my research path.

Words are not enough for expressing all the thanks that are due to the people from the Opera group. First words go to Sriram Srinivasan for being a guide, a realistic-optimist, a pushing-force, a café lover and all-knowledgeable-internet-savvy. Thank you for all the feedback and plain café talks. Eiko Yoneki's driving force of nature helped me through some of the doubts and hesitations of the PhD path, with the help of teas and cookies accompanied by her insight in the research field. Dave Eyers provided much of the call to reality and guidance through this "devious" way, with his expertise on all things techie and most of the non-techie. David Evans' knowledge and experience that he puts through his constructive and insightful chats were also fundamental for getting it through. Thank you, also to Jat Singh for providing me with incentive and feedback, using his fresh experience on the talks we had. Although it wasn't for long enough, Samuel Kounev contributed with his organization, good mood and well thought opinions. Salman Taherian was a good office mate, while it lasted, providing useful and relaxing discussions.

I owe all my friends a lot of gratitude for always being there, even when I was away.
Thank you all.

Contents

List of Figures

List of Tables

1 Introduction

Our work revolves around defining a middleware for Body Area Networks that provides multi-parameter correlations for applications monitoring the human body. Our proposal enables using models that describe correlations between sensed information to produce/infer new information. Applications are relieved from doing these correlations, needing only to request the required information. The middleware also provides a hardware abstraction of the underlying resources.

In this introduction we describe the current *ground* for health care and the direction we propose as the way forward. Our contribution to this path is presented at the end.

1.1 – Where are we?

The motivation for our work rises from the increasing health monitoring needs for the population and the self-awareness that people want to have of their physical activity/exercise. These, especially the first, are the main motives for the growth of sensing around and within our body. This need is spurred by several factors. One is the **increase in chronic diseases**. A report from Partnership for Solutions National Program Office (in 2004) [3] and a World Health Organization (WHO) report [132] show the increase in the number of affected people. Another factor is **population ageing**. WHO [131] states that in 2008 the average life expectancy for the global population was 68 years with 80+ years in some developed countries (most European countries, Canada, Australia, Japan, New Zealand, etc.). And the last point relates to global projections of an **increase in deaths related to cancer and heart diseases** [129, page 25]. Other figures from WHO [134] point to high Blood Pressure (BP) (leading cause), high blood glucose and physical inactivity as three of the four main attributable causes of mortality; tobacco is the second. In the same report, high BP and high blood glucose were in the top ten causes

of an increase in Disability Adjusted Life Year (DALY)[1]. Another fact is the population growth and especially the expected decrease in health-worker to patient ratio. From WHO data up to 2009 [133], the ratio is still being maintained or slightly increasing, but the ten-year work plan from 2006 of WHO [130] includes to "*sustain [a] high performing workforce*".

The current healthcare approach is based on a model where the patient, or when followed by a physician, the doctor, first detects symptoms that then lead to a search for a diagnostic. Treatment then follows, which may involve hospitalization. As described by Gupta [39] this is a "*manual, slow, costly and inefficient*" process. As portrayed by Andy Grove's 2003 *main-frame* metaphor [98], the system is based on a centralized model: "*expectation of society is that everybody should have the right to have access to this "mainframe"*". The focus is on the *central components*: the physicians and healthcare units. This model is being overloaded with demand, and costs will become unsustainable with the current trend. The same point is also defended by Bardram [8] where he adds that current developments (in 2008) seemed still driven by/to the centralized model, where the research was directed to the *main-frame* parts (Information Technology (IT), information processing, storage, etc.) as opposed to the patients themselves. He postulates that a change is needed to a pervasive model. The characteristics need to evolve from the centralized model to a pervasive model, from Bardram [8]:

$$Acute \rightarrow Continuous$$

$$Hospitalization \rightarrow Home\ \&\ out\text{-}patient$$

$$Reactive \rightarrow Pro\text{-}active\ \&\ Preventive$$

$$IT \rightarrow Assisting\ Technology$$

$$Centralized \rightarrow Pervasive$$

$$Sampling \rightarrow Monitoring$$

$$Doctor\text{-}centric \rightarrow Patient\text{-}centric$$

Which leads us to the next section.

1.2 – Where do we want to go?

The ailments from the previous section: chronic diseases, mortality and morbidity associated diseases, ageing population and the decrease in population health coverage can all be abated by a continuous monitoring environment, as is shown by the several studies on monitoring glucose for diabetes and heart rate for patients with cardiovascular diseases[2]. One could also argue that for sport practitioners providing wearable, unobtrusive, easy to use and more precise monitoring

[1] DALY is a measure used by WHO of the impact of disease in a person's life. It measures the productive years lost due to an illness, disability or premature death.

[2] Regarding cardiac monitoring there are studies dating back to 1977, where Brodsky et al. [17] evaluate portable monitoring. A group of experts in this area as produced a document with guidelines for monitoring in Implantable Electronic Cardiovascular Devices (IECDs) [128], in the document's table 1 they define the objectives of monitoring divided by patient, IECD, disease and communication objectives. A list of studies and devices can be found in Burri and Senouf's work [18].

conditions could spark exercise. This exercise motivation should range several age groups, with especial relevance for the elderly population that would gain confidence to practice from this uninterrupted observation. However, the more relevant argument is that there is a need to change from a traditional healthcare, which reacts to a detected malady, to a pervasive healthcare, where continuous monitoring enables an early detection of a disease and allows for proactive action as defended by Gupta, Kulkarni and Öztürk [39, 61].

This need has motivated technological developments associated with healthcare and sensing and sparked market interest. PricewaterhouseCoopers [90] made a score card of medical innovation in nine countries to assess the USA's position in the *race for global leadership*. The report showed an expenditure increase on health in those countries. MobileHealthNews [28] reports on investors putting in $233 M on Mobile Health. Forrester Research [15] sees that after a struggling start the *healthcare unbound*[3] market will *sky-rocket to $34 billion by 2015*. Products are already available for monitoring several parameters like ElectroCardioGram (ECG), BP, oxygen saturation, multi-parameter devices, defibrillators, health hubs and *controlled* pills. And research is being done to enhance them in terms of sensing capabilities[4].

Applications will be needed to handle a myriad of information: monitor, analyse, control, warn, store, etc. But given all the different ailments and objectives (including sports), one application will not necessarily fit all purposes. A simple example would be self-assessment for a jogging activity while being monitored for diabetes. However, there can be more complex environments: where one is being monitored for more than one disease; where the user wants to assess different parameters in his activity (sleep patterns, calories spent, calories intake, etc.). This prompts the need to cope with several applications accessing the data sources.

As introduced in the last section the global idea is to move from the incumbent model of the hospital-centred treatment to a pervasive model where the patient is the focus of the healthcare. As put by Andy Grove [98] in the *mainframe* metaphor, one needs to move to a "*health-care equivalent of the low-cost PC*". As Gupta [39] argues, the objective is to move to an "*automated, real-time, inexpensive and very efficient*" system. The focus being on a **continuous monitoring** that leads to a **pro-active and preventive** system. The move encompasses hospitalization only for very serious conditions, envisioning *automated diagnosis and treatment*. This will naturally lend itself to **home & out-patient** treatment and monitoring. As mentioned, one needs also to "gear" the current research from focusing on the centralized medical information systems towards **assisting technology**. Bardram [8] also discusses a persuasive system, that tries to "steer" people into the correct behaviour and life-style that is best suited to their situation (the sport motivation could also fit this). And again resorting to market studies examples, Forrester Research in 2002 [9] already states the need to have a "*major shift, enabled by technology, to self-care, mobile care, and home care*". This empowerment of the individual has the added advantage that endows her/him[5] with a self-awareness and responsibility of her own malady.

In this context we propose Body Area Networks (BANs) as an integral part.

1.2.1 BANs

Let us start with a nomenclature standpoint: BANs are also called Body Sensor Networks (BSNs), Wireless Body Area Network (WBAN) and, to a lesser extent, Body Area Wireless Sensor

[3] What Forrester names personal medical monitoring.
[4] We describe sensors on the background section of §2.1
[5] We use the feminine version of the article from this point on without any loss of generality.

Networks (BAWSNs). As we will discuss, we do not assume that the network needs to be wireless, thus WBAN and BAWSN are too specific, and we accommodate actuators in our architecture, i.e. BSN may be too specific. Although we *accommodate* actuators most of our thesis work is based on sensors and the data they collect. As such we use BAN when the context refers to either sensors or actuators, e.g. when describing hardware abstractions, and BSN when only sensors are relevant, e.g. when describing information collection.

With the last paragraph in mind, we define a Body Area Network as a network of sensors and actuators within the limits of the human body area (inside the body or over the body) that communicate with each other and with a central more powerful node. These body nodes can measure physiological data, movement, position, etc. and dispense medication, electric shocks (pacemakers), provide imaging for poor sighted people, etc. The more powerfull node can be a mobile device, e.g. a smart phone, Personal Digital Assistant (PDA) or a fixed house central hub. The objectives of this network can be many-fold (which is one of the points of our thesis) where some examples are: monitoring vital signals (disaster victims, 1^{st} responders), health monitoring (home-care, in hospital), self-assessment (sports), deriving user-context (moods, coordinated group movement). The examples illustrate our focus on sensing, neglecting actuation. However one can also picture: closed loop insulin delivery, were the inputs for assessing insulin needs are not merely the glucose value, but also activity, stress level, etc., controlled drug release (using "pill size" containers remotely controlled), etc.

Research on BANs can be dated back to 1961 with work from Mackay [68] on radio telemetry within the body. The work from Zimmerman is also referred as a starting point for Personal Area Networks (PANs), with his 1996 paper [146] and MSc thesis [145]. Zimmerman mostly addresses the in-body communication, but also paves the way for the discussion of a network of devices within the body area. Naturally, several advances have been made in this area, notably developments in both the sensing/acting devices and the framework to handle/manage them. We discuss these in chapter 2.

BANs can provide part of the needs for monitoring and actuating in the context we described in the previous section. Especially on a front that has not seen much advance: **multi-parameter** retrieval and **correlation**[6]. Being able to tap into several inputs and, especially, being able to extrapolate new information based on them is a key functionality in BSNs. Another point is the realization that several applications will reside in the central node and access the resources, i.e. the BAN's nodes. Handling all the requests and **optimizing** those **resources** is also key to **fulfilling** applications' **requirements** and increasing the lifetime of the resources.

> *We defend that there is the need to provide multi-parameter correlations to allow correct, reliable and new (extrapolated) information for applications using the BSN. Meeting the multiple requests by different applications using the BAN and optimizing resource usage are the other essential capabilities of a framework for a BAN.*

[6] From another report on future markets, we can envision that "*The Global Multiparameter Patient Monitoring Devices Market [is] to Reach $3.4 billion by 2016*" [36].

1.3 – The transport to get us there

To achieve the aims motivated in §1.2, we propose a middleware layer that abstracts the underlying BAN to the applications, providing instead an information model that can be queried. The data on the model itself is derived using the raw data provided by the BSN. This middleware should also handle applications' requirements while optimizing resource usage.

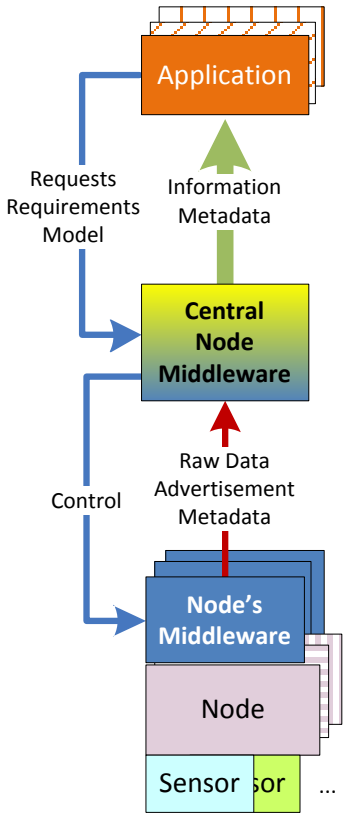


Figure 1.1 – Middleware functionality

The architecture we define is portrayed in figure 1.1, which is a typical middleware as proposed by Bernstein [11]. A middleware component resides on the central node and a "thinner" middleware layer on the devices (sensors or actuators). The main point we want to handle is the heterogeneity of available information and the correlation of disparate data. From Bernstein's definitions our work identifies itself as a framework, as we provide an Application Program Interface (API) to access the underlying resources, namely the nodes. We go beyond the framework/middleware definitions from Bernstein, as we add a service that infers new information. Our middleware not only provides access to the data from sensor nodes, but provides a service that correlates different information to infer new information.

An example of correlation of different sensing information is determining Cardiac Output (CO) from Heart Rate (HR) and Stroke Volume (SV). This calculation uses the Windkessel model as described by Sun et al. [111] and is a known physiological formula, $CO = SV \times HR$. Let us suppose we had an electrocardiac reader for SV, a HR reader and also an electrocardiac CO reader. We have an application that requires CO. With an intermediate layer (middleware) the process could be: **(i)** application requests CO information with specific requirements (maximum delay, maximum error, etc.); **(ii)** middleware checks its known models for deriving CO, and the available sensors to provide the raw data needed; **(iii)** after matching the possibilities with the request requirements it gets the raw data from the sensors; **(iv)** data is aggregated and metadata is added to the produced information. Examples of metadata are confidence on the value (based on the model and the data available), error margin (taking into account statistics of the sensors used) and time of assessment; **(v)** middleware provides the information and metadata to the application; **(vi)** application handles the information taking into account its metadata.

One could assume that getting the direct reading from the CO reader would be the best solution. However, there could be some factors that made using the correlation a better approach. If another application, or even the same, requested HR it could be more "economic" to get the SV instead of the CO. That would produce the HR requested and the CO as per $SV \times HR$. Other constraints could be if the CO reader had the battery nearly depleted or if it had an error greater

than allowed by the application's request.

Physiological models enable the correlation of the different inputs to infer new information. This field has seen great developments that range from the simple mentioned Windkessel model for CO [111] to others more complex such as insulin sensitivity.

Insulin sensitivity measures the ability of insulin to lower blood glucose, i.e. how effective the insulin is in enabling the storage of blood glucose in cells. Insulin sensitivity as derived from Keener and Sneyd [60, pages 804–806] is:

$$\text{Insulin Sensitivity} = \frac{\int_0^\infty \frac{G_b - G(t)}{G(t)} dt \left(p_1 + \frac{1}{V} \frac{\int_0^\infty r(t) dt}{\int_0^\infty (G_b - G(t))\, dt} \right)}{\int_0^\infty (I(t) - I_b)\, dt}$$

In the formula, $G(t)$, $I(t)$ and $r(t)$ are respectively blood glucose, blood insulin and glucose intake as a function of time. G_b is the glucose base value, I_b is the insulin base level and V the glucose volume per unit of body weight. p_1 is a constant related to the glucose decrease rate. This model is used in a meal tolerance test where insulin and glucose in the blood are measured along the test and the food intake is controlled. The $\int_0^\infty r(t)dt$ expression is the total food intake during the test. As all values are known it is possible to calculate the insulin sensitivity.

In our middleware approach we intend to embed this knowledge in components. This allows re-use of the component within the system by whatever other components need it. As an example we could define the $CO \leftarrow f(HR, SV)$ in such a component. If the system needed CO it could use the component to produce it (as long as HR and SV were available and the requirements for getting CO were met by this component). In chapter 4 we discuss in more detail the models and how they are used within our proposal.

Some further points to consider:

- although we focus our examples on in-body and over-the-body devices, the work is applicable to networks where the device is in the proximity of the user and communication is possible (e.g. a wireless scale);
- as mentioned, "to communicate" does not imply in what medium, which could be wireless, intra-body or wired (we discuss this in §2.4);

Within our thesis we assume the following:

- there will be a network star topology which we discuss in §3.2; where the central node acts as the hub of the network;
- there will be several sensors in a BAN able to sense different or the same parameters (more on this in §2.1);
- there will be several applications residing on the central node of the BAN;
- applications will need to access the sensor network to function;
- applications will have requirements on the inputs they want to receive (frequency, delay, etc.);
- there can be colliding (not necessarily conflicting) requests for sensor input from different applications;
- correlation of different inputs to provide higher level information will be a necessity.

The last bullet point is the key point to our thesis.

1.3.1 Contributions

Our main contribution is presenting a framework that provides a feasible way for application developers to correlate multi-parameter inputs in BSN using re-usable components. These correlations are based on models that can be pushed to a middleware. This enables the middleware to optimize resource usage while providing applications their requests. Namely we introduce the following:

Models in the framework: we defined an architecture based on modules (components) that state their information production and their input needs for that production. These modules also have associated a cost and capabilities for that production. When a requirement (or more) enters the system the middleware combines modules so as to produce the requested information while meeting the requests' requirements. The possible combinations are defined in the correlation model. The model abstraction is detailed in chapter 4.

Optimization: the choice of the modules to use for the requests issued is not constrained just by requirements. The middleware, apart from knowing what raw data is available, also tries to minimize the cost of using those modules. This takes into account the inherent costs and capabilities of the modules, as well as the requirements from the requests. Here we can see the advantage of managing all the applications' requests globally. This "omniscient" view provides the opportunity for "aggregating" similar or intersecting requests. We describe the optimization in §4.2.

In the process to achieve these goals we also defined and develop the following concepts:

- **Hardware abstraction:** we developed a framework that provides an abstraction for the devices on the BAN. For this, we defined data structures that hold the data (e.g. sensed data like HR, BP) and metadata (e.g. actuator characteristics like action, control parameters). These data structures are then used in a protocol for conveying commands and receiving responses from the central node to the devices. This protocol is oblivious to the underlying communication medium where it is being transmitted. These commands and data gathering are also abstracted, which means that only the information being requested/transmitted is relevant for using the layer. This provides an abstraction over Operating Systems (OSs), communication layers and nodes' data access. We discuss this further in §3.3.

- **Service Discovery (SD):** a needed characteristic of a middleware for managing devices is the ability to discover them. As such, based on current SD technologies we developed a service for our middleware. The wireless communication standards for low power devices have defined SD capabilities for networks with these devices. Bluetooth [14] and Zigbee [144] are the more widely used approaches in Wireless Sensor Networks (WSNs). Our aim with developing this service was not to necessarily improve on current work. As such, our approach borrows from these standards, simplifying some structures for a BAN and adding pro-active advertisement by the nodes. We discuss this in §3.6.

- **pub/sub system:** the inter-communication between modules in the model is done through a Publish/Subscribe (pub/sub) system. Modules subscribe to the inputs they need and publish their production. This is described in chapter 5.

To test the concepts defined, we built a middleware with most of the ideas discussed. It is currently mostly a working implementation in Java without particular optimizations. The devices used for the tests were SunSPOTs from Sun [113] as they have native support for Java with their Virtual Machine (VM) named Squawk. Although we defend the notion of supporting different types of devices we have not implemented the interface classes for other devices. We go into more detail in chapter 6.

1.4 – Outline

We start our discussion on BANs by describing their current state, their network characteristics, the sensors used/available and make some considerations on their hardware. After this background review, we discuss in chapter 3 our middleware proposal and how it fits our goals, the chapter is focused on a hardware abstraction layer. We dedicate chapter 4 to discuss the model usage within our middleware. In chapter 5 we address the flow of information using the model from the previous chapter. In chapter 6, we describe some details regarding our prototype implementation. We finish with chapter 7 presenting the conclusions and discussing some future directions. The appendices apart from the bibliography and acronyms list, present some further details on our implementation and the software used throughout the thesis' work.

1.5 – Notation

In some chapters we use Unified Modelling Language (UML) [83] diagrams to better illustrate the system components and their interaction. No specific knowledge is needed for reading the diagrams, and they only complement what is described in the text.

We use color in some figures to better distinguish some elements. This will not impose any restriction on black and white printed versions, as those elements also have other visual distinctions, such as stripes and vertical lines.

When we discuss component parts that have a direct reflection on the code developed, we use fixed width fonts as `NodeComm`.

Acronyms used are expanded the first time they are used in a chapter. After that only the acronym is used. For the most commonly used acronyms (e.g. IP[7], IEEE) and acronyms that are not relevant for the context (example of cellular networks UMTS) we use only the acronym. All used acronyms are defined in appendix B.

1.5.1 Nomenclature

Some of the terms used in this thesis are described here to remove any ambiguity that may occur.

Node is a hardware component that hosts sensors and/or actuators. It has communication capabilities (wireless, wired, etc.). In most cases it runs an OS where there is the possibility

[7] Note that there is no reference to intellectual property, so this refers to Internet Protocol.

to deploy software developed. This is not however mandatory, as there are some nodes that do not allow for this. In chapter 4 we may use "nodes" to refer to elements in a graph. This is only done where it does not raise any doubt;

Base Station (BS) is a special node on the network. It is a more capable node (processing, storage, energy, etc.) where the applications for which we aim our work run. Some examples could be a smartphone, a PDA or a laptop.

Model is a description of correlations between different information that is collected or produced in our system. By itself it represents only the definition of all the correlations known. In chapter 4 we describe how we build diagrams based on the known correlations, thus on known models.

Resource mainly refers to the nodes in the BAN. It is used in cases where nodes' resources themselves are relevant, such as energy, processing, communication.

2 Background

In this chapter we provide the context to Body Area Networks (BANs), discussing nodes for actuating and sensing, applications that use it, their network characteristics, energy usage and what distinguishes them from Wireless Sensor Networks (WSNs). The contents of this chapter describe work parallel to our proposal, providing the basis for discussing what we propose. We address this at the end of the chapter.

2.1 – BANs' nodes

We start this chapter by trying to materialize a BAN, with examples of possible nodes for it taken from available products and research prototypes. Our objective is to illustrate the context of our work and provide concrete instances of applicable scenarios. Figure 2.1 illustrates such a scenario with actuators and sensors in the human body[1].

Table 2.1 realizes the sensor examples from figure 2.1, with examples from commercial available products and research prototypes. The data from the table is derived from the references. In some cases we had to infer some information, in which case we duly noted it in the table. In others, cardiac biomarkers, it was not possible to extract or infer parts of the table data. Examples range from implantable sensors to wearable ones, biochemical compounds sensed to physical ones.

Data rates for the various sensors range from very few bits per second to some kilo bits per second. These values do not incorporate the time stamps and message headers, which impose higher needs from the communication channels. For illustrating time stamp sizes, we have, from the IEEE 11073 standard for health device communication [54], a definition of relative time data

[1] When we talk at the end of the chapter of correlations, the Vitruvian man will have more meaning.

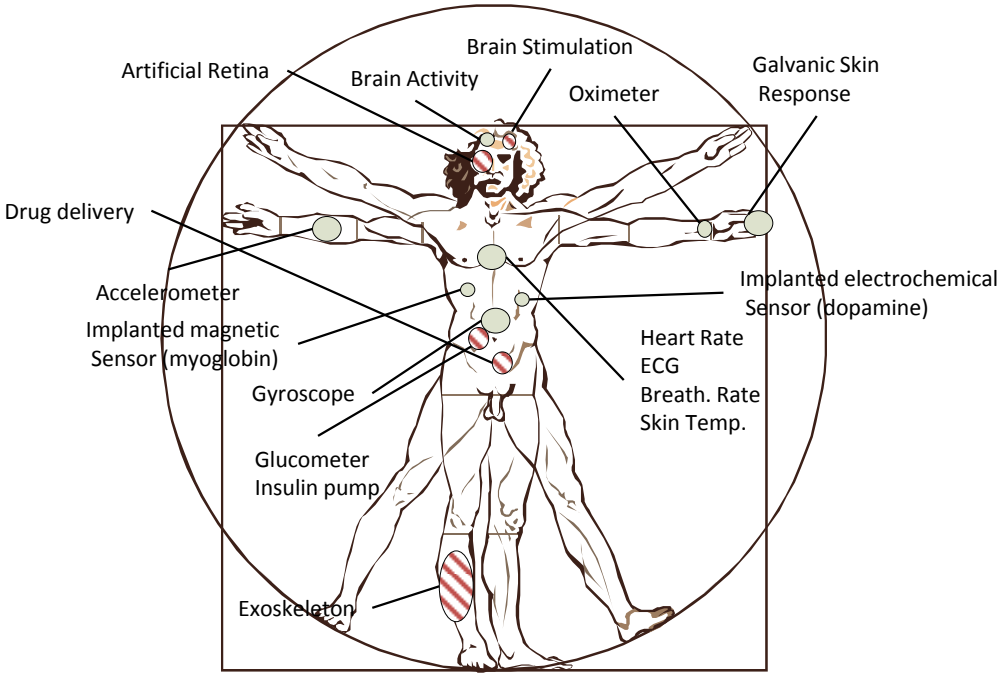


Figure 2.1 – Actuators and sensors in a BAN

(Vitruvian man from Microsoft Clip Art; royalty free)

with 32 bit size and for high resolution time a 64 bits size. This would be added for each sample, thus increasing the data rate. For the brain sensor, the sample size is not defined, and as such we do not have information on the data size. This would be over 640 kbps in the current design and over 4000 kbps in the desired 100 channels resolution.

Applications that use the sensed data are exemplified in table 2.2. As expected, applications that control diseases need to monitor several parameters. Only the parameters from table 2.1 are portrayed in table 2.2, i.e. the applications could use more parameters than the ones mentioned.

Table 2.3 gives some examples of actuators available as commercial products or research prototypes. Some of these actuators aim to have a closed-loop, i.e. being able to sense the environment and actuate accordingly. The closed-loop for diabetic management is a research topic that has seen much interest. However, it is still missing the control part of the loop. The Minimed example in table 2.3 provides the pump and the sensor, but the autonomic connection between both is still missing.

An important note is that these systems are currently disjunct. To the best of our knowledge, there is currently no framework to integrate the different sensor information. The same data could be handled differently by different applications, e.g. the relevance of Heart Rate (HR) for monitoring arrhythmias is different from monitoring sports activity. As we discuss at the end of the chapter, data could be correlated in different ways to produce new information. These points are the main focus of our thesis.

2.2 – Applications

In this section we provide more information on the types of applications we can see in future BANs.

Table 2.1 – Sensor examples

Device	Availability	Sensed	Technology	Frequency	Data Rate ∝	Energy ◊
BioHarness BT [140]	commercial	HR	detection of *QRS* complex in ECG	1 Hz	8 bps	21 h transmitting
		Breathing rate	conductive elastic measurement of thorax excursion	1 Hz	7 bps	
		3D Accelerometer	variability of a weight reference	50 Hz ⋈	500 bps ⋈	
		ECG	potential difference across electrodes in body	250 Hz	2500 bps	
		Gyroscope	angular momentum	1 Hz	9 bps	
		Skin temperature	thermistor	1 Hz	9 bps	
Actigraph GT3X+ [1]	commercial	3D Accelerometer	variability of a weight reference	30-100 Hz ⋈	360-1200 bps ⋈	31 days
Shimmer Research GSR Sensor [101]	commercial	Galvanic skin response	measure skin conductivity	up to 15.9 Hz	191 bps	60 μA
Nonin Onyx II [78]	commercial	O_2 saturation (full waveform)	measure light absorption by blood haemoglobin	75 Hz	1200 bps	2×1.5 V AAA (600 tests)
		O_2 saturation (display format)		1 Hz	8 bps	
Medtronic iPro CGM [71]	commercial	continuous glucose meter	electrochemical detection of glucose through its reaction with glucose oxidase	0.1 Hz ◁	1 bps ▷	up to 72 h
Brain sensor by Nurmikko et al. [79]	in research ‡	brain activity	microelectrode arrays detect neuron "firing"	40 k samples/sec×16 channels ℵ		12 mW
Molecular biomarkers by Ling et al. [67]	research prototype ⋆	serum cardiac troponin I, creatinine kinase, myoglobin	magnetic properties of sensors vary according to presence of biomarker	N/A		
Electrochemical dopamine sensor by Chan et al. [19] ⊗	research	dopamine	Interdigitated micro electrodes measured electrochemical reaction	50 Hz	–	10 pA

∝ data rates are based on the frequency and the accuracy in bits stated in the references; they do not include time stamps or message headers.
◊ values taken from the references as available.
⋈ for each of the 3 axis.
◁ sensor data is collected every 10 s by collector.
▷ our assumption of 9 bits per measure (up to 512 mg/dL (USA glucose units)).
‡ prototypes and clinical trials exist for devices with 16 channels.
ℵ more channels are needed (e.g. for decoding arm joint angles); 100 channels arrays are being developed.
⋆ tested on mice; measurement using Magnetic Resonance Imaging (MRI), but test values were extracted from explanted sensors.
⊗ the developed sensor was not made of nano tubes, which according to the authors, led to poor sensitivity.

Table 2.2 – Sensed values applications

Sensed	Applicability
HR	heart failure, arrhythmias, post-operative monitoring, sports training.
Breathing rate	asthma, Chronic Obstructive Pulmonary Disease (COPD).
Acceleration	fall detection, activity (Parkinson's, Alzheimer's, Stroke), reduced function (rheumatoid arthritis), sports training.
ECG	heart diseases, heart failure, arrhythmias, post-operative monitoring, sports training.
Posture (Gyroscope)	fall detection, sensory disturbance (diabetes, Parkinson's, Alzheimer's), sports training.
Skin temperature	arthritis rheumatoid.
Galvanic skin response	psychological arousal such as mental effort, excitement, shock, and especially stress.
Oxygen Saturation	post operative monitoring, asthma, COPD.
Glucose	diabetes
Brain activity	interpret commands for assistive devices; reconnect damaged neural pathways.
Cardiac biomarkers	myocardial infarction, heart diseases, heart failure, arrhythmias.
Dopamine	Parkinson's, stress level.

The IEEE's 802.15 working group for Wireless Personal Area Network (WPAN) has a task group for BANs, group 6. The group is "*developing a communication standard optimized for low power devices and operation on, in or around the human body (but not limited to humans) to serve a variety of applications including medical, consumer electronics/personal entertainment and other*" [49]. We will be referring to the work of this group in several places of this thesis.

When the group was starting, they did a survey of envisioned applications for BANs. This resulted in several medical and non-medical suggestions for applications. Proposals had to mention several characteristics of the applications, number of devices foreseen, channel used (in, out body), duty cycle, power consumption, etc. From the summary document edited by Lewis [25] we point out the following (see table 2.4 for some of the characteristics):

Vital signs monitor *(wearable sensor)*: several sensors for (medical) monitoring of physiological parameters. Sensors include ElectroCardioGram (ECG), temperature, accelerometer, pressure, breathing, Blood Pressure (BP);

Muscle tension stimulation *(wearable actuator)*: lacking the description, the name suggests an electronic muscle stimulator;

Sport training *(wearable sensor)*: aimed at professionals and amateurs, the objective is to monitor performance, using different physiological parameters (and thus sensor nodes). Gait length, HR, oximetry, acceleration would be required for most sports. Cycling would require motion analysis; team sports would need team interaction capabilities. Here time synchronization between motion sensors would be an imperative for correlations;

Glucose sensor *(implanted sensor)*: an implanted glucose sensor, which would transmit to a body surface *repeater*;

Endoscope capsule (gastrointestinal) *(in body sensor)*: a swallowable capsule that travels through the gastrointestinal tract transmitting video;

Deep brain stimulator *(implanted actuator)*: used for example for epilepsy and Parkinson's therapy. Stimulates with electrical impulses different regions of the brain;

Gaming applications *(wearable sensors)*: using motion and location sensor for Human Computer Interface (HCI), in this case gaming.

Some entertainment applications for video and audio streaming were also proposed. Abiding to the group's objective there were also animal monitoring examples of applications, which included physical, chemical, biological, mood and activity monitoring.

All the proposals:

- were either *low* or *extremely low* on power consumption (the entertainment audio/video related ones were the exception);
- used a star network topology where a central component (more powerful node) acted as a hub for the network; some applications could also use a tree, and others assumed a Peer to Peer (P2P) topology;
- had data link rate asymmetry: as expected applications would require more bandwidth in one direction than in the other. In most cases, the higher rate was directed to the central component, but in some (entertainment audio/video streams to headsets, glasses, etc.) it was originating from this central component. There were two exceptions to this trend, *forgotten things monitor* and *social networking using BAN*. The first is an application that detects when certain objects are out of range, which means they have been forgotten. The second application would be related to social interaction between people. Gaming (with auto partner pairing in the subway), exchange of personal info (business cards, match making) are some of the examples given. This assumes a P2P relationship between communication end-points, which leads to a symmetric need for bandwidth;
- mentioned Quality of Service (QoS) as a need, regarding latency and/or sensitivity to error.

Table 2.3 – Actuator examples

Device	Availability	Objective	Notes
MiniMed Paradigm Revel Insulin Pump [73]	commercial	infusion of insulin	has a sensor for continuous monitoring glucose values.
Brain stimulation [67] ‡	in specialized clinics	induce or apply electrical current to the brain for treating epilepsy, Parkison's physical symptoms, depressions, obsessive compulsive disorder	in our context, vagus nerve stimulation and deep brain stimulation are the most portable; the first sends electric impulses to the vagus nerve, the second has electrodes implanted in brain areas related to the disease to treat, generators are implanted in the chest for driving the pulses.
Exoskeletons eLegs [10]	company prototype	enable paralyzed individuals to walk; currently aimed at rehabilitation centres	wearable exoskeleton that allow walking in straight line, standing up, sit; sensor enabled control.
Artificial Retina [126]	prototype	electrical activation of nerves to provide motion and light detection for blind individuals	electrodes "fire" at 10 kHz; data rate 40 Mbps; energy up to 600 μA.
BioMEMS Drug delivery [80]	clinical trails	implantable micro devices with reservoirs for controlled drug delivery	electroresistive thermal controlled release; sensors for confirming drug release; sensor for triggering release.

‡ for a gentler introduction see the National Institute of Mental Health discussion [81]

Some of the applications fall outside of our working ground, as we are more concerned with the body area and not necessarily with the inter-body communication. Nonetheless, they present some examples of the potential uses of BANs.

Table 2.4 – Applications examples from IEEE's 802.15.6 [25]

	# Dev.	Channel	Duty Cycle	Power Cons.
Vital signs Monitor	< 12	On body	<1%	Low
Muscle tension stimulation	< 12	On body	<1%	Low
Sport training	< 12	On body	<10%	Low
Glucose sensor	< 12	In body to on body	<1%	Extremely low
Endoscope capsule	2	In body to on body	>50%	Low
Deep brain stimulator	2	In body	<50%	Extremely low
Gaming applications	12 to 24	On-body to On-body, On-Body ⇔ Out-of-body	<30%	Low

2.2.1 Requirements

From the IEEE 802.15.6 task group's document on applications requirements [25] we can extract the following:

QoS: applications will need assurance in the data connections to node devices. The most prominent one will be delay, another will be losses.

Data rates were not defined at the time the draft for the standard [6] was released, nonetheless several applications need specific data rates. From the technical requirements document [141] the range will be from 10 Kbps to 10 Mbps. The current draft for the standard has support up to 10 Mbps (using Ultra Wide Band (UWB));

Security will be required for the transport of sensitive data and access to it within the Base Station (BS) (privacy issues of data access outside the communication layers is not the group's focus).

The group also mentions that applications in BANs can have three types of data needs: **burst:** where there is a temporary need for high throughput; **low rate:** would be the most common, with a continuous low rate need (table 2.1 shows several applications that have this need); **emergency:** where data has to be transmitted regardless of battery life, other applications needs, etc.

In our view, error associated to measured data is also a requirement. However, this is not associated with the communication process and as such it is not the concern of the task group. We come back to requirements in §3.6.

2.3 – WSN and BSN

BANs (more specifically Body Sensor Networks (BSNs)) have similarities with WSNs, starting with having sensor nodes (albeit different) that form a network. However, we argue that there are several differences that make BANs prone to different problems, approaches and optimizations possibilities. Table 2.5 summarizes the aspects we regard as fundamental, stating differences and similarities.

In our view, BANs have a central component that receives all the information from the nodes and controls all of them. This is done as applications using it deem necessary. A Personal Digital Assistant (PDA), smart phone or a more powerful node can be this component or can act as a

Table 2.5 – BSN versus WSN (with input from Latré et al. [63, table 2] and Guang et al. [137, page 5])

	BSN	WSN
Distribution	**i)** Existence of a BS; **ii)** BS collects, maintains and processes the data; **iii)** Nodes will do minimal processing, sending all data to the BS; **iv)** Centralized system where BS controls all nodes; **v)** Node replacement is difficult in in-body sensor nodes; **vi)** Smaller number of nodes; **vii)** Nodes need to take biocompatibility, wearability into account.	**i)** A BS may or not exist or there may be several BSs (e.g. mobile nodes collect info, clustering); **ii)** As in BSN, but also on-demand querying; **iii)** Nodes will do processing, aggregation to alleviate communication or correlate results; **iv)** Distributed system, nodes decide cooperatively; **v)** Node replacement is difficult due to location, scale, etc.; **vi)** (usually) Wide areas covered by large number of nodes. **vii)** Nodes may need to be environment friendly, indiscernible from surroundings.
Comm.	**i)** One hop to BS; **ii)** Close range but attenuated by body; **iii)** Data rates heterogeneous.	**i)** Multi hop through network of sensor nodes; **ii)** Long(er) range; **iii)** Data rates homogeneous.
Data	**i)** Interest in different types of data; **ii)** Correlation of different type of data; **iii)** Accuracy from correlation, node accuracy; **iv)** Losses may not occur in emergency procedures, less redundancy; **v)** Likely to have new types of sensors nodes added; **vi)** Prone to have different applications using the same resources; **vii)** Security associated with personal/patient data.	**i)** Single or few different types of data; **ii)** Aggregating the same type of information; **iii)** Accuracy results from redundancy of inputs of the same data type; **iv)** Losses circumvented with redundancy; **v)** Typically static in terms of node types (adding new nodes of the same type); **vi)** Deployed with one specific application in mind; **vii)** Security level varies.
Energy	**i)** Constrained; **ii)** Lifetime years/months/days; **iii)** Recharging may be possible with scavenging or inductive coupling; **iv)** Scavenging from body vibration, temperature.	**i)** Constrained; **ii)** Lifetime years/months; **iii)** Recharging may be possible in some scenarios with natural energy or scavenging; **iv)** Scavenging solar, wind, vibration (machinery, bridges, etc.).

Gateway (GW) to the central component (e.g. a PC on a home environment). As we discuss in §3.2.1, the network topology will be a star, where all nodes are one network hop away from this central component, which we call a Base Station (BS).

Nodes will refrain from much processing, delivering data to the BS mostly unprocessed. This is not to say that nodes will not produce some interpretation of its data as another source of information, e.g. producing a HR from an ECG. What we mean is that nodes will not correlate or aggregate data from different sensors and will not change behaviour based on readings collected. That is, nodes will not have autonomous behaviour with decision taking. The Equivital node from Hidalgo [46] is an example of a node that while doing some processing corresponds to our view. The device is a strap-on with ECG, gyroscope, skin temperature and accelerometer sensors. Among its outputs, it has HR derived from the ECG. HR is extracted from the ECG by the node itself, and the BS would be interested in both. So, nodes will have some inference on them, but limited to the data collected by themselves. Nodes will not take any decisions regarding actions or change of sensing behaviour.

In BANs, the set of data treated will be very heterogeneous, with possibly complex relationships

among the different types of data. BANs are prone to have different types of nodes added to an already deployed network.

In WSNs the central component may or not exist and if present will have a more limited view of the whole system. WSNs tend to process the information in a distributed approach; with nodes treating, correlating and aggregating information, forming clusters, etc. The network in WSNs is multi-hop as they tend to cover large areas. Sensors in WSNs are homogeneous in terms of hardware and data acquisition. As such, the information processed will normally be of the same type, although some frameworks do offer limited support for heterogeneity, where the proposal from Steffan et al. [108] is such an example.

Both frameworks have similar energy constraints, with research on recharging also sharing some commonality, as the surveys from Paradiso and Starner [88] and from Roundy et al. [96] show (we dwell on them in §2.5).

The differences stated lead to an easier to manage network in BANs in terms of connectivity establishment and coordination/optimization of sensor nodes, given the single-hop topology. This opens more possibilities for optimizing the sensor network usage. However, BANs monitor different types of information with complex interrelations between them. These interrelations may be different from application to application. To add to this, applications will also have different requirements on the usage of the network (delays, error, rates, etc.).

2.4 – Network characteristics

The BAN is the network comprised of the nodes surrounding the human body, similar to that depicted in figure 2.1. This network will connect to a local network or a wide area network, either connecting to a home hub in the first case or to a remote monitoring service for the latter. This is what is illustrated in figure 2.2. Within this thesis we address only the BAN, but we should keep in mind that a gateway node, most likely the BS, will need to interconnect with other networks.

Regarding the number of nodes in a BAN, Latré et al. [63] citing other sources state that 20 to 50 nodes are expected. In a Patel et al. [89] publication, the IEEE task group assumes that the desired number should be less than 64.

In its technical requirements [141], the 802.15.6 group states that the communication layers "*should support simultaneous co-located operation of at least 10 randomly distributed multiple BANs in a volume of* $6 \times 6 \times 6$ *meters in crowded places such as subways, hospital wards, music concerts, etc.*" They also require bandwidth sharing between co-located BANs with a coordinated approach to high duty cycle applications that provides a graceful degradation and no coordination for low duty cycle applications.

For the physical communication system, the medium can be wireless, wired or use the human-body. We will briefly discuss these options.

2.4.1 Wired versus wireless

Currently the most frequent option in commercial products and research prototypes is to use wireless communication between nodes. Bluetooth [13] and IEEE 802.15.4 [48] are the most prominent approaches. ZigBee [144] is a network layer that stands over 802.15.4. ZigBee is a

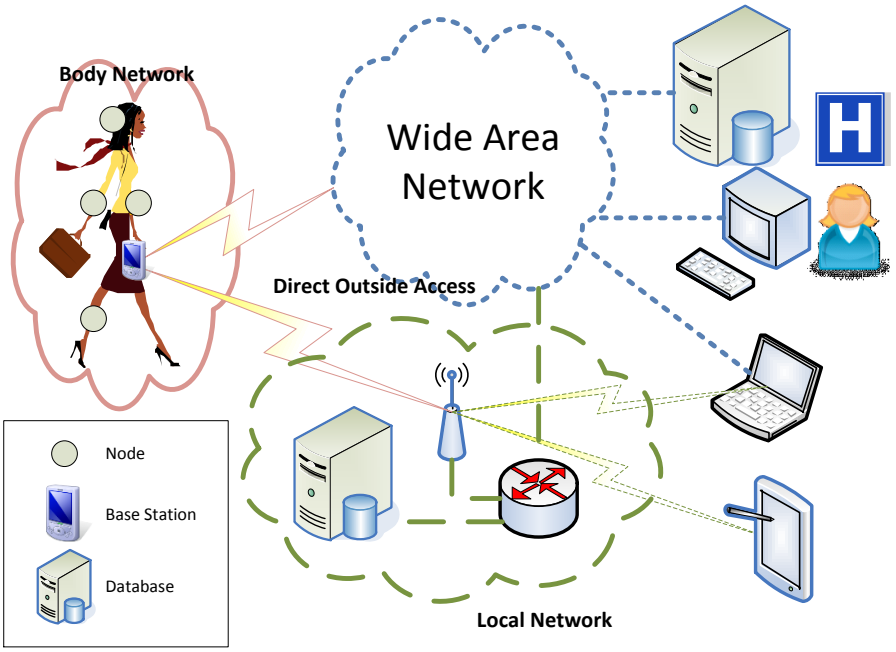


Figure 2.2 – Networks involved in a BAN

consortium based standard that, on top of 802.15.4, defines application profiles and a network layer module. IEEE 802.11 [50] is not considered as an option, mostly due to its high energy consumption. If used, it is mainly for connecting the BS to the rest of the infrastructure. The same applies for cellular technologies.

UWB [59] has been studied as an approach for communication within the BAN. As it can provide higher data rates at short distances, it is seen as a good candidate for these environments. The IEEE 802.15.6 group [49] uses UWB in one of the MAC/PHY options that they are defining for BAN communication. There are several studies on the use of UWB in BANs (some from the IEEE group). Zasowski et al. [139] provides an example of propagation studies on an anechoic chamber to measure the channel parameters from 3 to 6 GHz UWB waves. The authors reached the conclusion that it is possible to use UWB for small distances (15 cm), despite the strong variability in reception quality due to the human body. They also suggest the use of multi-hop communications for front to back communication[2].

Wireless systems however have some drawbacks:

- **Energy consumption:** although great efforts have been made to reduce the energy necessities of wireless communication, it still is the most power consuming component in a node. Greater capacity currently implies bigger sized batteries and size in BANs is an extremely important factor. The new developments in battery technology (see §2.5) can however mitigate some of these drawbacks;
- **Rates:** Bluetooth in its late 4.0 incarnation [14] supports link rates of 1 to 3 Mbps. The low power version [14, Vol. 6] has link rate of 1 Mbps. These are the over-the-air rates, while for the application layer it amounts to "*721.2 kbps for Basic Rate [and] 2.1 Mbps for Enhanced Data Rate*" [14, Vol. 1, page 17]. 802.15.4 is capable of throughputs of 250 kbps over the air. As we mention, the 802.15.6 draft defines 10 Mbps using UWB;

[2] As we mentioned we discuss the topology (star versus multi-hop) in §3.2.1.

- **Propagation:** signal barriers, multipath effects and the presence of dielectrics can affect the quality of the signal received. In BANs the link path is short, however the human body can shadow the signal (e.g. communication from the front to the back), water (50% to 65% of the human body) absorbs 2.4 GHz waves (802.15.4 and Bluetooth's working frequency) and reflections in the working environment are also present;
- **Interference:** as most of these standards work within the 2.4 GHz frequency, we are bound to have problems of coexistence. Bluetooth, for example, is able to cope with 802.11 (also in the 2.4 GHz) by hopping to different channels, but there is a limit to this capability. We also have the problem of different BANs coexistence;
- **Security:** wireless is a shared medium by default. As such, it is easier to eavesdrop on wireless communications;
- **Health Issues:** at present, international organizations provide guidelines for safe usage of devices that use radio frequencies, where the World Health Organization (WHO) [135]'s International Commission on Non-Ionizing Radiation Protection (ICNIRP) [58] is a reference. This leads us to believe that current radio frequency equipment is working within safe limits, however there are still studies and concerns about the usage of wireless devices near the human body, where the recent report from International Agency for Research on Cancer (IARC) is one [7] [3]. Naturally, in BANs these issues are augmented, which will raise public suspicion and resilience to embrace these technologies. The IEEE 8021.15.6 group is taking into consideration the Specific Absorption Rate (SAR) regulations. These regulations define the maximum exposure to electromagnetic fields a device may impose on a human body so not to cause any health problem. Several guidelines exist for the USA, Australia, Europe and Japan. As example the limits for the USA and Australia for head exposure are 1.6 W/kg in 1 g of tissue and for Europe and Japan 2.0 W/kg in 10 g tissue [141, section 15].

The advantages of wireless are easily noted as they drive their current widespread use: ease of initial deployment, easy addition of new nodes, no wires.

The issues, however, lead to the need to research the usage of wired communication paths in BANs. In some scenarios this can be a possible solution, namely when the node lies outside the body. As arguable advantages of this concept we could mention the opposite of the drawbacks of wireless pointed previously, but disadvantages also exist:

- **Node Placement:** in wired communication there usually needs to be an already foreseen connection point to add new nodes. As such locations to place new nodes are restricted;
- **Full Mesh:** being able to have connectivity between all nodes is more difficult due to the number of connections that need to exist. And again, foreseeing future nodes may lead to complex architectures;
- **Lack of Standards:** currently, to the best of our knowledge, there is no research on protocols for this type of communication. However some serial protocols deployed in other scenarios already exist and can be used in this environment (e.g. Inter-Integrated Circuit (I^2C), Serial Peripheral Interface (SPI), Controller Area Network (CAN)).

[3] Refer to the same WHO references [58, 135] for more.

Smart fabric is a technology that enables wiring textiles while making them fit for clothing. The garment's textile is thus conductive enabling sensing as discussed by Pacelli et al. [87] and data transmission. Smart fabric was introduced several years ago by Post and Orth [92], but there are few companies providing support for it [106, 119, 120]. Coupling smart fabric with techniques such as Pushpins by Lifton et al. [66] and Networked Surfaces by Scott et al. [99] may provide solutions to these problems. Smart fabric producers are already realizing products capable of transporting electric signals in normal textile yarns. Thus, one can produce garments with the capability of carrying data information and electric power to sensor nodes. It is common to refer to these textiles as *e-textile*.

Pushpins [92] are sensor nodes that can be pinned to a specially crafted surface that enables communication paths. The pins have two different length contacts that can be pushed into the laminated surface. The surface has two layers of conductive material, at different depths. This permits that pins inserted into the surface establish conductive paths to other pins, regardless of their position on the surface. A software layer then handles the communication setup, neighbour discovery, addressing, etc. Note that Pushpins do not necessarily use just these contacts for communication, there was a radio module under development and an infrared one was already available. However, we want to stress the part that addresses some of the problems we mentioned for wired communications in smart-fabric, namely node placement and full mesh.

In Networked Surfaces [99], the objective is similar to that of Pushpins. The surface is augmented such that special objects /qccan acquire connectivity to data and/or power infrastructure, simply by being in physical contact with that surface. This contact could be made anywhere in the surface. In this proposal several mini-pads for establishing contacts exist on the Surface. A distributed architecture manages these pads, detecting connectivity with objects.

These two approaches would not be directly applicable to smart fabric, but could be used as starting points. Note that Pushpins is from 2002 and Network Surfaces from 2000, but to our knowledge nothing has been applied to e-textile. An *all conductive t-shirt* would make wired communication a much more attractive solution for BANs.

2.4.2 Communication using the human body

Another channel of communication is the human body itself. Body Coupled Communication (BCC) uses near field intra-body communication, with very low frequencies. 0.1 to 1 MHz carrier frequencies were suggested in the seminal work of Zimmerman [146] on the subject. His first prototype used a 330 kHz wave for a 2400 bps data rate. It is with Zimmerman and Gershenfeld's work that the term Personal Area Network (PAN) appears. The PAN area is a slightly wider than a BAN; a weight scale could be an example of a node in a PAN, exchanging business cards by shaking hands is another. In BANs, this last example would mean the interaction between two different BANs[4].

There have been developments in BCCs, frequencies are now around the 30 MHz where measured gains have been higher than in the kHz region as Hachisuka et al. [41, section 2] refer to. The work from Moon et al. [75] in the 45 MHz frequency showed high fidelity with transmission at 30 cm distance (this indicated a capacity of 10 Mbps at this distance)[5]. Other experiments by Park et al. [56] were able to transmit videos between two ultra mobile PCs using a

[4] In our work we do not exclude PANs, as what we propose is also applicable to them.

[5] The main purpose of Moon et al. was to investigate an electrode that was bio-compatible with the human skin for long term uses.

2 Mbps connection with 10^{-6} error rate, at a selective frequency between 8–40 MHz. In this case distance was higher, as all experiments involved two devices, which the subject's body connected.

Advantages of BCC include increased channel privacy. Since the communication is restricted to the human body, interception is more difficult than in other wireless settings. Another advantage is its low transmission power that makes it less troublesome with regards to SAR compliance. Its low power also implies energy savings when compared to wireless communications. Falck et al. [31] use this low power, limited area of reception to establish a body identifier within devices in the same body, after which they use another channel (IEEE 802.15.4) to establish communication. In this work, BCC is used to certify the devices in the same body and establishing a unique body identifier. 802.15.4 is then used for higher data throughput. Their prototype system transmitted at 125 kHz, with a throughput of 4 kbps and power consumption of 2 mA at 3 V.

We have that BCC can provide some of the advantages of wired communication, with problems of node placement and mesh connectivity more easily addressed. There are however issues regarding interfacing with the body (probe sizes), skin irritation (that Moon et al. [75] addressed), communication standard and low rates (the higher rates are only now being researched).

2.4.3 QoS

As we mentioned in §2.2.1 applications will need quality assurance in some of the connections they make. To handle this there are some proposals for QoS on BANs. We briefly mention some.

The IEEE 802.15.6 group mentions QoS in the current draft, but it does not define it. There are some proposals from the group. Shu and Dolmans [103] propose defining two types of communication handling in carrier sense networks: α has higher latency and reliability guarantees than β. Latency management is achieved by controlling the back-off window so that α connections wait less time for transmitting. Reliability uses back-off attempts, α connections have higher back-off attempts than β. As the authors describe, this is a differentiation service.

Hernandez et al. [45] proposed to the IEEE group augmenting the correction coding for higher priority applications. The idea is to use Forward Error Correction (FEC) and retransmission requests for applications that require more reliability. Coding is part of the scheme for the FEC and retransmissions. These retransmissions are based in parity bit codes instead of full retransmissions. This diminishes overhead, decreasing latency for retransmissions. They suggest different schemes for encoding/decoding that react to channel conditions.

Outside of the IEEE group Zhou, et al. [142] aimed at providing a framework that handles QoS guarantees by using reserved slots for transmissions. They start by defining an abstraction of the MAC layer so that their framework is agnostic to the underlying network[6]. This virtual MAC enables collecting information regarding the network layer so that the QoS scheduler and the admission control components can ascertain network characteristics. They assume a star topology where the central node is the coordinator of the QoS scheme. The proposal defines specific time intervals for reserved traffic. There is an interval for traffic from the central component (aggregator) to the nodes and an interval for the reverse path. A best-effort slot occurs after the reserved slots. The reservation for traffic from nodes to aggregator is polled by the aggregator from the nodes. QoS scheduling techniques are defined for reserving time slots. Admission control is also part of their proposal.

[6]This is a similar objective that we have in the hardware abstraction layer presented in chapter 3.

These two approaches, from the IEEE group and from Zhou et al., are different and at different layers. Although, it would seem that this would allow them to be combined, the reservation approach from Zhou et al. would not benefit from the higher quality of the IEEE group's proposals. Zhou et al.'s proposal manages access to the medium as they introduce time slots. Although agnostic to the underlying layer, they circumvent the layer's medium access control by defining their time division approach. Even the best effort traffic that would use the regular access to the medium, would not benefit from the IEEE proposal as this traffic would not be marked as higher quality. Another point is that the current virtual MAC from Zhou et al. does not enable discovery of QoS parameters from the underlying network layer. This is by design, so that the abstraction does not depend on capabilities from specific implementation of lower layers. However, this also undermines the possibility of using these lower layer capabilities.

2.4.4 802.15.6 communication channels

The IEEE 802.15 task group 6 will adopt BCC as a communication channel for BAN, referring to it as Human Body Communication (HBC). Adding narrowband (from 402 - 405 MHz to 2400 – 2483.5 MHz), the 802.15.6 working group will define three physical channels for communication in a BAN [6]:

- **Narrowband:** with information data rates from 57.5 kbps up to 971.4 kbps;
- **UWB:** with information data rates from 0.5 Mbps up to 10 Mbps;
- **HBC:** with information data rates from 125 kbps up to 2 Mbps.

The group's timetable indicates circulation of the standard for approval at the end of 2011.

2.5 – Energy

Power sources pose a considerable constraint on BANs. Nodes need power mainly for communication, sensing/actuating and processing[7]. Currently energy for these operations is mostly drawn from batteries with research in other sources. The examples from tables 2.1 and 2.3 use batteries, direct cable connections (brain sensor and stimulator) or induction (biomarkers, drug delivery).

In table 2.6 we have some examples of wireless radio chips and their power consumptions. Comparing with table 2.1 and the discussion by Hanson et al. [43], it is noticeable that communication spends a good percentage of the power budget. Of course, the comparison must also include duty-cycle, which may be lower for communication systems if data is stored locally or processed on the node before being sent. For completeness, an example of processing consumption is the 16 bit ultra low power MSP430 microcontroller from Texas Instruments [118, section 2.3] that in active mode consumes 300 μA and in the lowest power mode 0.1 μA. This microcontroller is used, for example, on the Shimmer platform that the galvanic skin response sensor from table 2.1 connects to.

The use of batteries has two main problems: size of batteries and their replacement/recharging. While in some cases replacement is easy, and so batteries can be used (e.g. glucose meter), in others it is not a real possibility (e.g. implanted nodes). Even the first case would be considerably more wearable if replacement or recharging was not an issue. Recharging implantable devices

[7] Including analogue to digital conversions.

usually involves induction techniques as discussed by Olivo et al. [82]. Battery size is hindered by the power output capability that is related to the area of the electrodes as explained by Roundy et al. [96]. Technologies to decrease thickness while increasing area are an approach and the advent of nano-particles opened the field even more[8].

Table 2.6 – Wireless chips energy consumptions examples

Device	Technology	I_{sleep}	I_{idle}	I_{Tx}	I_{Rx}
Texas Instruments CC2540 [117]	Bluetooth low power	0.4 μA	0.9 μA	21 mA (-23 dBm)	19.6 mA
Texas Instruments CC2420 [116]	802.15.4	20 μA	426 μA	8.5 mA (-25dBm)	18.8 mA
Amp'edRF BT22 [2]	Bluetooth 2.1+EDR [13]	0.5 mA	3.6 mA	30.5 mA (master), 33.5 mA (slave) ‡	
RedPine RS9110 [94] ℵ	802.11bgn	0.52 mA	1.10 mA	200 mA (15 dBm)	149 mA

I_{sleep} – current drain when radio is in a connection-less state;
I_{idle} – current drain in standby mode (between connection/advertising);
I_{Tx} – current drain while transmitting.
I_{Rx} – current drain while receiving;
‡ – Using 921 kpbs on the serial interface for communication, only master/slave data available
ℵ – advertised as ultra low power; idle is connected to access point in power save mode (beacon 200 ms); transmission and reception at 22 Mbps

In a BAN there are some resources that can be scavenged for energy production for low power nodes. As Paradiso and Starner [88] point out, vibration/movement and thermo gradients are such resources. For kinetic sources we could use the body's movement, foot steps (e.g. electrostatic generator in boots), etc. Thermoelectric conversion takes advantage of temperature differences and the associated heat transfer. A voltage difference is created between elements due to the temperature difference. Paradiso and Starner [88] point out that the efficiency of these sources runs very low (under 10%) even for high temperature differences (from 200°C to 20°C). However, they give an example of a wristwatch powered by such a generator, which uses a battery for "cooler" times. A factor that impacts the power generated is area size of the thermo-couples, the pair of elements where the temperature difference is captured. Olivo et al. [82] cite experiments of thermocouples with 1 cm^2 area and a 5°K temperature difference producing 1 μW and a commercial solution with 95 mm^3 and temperature difference of 5°K producing 30 μW.

Olivo et al. [82, Table III] discuss the various sources for implantable devices and summarize the power generated in some experiments. Inductive links are the biggest providers ranging from 0.14 mW to 150 mW from the references they collected. Kinetic sources only provide from 40 μW to 80 μW, and thermoelectric 1 μW to 30 μW, as we saw. Another interesting source are fuel cells. These cells have the advantage over regular batteries of maintaining production of energy as long as the reactants that compose them are available. Olivo et al. [82] point to glucose fuel cell experiments that led to productions of 2.2 μW to 430 μW.

Power is a field of research with several applications, one being BAN nodes. Node deployments depend on it, as energy is a need in most of these nodes and their particularities (size, placement, accessibility, etc.) make them demanding users.

[8] For some discussion on nano technology the interested reader is referred to http://www.understandingnano.com/batteries.html as a starting point.

2.6 – Our work in BANs

In this chapter we set up the context for BANs. The issues we discussed are not directly the focus of our work but influence it and drive some of its purposes. With this chapter we tried to illustrate that background so that our proposals come more naturally.

Our underlying BAN scenario is that portrayed in figure 2.1. Users are *manned* with several sensing devices (acceleration, blood glucose, heart rate, oxygen saturation, breathing rate, Geographical Positioning System (GPS), etc.), actuators (drug delivery, insulin pump, etc.) and a central component (PDA, smart phone, house hub, etc.). Several applications that process information about the body reside in this BS. These applications can range across several fields: exercise monitoring, post-operation surveillance, emergency triage, HCI, etc.. Several of these may be running at the same time on the BS. Applications will need the same or different information with stricter or looser requirements. New node devices are "plugged in" to enhance or add capabilities to the resource pool while others lose connectivity or "die" out. This dynamic, volatile environment poses difficulties in managing and discovering resources to accommodate application requests.

Our proposal is a middleware layer that mediates the interaction between the application and the resources, the network nodes. Applications will only need to issue *information* requests to the middleware with a set of requirements to be met. The middleware will in turn map these requirements to metrics that will serve as quantifiers to assess the capability of the underlying resources to fulfil the requests.

The main objective of this middleware is enabling correlation of information to infer new information. Thus, applications can request not only what the nodes can sense but also information inferred from the sensed data. An example of correlated information is the model we saw in chapter 1 (based on Windkessel model as described by Sun et al. [111]):

$$CO = SV \times HR$$

that is, Cardiac Output (CO) (volume of blood outputted from the heart per unit time) is given by the volume of blood pumped by the heart per beat (Stroke Volume (SV)) times Heart Rate (HR). Another formula is related to oxygen delivery, i.e. the amount of oxygen that is made available to the body per minute. This is presented by Law and Bukwirwa [64] as:

$$O_2\,delivery = CO \times Hb \times 1.31 \times SaO_2$$

with Hb being the concentration of haemoglobin, SaO_2 the oxygen saturation and 1.31 a constant for unit conversion. This illustrates correlation of information from another calculated information.

Another example, not formula-based, of inferred information, could be a heart monitor alarm that takes into account current body movement to assess the relevance of increased HR.

Applications can request the same or different information. Shared requests should be optimized by the middleware so as not to strain resources. Moreover, these commonalities will occur more often when inferred information is requested. All data that is used to infer the

requested information is subject to commonality detection, i.e. intersection in requests will not occur only in the requested information but in all the information tree used to correlate the information.

In the following chapter we discuss our proposal to achieve these goals.

3 Hardware abstraction layer

This chapter introduces the middleware we propose and its two layers. The *bottom* layer is responsible for hardware abstraction and communication between the nodes and the Base Station (BS). This is the main focus of our discussion. The upper middleware layer, responsible for information abstraction and its management, is analysed in chapters 4 and 5.

3.1 – Introduction

The aim of middleware architectures [11] is to provide applications with an abstraction over the underlying complexity; be it hardware (the type of node, what temperature sensor, etc.), Operating System (OS) (TinyOS, SunSPOT Squawk, etc.), communication layers (ZigBee/802.15.4, Bluetooth, etc.) or Service Discovery (SD) (Bluetooth, ZigBee, SensorML, etc.).

In our case, adding to the above mentioned responsibilities, we state that Body Area Network (BAN) middleware should:

A) collect data from sensor nodes;
B) convert these data to relevant information in a human body model, and collect metadata associated with these data;
C) correlate, according to the known human body models it has, the information received and its metadata;
D) answer requests from applications based on the information in the model while providing the related metadata;
E) optimize resource usage (turn on/off, increase/decrease frequencies of collection, etc.) while complying with requirements set by the applications.

Note that points C and D encompass deriving new information from data sensed by the nodes, according to applications' requests.

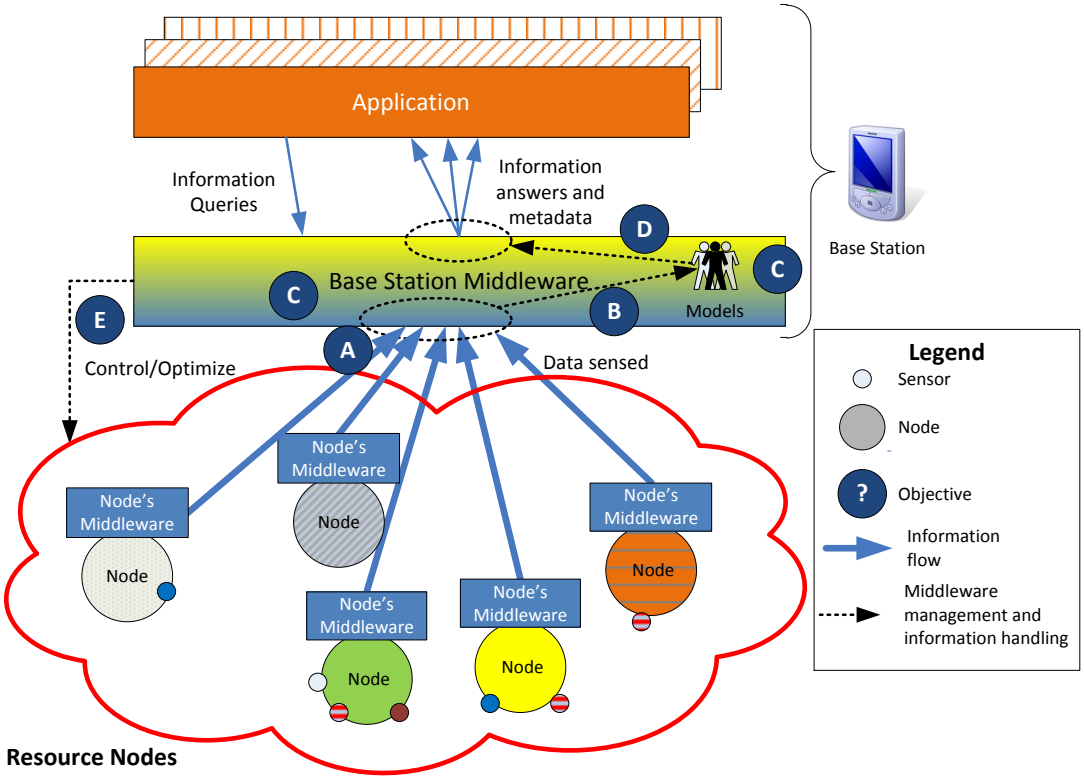


Figure 3.1 – Proposed global architecture

Figure 3.1 depicts the architecture where we can see several different applications assessing the resource nodes, which are heterogeneous (different hardware, different sensors, etc.). The figure also shows that some middleware parts run on the nodes and the main block on the central component. As we discussed, this central component will be a more capable node such as a Personal Digital Assistant (PDA), smart phone, home PC, etc. Nodes will be the sensor and actuator devices. As mentioned, we are more focused on the sensing part of the BAN.

The middleware on the node is responsible for:

i) advertising nodes' capabilities (sensing information, accuracy, rate, resources available (energy, processing, network), etc.);
ii) receiving and processing the central component's commands (turning sensors on/off, changing rate, etc.);
iii) answering requests from the central component.

The middleware on the BS has greater responsibilities, which are to:

1) incorporate models for correlation and representation of information;
2) receive requests from applications with requirements to be fulfilled. An example could be an application requesting Blood Pressure (BP) with a set of requirements (delay, frequency, etc.) and the middleware discovering which components to use for delivering that information (data from sensors and eventual correlation of these data);
3) derive from the different models, requirements and resources available, a solution that satisfies applications while optimizing resource usage;
4) control nodes according to the solution found, and request and receive information from them;

5) maintain the model flow of information and associated metadata based on sensor input (data and new sensor advertisements) so as to re-evaluate the solution found.

Point 3 for the central component incorporates the optimization process and is the most relevant to our work.

We can split our middleware into two generic layers that separate its main responsibilities: hardware and information abstractions, as shown in figure 3.2. The hardware abstraction interacts directly with the nodes and provides an interface that hides all the detail regarding this interaction. From the points stated above the hardware abstraction achieves: A, D (partially), E (provide support for); all points from the node middleware; 4 for the middleware in the BS. The Information abstraction layer interacts with the applications, providing an Application Program Interface (API) to assess the information model it maintains. This amounts to points B, C, D; and 1, 2, 3, 5 for the middleware in the BS.

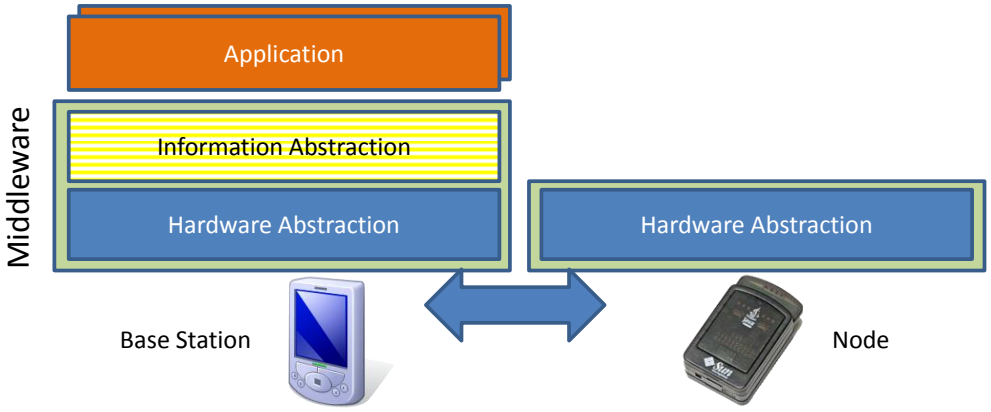


Figure 3.2 – Middleware layers

From figure 3.2 it is easily seen that all information processing is done in the central component, the BS. Our approach is to consider the nodes in the BAN as dumb parts of the system that "obey" the commands from the central component and report back to it. All the decisions and processing are done in the central component. We discuss this further in the next section.

3.2 – Application needs and design decisions

Our proposal is based on what we assume will be applications' needs in a BAN architecture:

- applications need to issue requests for specific information with a set of requirements to get that information (e.g. : blood pressure every 30 secs with a 95% confidence interval);
- for being able to fulfil applications' requests, access to more than one type of information (possibly with different requirements) is needed or greatly enhances the information collected.

Our system approach implies that:

- models that define how different information correlate to produce other information are available (we will expand on this in chapter 4);
- sensor nodes have several sensing capabilities (heterogeneous nodes transmitting different types of data).

During our design process we took the following decisions:

- applications are hosted in a more capable node (in terms of processing capabilities, battery power, network connectivity) than the regular sensor/actuator node, i.e. applications will run on the BS;
- the middleware is deployed on the more capable node and on the resources' network (in these less powered nodes the middleware is a *thinner* layer);
- the network topology is a star where the sensor/actuator nodes connect to the BS in one-hop communication link (see next section).

3.2.1 Star topology

We assume a star topology as the most natural option for a BAN with close distance and centrality of the BS with regard to the human body. This assumption is not without problems, as we have seen in §2.4.1, when we referred to interference and propagation issues of wireless communications: signal barriers, multipath effects, presence of dielectrics can affect the quality of the received signal, human body shadowing the signal (e.g.: communication from the front to the back and body water composition that absorbs 2.4 GHz waves) and reflections in the working environment.

Natarajan et al. [76] did an analysis of multi-hop versus star topology using experiments with a TelosB node[1] on a 2.4 GHz radio wave frequency. Although limited to the specific radio hardware of the TelosB (the CC2420 from Chipcon) and to the frequency of 2.4 GHz (the one used for Bluetooth and 802.15.4), some insight can be taken from the study.

Several aspects were tested: Packet Delivery Ratio (PDR), number of retransmissions, energy consumption and network lifetime. These retransmissions result from packet losses and from multi-hop traversal, i.e. routing from intermediate nodes. They used a 12 node BAN with experiments run on 14 volunteers in 3 different environments. The environments were: a laboratory room (with several reflective paths), a home environment with large areas, and a roof, which exemplified an open space. The authors defined two multi-hop routing schemes. One optimized for PDR, choosing paths that led to a higher PDR regardless of number of hops. The other scheme optimized for the number of retransmissions. Different transmission powers were also tested.

The multi-hop retransmission scheme topped the other options in every metric except for PDR, where the protocol tuned for PDR proved the best. In the laboratory environment the star topology was always very close in every metric to the retransmission scheme even for low transmission power. In the other cases, for intermediate to high transmission power the star topology was also close to the retransmission scheme. Even in network lifetime and energy consumed these high power settings were the best option as they led to fewer retransmissions and an overall gain, even for the retransmission scheme.

The article concludes that the retransmission scheme is the one with most advantages. They note that there is no one-size-fits-all solution and that the star topology will be inherently easier to implement with regard to routing protocols. Multi-hop will need to store state (unless on-demand protocols are used), and have network discovery running, which adds to network and computational overhead. Optimizing for a specific metric will be another task for the node to process. The retransmission scheme aims for a low number of hops and, as such, complexity and network overhead should be low.

[1] These programmable nodes were developed by the company CrossBow (http://www.xbow.com/). They are no longer produced.

As only the TelosB radio was tested, the energy consumed and the network lifetime, which is based on energy consumed, are specific to this hardware. This is duly noted in Natarajan et al.'s paper. We would add that transmission and reception is also biased by the radio used, which makes the PDR statistics specific to the CC2420.

The IEEE 802.15.6 group [49] that is defining the Media Access Control (MAC) and physical layers for BANs is envisioning several different frequencies: narrowband (around 400MHz, 860MHz, 900MHz and 2.4GHz frequencies), Ultra Wide Band (UWB) (on the 4GHz and 6.5GHz to 10GHz frequencies) and also Body Coupled Communication (BCC) or Human Body Communication (HBC) (in the 16MHz and 27MHz frequencies). This will provide more freedom on the network topology. In the draft standard the topologies defined are a star and an extended star with two hop limit to the BS[2].

We decided to define the star as our topology so as not to incur complexity that would make us deviate from our goals. Moreover, as Natarajan's experiments seem to indicate, it is not clear that multi-hop is a better approach with current technology.

3.2.2 Dumb sensors

In our architecture we are assuming that all data processing occurs in the BS, the central component. Recall from §2.3 that nodes may do some local processing to infer information using their sensed data (e.g. Heart Rate (HR) from ElectroCardioGram (ECG)), but do not take decisions or use other sources apart from themselves to collect data.

BANs will, with some exceptions, be composed of nodes that are low in storage capacity, processing power and energy supply. Moving the processing to the more powerful BS enables the resource nodes to have more capacity for acquiring/actuating. Some energy is saved due to this off-line processing, but it is offset by the added communication. Therefore, in scenarios where the middleware may only need data from the node in some specific conditions, these conditions could be interpreted by the node. Unnecessary communication could thus be avoided, leading to an increased node life.

3.3 – Hardware abstraction layer

From this point forward, when we refer to middleware, we will be addressing both the middleware on the BS and on the nodes. When the discussion specifically applies to just one of those parts we will highlight this.

The hardware abstraction layer of the middleware is responsible for:

- allowing the control of the nodes, including (for sensors) their sensing services;
- allowing the discovery of the services provided by the nodes;
- hiding the details of the OS running on the nodes;
- providing a communication abstraction that hides the underlying network/link protocol (Bluetooth, 802.15.4, etc.);

[2]In the Natarajan et al. article [76] they used a variant of the PDR optimized routing that was limited to two hop paths. This was so as to lower the path size, that was getting to 2.57 hops average length. The restriction led to an average of 1.67 hops.

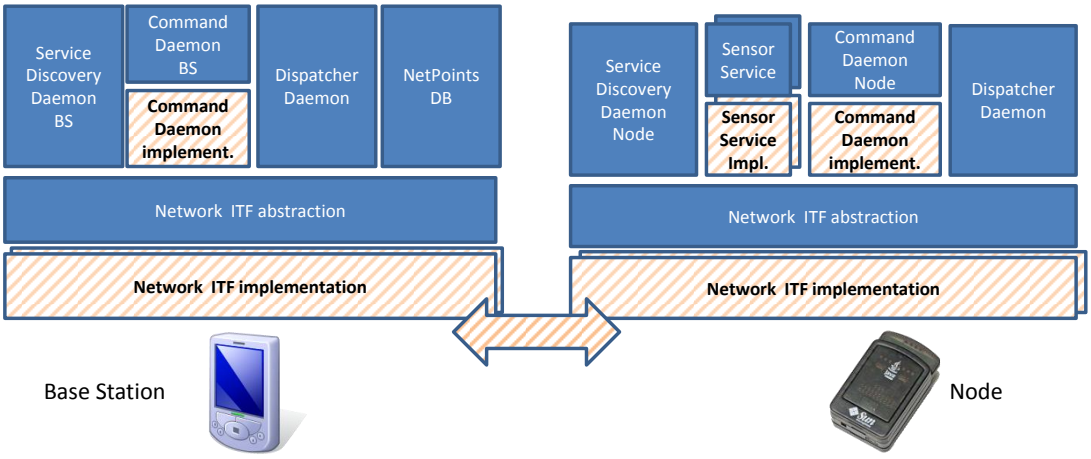


Figure 3.3 – Middleware components on the BS and node

  - defining the data structures and messages that permit the communication between nodes and the BS;
- receiving data from the nodes regarding the sensing data, handling the metadata received (time of collection, etc.) and deriving new metadata (delay, errors, etc.);
- allow easy deployment of new nodes (i.e. enabling a new type of node to be used within the middleware should not be a difficult job).

The components of the hardware abstraction layer are depicted in figure 3.3. Note that the BS does not have sensing services. This does not mean that the BS is not allowed to provide sensing services. It only illustrates the separation of concerns. These services could be instantiated if the BS has sensing capabilities.

The blue filled components from the figure are the generic components of the framework that are independent of the applications and the node devices. The orange striped components are specific implementations for the devices that deal with the intrinsic details of that hardware. Note that these implementations are per different hardware components and not necessarily different for every node. If, for example, a network interface implementation is developed for Bluetooth it could be re-used by every node (including the BS) that has a Bluetooth interface. The architecture follows the *bridge* design pattern to provide a generic abstraction for the network interfaces, the sensors themselves[3] and for commandeering the node.

The node representation in the picture is a SunSPOT [113]. Our development was mostly done on these devices (we discuss it in chapter 6).

3.3.1 Network

The network interface abstraction provides a generic way to use the network regardless of the underlying hardware instance (Bluetooth, 802.15.4, etc.). This provides an API that is then implemented for the particular network interface(s) of the node. The generic API allows sending of messages (broadcast or unicast) and waiting for messages (listen to). Messages are seen as data blocks from this API, and message classes provide easier access and message building functions to higher layers.

[3]Remember that a node may have more than one sensing capability. Each sensor service block controls one such instance.

An abstraction for communication with a specific node is also provided. This component uses the network API and enables direct communication with a specific node; again transparently to the real network interface that the node uses. This `NodeComm` component defines an API to send messages to the node it represents. The `NetPointsDB` from the figure holds all the `NodeComm` components that the BS knows about. Note that it is not a `NodeComm` per sensor/actuator, but per interface in a node, which may host several sensors/actuators. This way, requesting data from a sensor involves knowing which node it belongs to and then sending the request to the node. As the `NodeComm` knows the specific network interface to use, this also hides the network interface used to contact the node. The sensor/actuator being addressed is identified in the message sent to the node (see §3.5.2). A node may find it useful to have such a `NetPoinstDB` component if a multi-hop topology is used. In this case the node needs to know its neighbouring nodes.

The network interface component is also able to handle message fragments. This is needed when the message to be sent exceeds the network's maximum packet size (Maximum Transmission Unit (MTU)). In these cases the component breaks up the message into fragments of the allowed size, i.e. the MTU, and sends each. On the receiving site it re-orders the fragments and reconstructs the datagram. Note that it only provides a *best-effort* strategy in that no control on received fragments is done, meaning that lost fragments imply a lost datagram.

A reliable message interface is also defined. The contract for this interface is that its implementation provides ordered, reliable delivery of messages, by setting up a connection. This could be used, for example, for instructing an actuator for medication administration where correct delivery and avoiding duplicated messages is relevant. On the other end, monitoring a HR can miss *some* measurements.

The abstracted network interface does not provide flow control for messages sent. This is not *per se* an issue, as the case when this is most relevant is when receiving measurements from sensors. As we will see, the sensor services allow us to set rates for collecting and sending these measurements. This enables the information layer to have some control on the flow of data.

Although the network examples we have given so far (Bluetooth, etc.) point to the network abstraction being "situated" at level 3 of the IP stack, this is not mandatory. The network abstraction expects to have implemented: message sending (unicast and broadcast), getting addresses of endpoints, listening to messages and setting up reliable channels. This can be done using any layer of the IP stack (for example on top of the transport layer). However, in most cases what is needed is an implementation for a specific link layer interface. This also implies that in most cases the MAC layer will only be used and not changed or circumvented. But again, it is up to the developer of the network implementation to define that.

This flexibility leaves some room for using multi-hop technologies as we discussed above. If the network implementation either relies on a network layer that handles multi-hop or adds it in the implementation, the middleware transparently runs on a different topology. Note however that some of the profiles we discuss further may be affected by these intermediate nodes in the path. Optimization will also not take into account these intermediate nodes.

As expected, for communicating between the node and the BS, the same type of network interface must exist on both.

All the functionality defined is available for both nodes and BS.

3.3.2 Daemons

We use the term *daemon* to mean a component that is able to process input as a separated part of the main process. The usual implementation would be in a different thread from the main thread, which is supported in the JVM of SunSPOTs. In nodes where their OS does not support this, an implementation that allows for different call back functions should be supported. For example, TinyOS [121] allows for split-phase or non-blocking operations. In this OS, there is also the possibility to use "tasks" that enable longer running code to be "posted" to run in parallel[4].

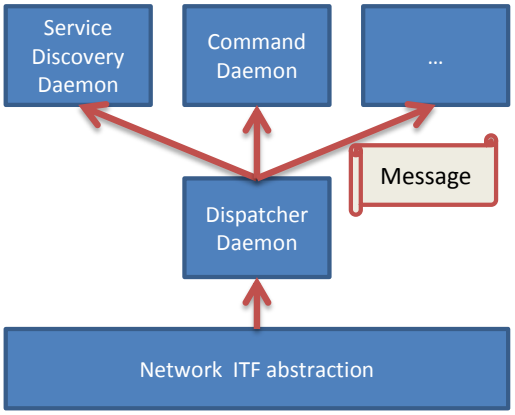


Figure 3.4 – Dispatching of received messages

In an effort to modularize the treatment of different types of messages we define different components to handle different parts of the protocol, which also allows for an easier expansion of the protocol. As such, we define daemon components as responsible for handling messages for a specific part of the protocol. There is a *dispatcher daemon* that receives all the node's messages and (as the name implies) dispatches them to the daemons responsible for the specific types. Currently there are two protocol daemons, one for the controlling nodes and the other one for service discovery.

From figure 3.4 we have:

Dispatcher Daemon just inspects the message type and, according to the registered daemons it knows about and the message types they handle, forwards the message to one of them.

Command Daemon is responsible for handling control messages. It handles and issues requests for data, setting rates for *collection* and *information sending* and also for turning the node on/off. As part of handling requests it also deals with the replies by interfacing with the Sensor Service portrayed in figure 3.5. These replies carry the measurements taken by sensors. On the BS side this daemon issues commands for the nodes. On the nodes, it replies with measurements and sends acknowledgements to commands.

Service Discovery Daemon handles the service discovery part, as expected. As is easily perceived, this an important component in a dynamic environment such as a BAN. The SD allows for nodes to advertise their capabilities and for the BS to query nodes about capabilities needed. We discuss it in more detail in §3.6.

From figure 3.3 we can see that daemons run on both the nodes and on the BS. The `DispatcherDaemon` has the same functionality on nodes and the BS. We discuss the `ServiceDiscDaemon` in §3.6. The `CommandDaemon` on the BS issues commands and receives measurements (sent through command messages). On a node it does the reverse, receives commands from the BS and sends out measurements. Acknowledgements are sent for commands received and single measurements by the nodes and the BS respectively.

The daemons described, as expected, handle the defined protocols. The information abstraction layer uses them to query for data types (service discovery daemon), control the nodes (command daemon), etc. On receiving these calls, the daemons issue the necessary messages

[4]From TinyOS 2.1.0 there is a TOSThreads library to enable the use of threads in TinyOS.

for the requested action. Notifications also flow to the information layer as measurements are received, nodes are discovered, etc.

The message dispatching from the network abstraction to the specific daemons involves sending to a queue that is managed by each daemon. As in the network abstraction no flow control is provided and daemons *quickly* process the messages in the queue, posting the longer tasks to run either separately or later.

Note also that some daemons may need specific implementations depending on what device they are running. The command daemon is such an example, as it hosts the functionalities to turn on/off the devices (we discuss sleep patterns and power cycles in §3.9.4). These functionalities are specific to a particular node's OS and as such need a particular implementation. The SD, on the other hand, does not use any specificity of the hardware. It only resorts to the network interface, which is already abstracted.

3.3.3 Sensor services

A sensor service is the abstraction of the sensing capabilities of a given node. In figure 3.5 its relationships are illustrated. The sensor service abstraction provides an interface to collect sensed values. It allows setting the frequency of collection (e.g. take a sample every 5s), through `SensorCollect`. The frequency at which collected data is sent (e.g. send every 22s bulk data collected every 5s) is controlled through `SensorSend`. The collection is node specific and as such needs to be implemented for the particular sensor in the node. These sensor services are associated with a sensor profile that describes what the sensor senses, in what units, with what limits, errors, etc. These profiles are discussed in §3.5.1. The command daemon can access the sensor services to be able to control them.

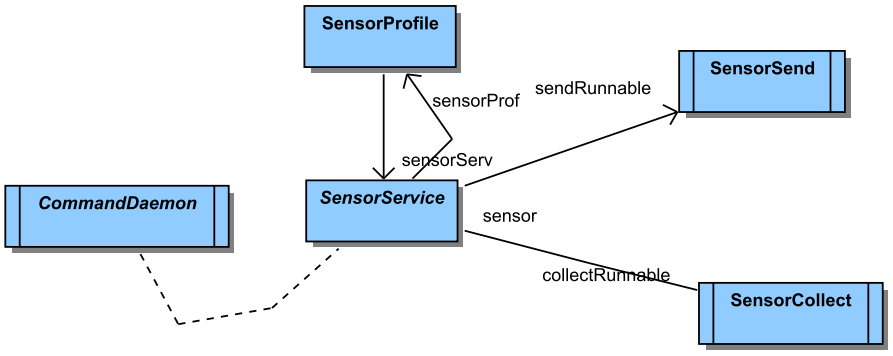


Figure 3.5 – Command daemon relationships

3.3.4 Active components

In figure 3.6 we illustrate how the components *live* in the node/BS. The diagram is a Unified Modelling Language (UML) collaboration diagram where the connections mean that information is flowing between the components. Components that are active (able to process input in parallel as discussed in the beginning) have extra vertical lines on the sides (e.g. `SensorSend`). The figure can represent a node or a BS (where the sensor services would not be instantiated). Every network interface is instantiated as an active `NetworkItf`. Messages received in these interfaces are forwarded to the *DispatcherDaemon*. According to the message type, the dispatcher sends it to the appropriate active daemon. The `CommandDaemon` can set the collection and sending

rate using the interface from the `SensorService`. Note that only the `SensorCollect` and `SensorSend` are active. The information abstraction layer accesses the daemons as needed to request information (but not in the nodes).

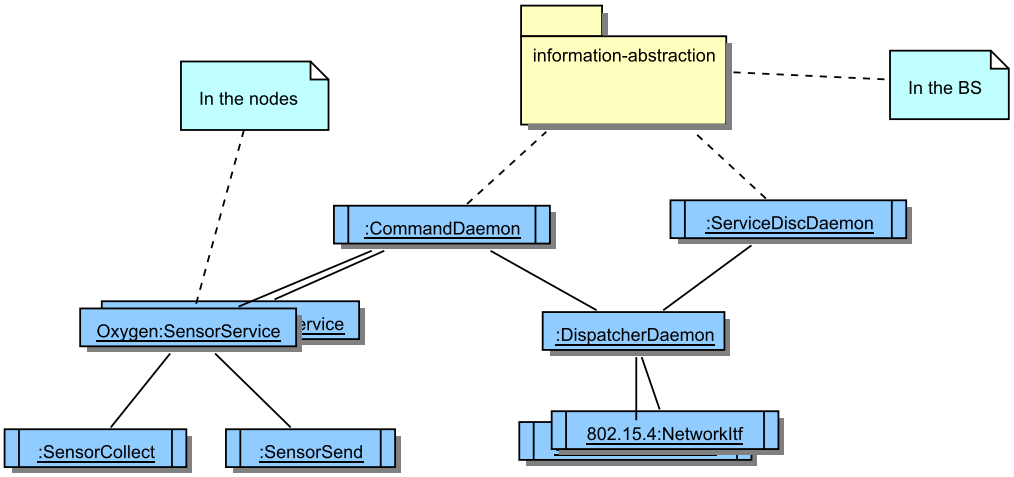


Figure 3.6 – Hardware abstraction active components

3.4 – Information abstraction layer

After the data collection from the lower layer, the information layer is responsible for using the collected data to *populate* the models used. This encompasses inferring new information as described by the models. The metadata received from the hardware layer (such as errors, delays, rate, time of collection, etc.) should also propagate through the models, and the rules for these should be defined in the model. These models can range from *tree-like* structures (similar to MiLAN [44]) or more complex ones using mathematical models of physiological processes (similar to the Physiome project [47]).

The model to use should ultimately allow the middleware to: **i)** unequivocally identify the information to be managed and its source(s), **ii)** infer commonalities between different models so as to optimize resources, and **iii)** clearly interpret how the metadata correlates in the model.

The optimization process relies on the models to calculate how the applications' requirements can be fulfilled using the resources in the most efficient way possible. Point ii) is important for optimization as it means detecting different applications using different models that need the same information somewhere in their models. This allows information to be acquired only once using the stricter requirements. We detail the information abstraction layer in the next chapter.

3.5 – Data structures

Several data structures were defined to allow for the abstractions described. They were based on input from the following standards: IEEE 11073 [52], Bluetooth [13], ZigBee [144] and SensorML [16][5]. Note that this does not mean we are compliant to any of those standards.

[5]Example: the units (see figure 3.7) used are the ones defined in IEEE-11073.

The basic idea was to reuse already defined structures without being overwhelmed with the complexity of fully implementing the standards. We also adapted them to what we believe are the needs of a BAN.

3.5.1 Profiles

We defined **Node Profiles** that state what a regular node (following that profile) has in terms of capabilities, namely: storage, network interface(s), and sensor(s). **Sensor Profiles** define the hardware of the sensor and the data provided by the sensor. In figure 3.7 we show these relationships. These profiles have defaults that serve as global knowledge for the BS (e.g.: stating the profile for a SunSPOT implies the presence of a temperature sensor, light sensor, accelerometer, 512KB RAM, ARM processor, etc.). These can however be customized and advertised differently by the sensor node, thus providing flexibility to accommodate variability. All the types and profiles described have a *wire representation* that defines how they are transmitted in the protocol messages. A code is also a part of these profiles to uniquely identify the profile being referred to, as an example the value `2` is the code that identifies the node profile of a Mica2 (another commercial programmable node).

The profiles are:

Data Profile defines the data that a sensor is able to produce. It defines the type of information (blood pressure, temperature, etc.), the structure to transport its value (integer, text, sequence of, etc.), and its units. There are attributes that are instantiated according to the node where the profile is being run, namely a Universal Unique IDentifier (UUID) that identifies where the data came from. UUIDs are used in Bluetooth to uniquely identify source points. They allow us to identify what data service is provided and on what node. This profile is also used in the information layer to describe the data produced in that layer.

Sensor hardware Profile defines the hardware of the sensor. It describes its accuracy, latency time (delay between sampling and availability of the output), integration time (time to assess/measure the phenomenon being sensed), maximum and minimum rate of sensing. These parameters are part of the *Detector* type in the SensorML specification [16, Annex C.].

Sensor Profile defines the sensor, with the data profile and hardware profile.

Node Profile defines the global characteristics of the node: processor, memory and storage, power, OS, network interfaces and sensors (profiles) it has.

The accuracy of the sensor hardware profile is given by a structure that describes the sensor's error in readings. The error can be constant within the sensor's range of readings, a percentage of the reading, or given by an error curve. This error curve is not defined in this error metadata structure, the structure only indicates the curve used as a 15 bit entry (enabling reference to 32768 curves). The curve itself should be defined elsewhere.

Measurements

Measurements are data structures that specify the time the measurement was taken, from what node, for what data type and the reading itself. They relate to a data profile by incorporating the UUID of the data profile, and thus defining the node and to what type of data it relates to, figure 3.8 illustrates this. The data type specifies its encoding on messages and has an untyped value so it can hold any data type. The time type has units for the time and if it is an absolute

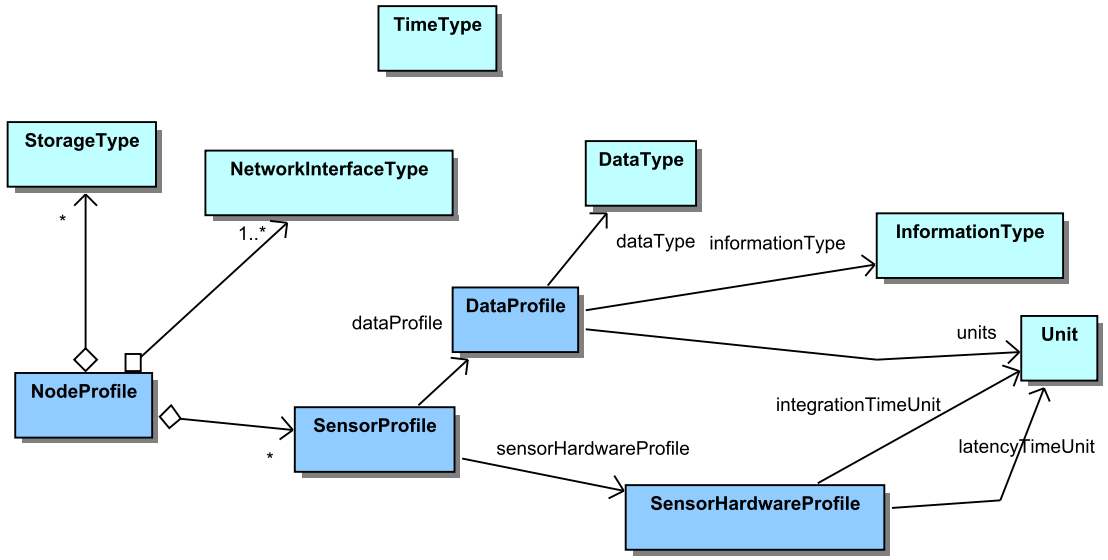


Figure 3.7 – Profiles and types

time or relative one, e.g. “how many milliseconds have passed since we started counting”. The relative time can have a high or low resolution, that is indicated by `timeRelHighRes`.

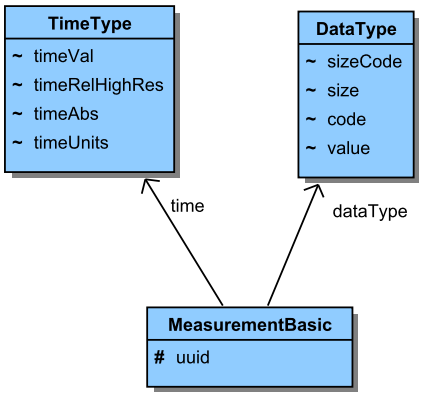


Figure 3.8 – Measure data structure and data type

The maximum *sensing* rate from the sensor’s hardware profile and the maximum *sending* rate from the interface types can be used to calculate the *timeliness* of the data. The sensing rate gives the period at which data is available to be sent regarding the phenomenon being sensed. The *sending rate* gives the amount of data that can be sent per unit of time. In most cases the nodes are able to send the data as soon as it is available, but in higher sensing rates and with more data to be sent this can be important. Note that measurements have a time of collection, which means that if only historical data is relevant, and not immediate alarms, the timeliness may be irrelevant. Note also that *latency* and *integration time* allow a finer detail on the time from phenomenon occurrence to data being available to be sent. The sensing rate should incorporate this delay (of both times).

Storage

As portrayed in figure 3.7 the node profile has one or more associated storage types. In some cases it may be preferable for the BS to let the nodes store the readings and either send them

later or for the BS to poll them. The storage type describes the type of storage (flash, RAM, etc.) and its size.

3.5.2 Messages

As an integral part of the protocol, we defined the messages for the interchange of information between the BS and the sensor nodes. In essence, the protocol is what enables the support of heterogeneous sensor nodes. As long as nodes comply with the protocol and use a network interface type supported by the BS[6] the communication is independent of the node's device-specific characteristics. The protocol mostly follows a request/response cycle and messages are defined for service discovery, commands and queries. Data read by the sensor is sent on reply messages. These messages are the exception to the request/response cycle. Nodes can stream (periodically send) measurements after being instructed to do so, where all the readings result from one request from the BS.

For SD there are query and advertisement messages. Command messages allow the setting of rates for collection, sending, turning the node on/off, requesting and receiving readings. All messages have a code that identifies their type and a unique identifier of the message. The latter is used to match requests with replies and queries with advertisement answers.

We detail the SD messages in §3.6.

Command messages

As stated, the command messages are used to:

Change rates: the BS can request to change the rate (start/stop included) of collection of sensing data for a specific sensor on a specific node. Parallel to collecting, we can set the rate of sending. This means that the node may need to store measurements and send them in bulk. When these rates are set the node issues, at the specified send rates, a reading reply message (bulk or single accordingly). In table 3.1 we have the possible combinations for both values. The specific sensor to address in the command is identified by the UUID in the message.

Change node's state: instruct the node to turn on/off. This implies that the node is able to sleep, saving energy, and check for new messages periodically.

Requests: the BS issues requests for specific data identified by the UUID. The BS can also issue a bulk request, where all the data currently held by the node for that UUID should be sent. This allows data to be obtained not only at periodic intervals (by using the rates above), but also when needed.

Replies: these are messages that hold the measurements collected. They can be single readings (holding only one measurement) or bulk replies (holding several measurements). As the sensors may be sending readings periodically, these messages only require one request from the BS setting the collection and sending frequency.

Messages that are *directed* to a specific sensor within the node (change rates and requests) carry the UUID of the sensor.

All the messages are acknowledged, either by an explicit ACK message or by a response to a request. The exceptions are the periodically sent reply messages. These are not acknowledged

[6] This means that the BS must have an interface for each different interface on the BAN.

Table 3.1 – Collection rate versus sending rate

	Collection	Sending	Relation	Action
1	undefined	–	–	Do not send or collect
2	defined	undefined	–	Send immediately after collecting.
3	defined	defined	col $\leq$ send	Assume error. Proceed as 2nd row.
4	defined	defined	col>send	Store the collected values and send them at the stated send rate.
5	defined	defined	send==0	Only collect and store. Assume that a specific request will be sent.

back to the node. The assumption is that they do not need to be resent if lost. A *not-ACK* message is also defined to signify that the message was received but the request/command was not done. A *reason* field is part of this message.

Replies

Reply messages are the ones that carry the measurements with sensed data. Note that if different data is requested on the same node, each different data would be sent (even if in a bulk) in different reply messages. There is no provision for sending different data profiles in the same reply message. Apart from identification of the data, this would raise problems for different rates of sending.

When the BS sends requests for the node to periodically send some data, the node sends reply messages with the measurement at the defined period. These messages are seen as replies to the first command and are not acknowledged, as we described earlier. This may lead to measurements not being delivered. If it is important not to loose any data, a reliable connection should be established for sending these data. As mentioned in §3.3.1 a reliable connection will imply reliable and ordered message delivery [7]. Note that the interface for explicitly requesting no losses is not yet defined. This should be part of the requirements of the requests, leading to a choice of a reliable connection if the application stated that losses were not allowed.

Reliability could also be developed within the middleware itself by providing acknowledgements in the network abstraction. This would mean storing the measurements until their reception was acknowledged by the BS. This could be used where no reliable connectivity was offered by the underlying OS. Non-reliable connections can be used when some measurements can be lost and minimizing delay and/or message overhead is more relevant.

3.5.3 Final comments

Figures 3.9 illustrate data types and a reply/request message. Message codes denote the message type (single request, change of rate, query discovery, etc.). Messages also have an identifier of 16 bits. In figure 3.9.a we have a request for a single measurement identified by the UUID; the message should be sent to the appropriate node. Message 3.9.b, if the collection and send rate are as row 4 from table 3.1, instructs the periodical sending of messages with bulk measurements. These replies with measurements are illustrated by figures 3.9.c and 3.9.d. Note

[7] Note that in unreliable connections, messages may be delivered out-of-order. In these cases, measurements' time stamps may be used to re-order data.

that the UUID and the data type are the same for all the measurements and are identified in 3.9.d. The time for measurements is an absolute time (denoted by `01`).

code	id	UUID
`00011110`	*16 bits*	*24-136 bits*

3.9.a: Request single measurement

code	id	UUID	rate col	rate send
`00010111`	*16 bits*	*24-136 bits*	*32 bits*	*32 bits*

3.9.b: Request for changing data collection and sending frequencies

code	id	req id	measurements
`00100011` *8 bits*	*16 bits*	*16 bits*	*variable*

3.9.c: Reply with bulk measurements

# measures	UUID	data type	value	time type	time value	...
8 bits	*24-136 bits*	*8 bits*	*variable*	`01`	*62 bits*	...
			per measurement			

3.9.d: Measurements

Figure 3.9 – Command message examples

One should note that in none of the data structures described was there any mention of a body/user identifier. The assumption is that the communication is to be secured/authenticated using a side-channel. As such, only nodes within the body area or that have been instrumented by the user may join the communication network and thus the body area. We discuss this further at the end of the chapter (see §3.9.4).

As mentioned, some of data structures we defined follow the IEEE 11073 standard for point-of-care medical device communication, namely we took into account the Application profile Optimized Exchange Protocol [54] and the Domain Information Model [53].Although, IEEE 11073 uses ASN.1 for data definition and Medical Device Encoding Rules (MDER) for encoding the data we opted to follow the encoding from the Bluetooth specification [13, Vol 3/Part B/ Section 3], as it was simpler[8] and equally capable for our needs. In figure 3.9.d the value is encoded using that specification. The encoding supports the regular types: booleans, integers (signed and unsigned), text, UUIDs and URLs. It also supports sequences and unions of these types. It lacks float encoding, so we added this using the IEEE 754 [51] representation.

3.6 – Service discovery

A SD service is the building point for a middleware system to be able to be aware of and control the resources underneath it. These resources are not pre-defined and may change with time, as new nodes may be introduced and older ones may disconnect. Knowing the resources

[8]The Bluetooth encoding has some similarities with MDER, but it uses a more straightforward approach to the encoding, with a small sacrifice on encoding size.

under its control is what enables the middleware to answer the requests of the higher layers while optimizing resources' usage.

In our discussion most of the examples are based in a sensor world, as this was our main focus. Nonetheless, most of the points can easily be transposed to an actuator environment.

In our view a SD system should have the following requirements:

Query capability: the query mechanism should be flexible to allow generic matching of capabilities;

Profile flexibility: it should be relatively easy to introduce a new profile description, to be able to advertise a new capability;

Overhead: the overhead added should be kept to a minimum;

Lower Layer Interaction: if lower layers provide functionalities that facilitate or incorporate SD responsibilities, the SD should build on those (eg.: notification of a new node);

Energy aware: the SD should be as power efficient as possible so as not to increase exceedingly the node's energy consumption. This relates to processing associated with SD and message exchange.

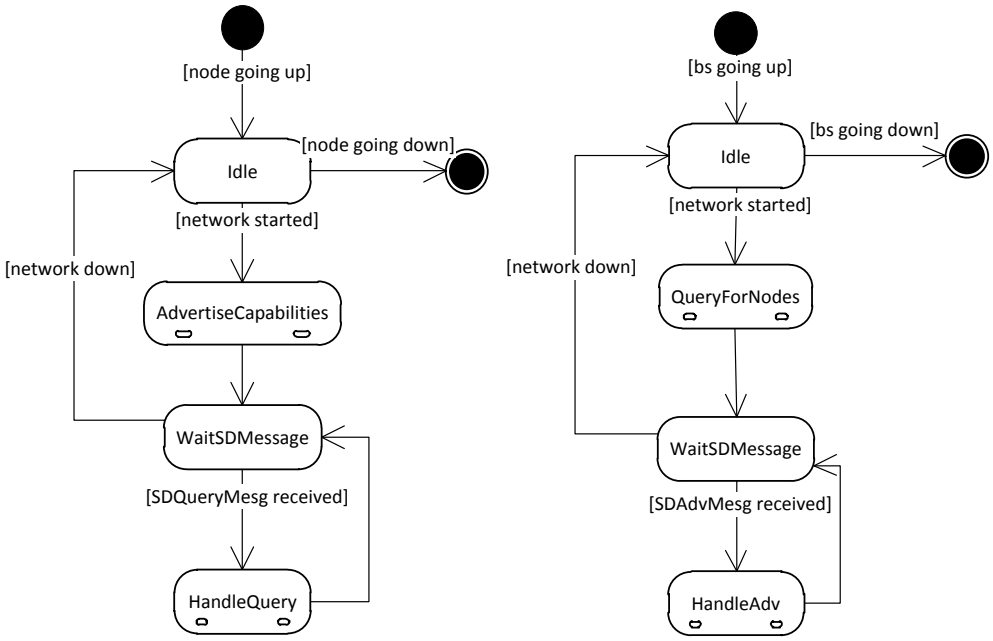
The aim of the proposed SD is: **a)** enable the advertisement of a new resource added to the network, i.e. a new sensor node with its capabilities; and **b)** enable querying the system for current available information. As mentioned, applications query the system based on their information needs and respective requirements. The sensor nodes, on the other hand, advertise the resources they possess. The middleware mediates this, by discovering the nodes that can provide the data needed and transforming the requirements set by the applications to requirements for the nodes. Note that applications can request an information that is not produced by the sensor nodes. The middleware can collect the information available from the sensors and correlate it to produce the requested information.

Requirements set by the applications in the context of the middleware can be regarding: accuracy of results (information error, confidence interval), information rates, and delivery guarantees (real-time, amount of losses, delay). This is mapped to **resources** in terms of: measurement capability (sample rate, error), network interface (bandwidth), energy (consumption, remaining) and the type of data sensed. The middleware is responsible for mapping the said requirements to resources. We use *capability* to mean the said resources, which are mainly measurement capabilities.

A proactive node advertisement is needed in the context of the middleware. This enables improvement of the fulfilment of application requests, by immediately making available new resources added to the system. An example would be a health monitoring application issuing a request for pulse rate to be delivered every 3s. The resources available only contained one node capable of a period of 5s. The addition of a new sensor for pulse rate (with its proactive advertisement) would enable the middleware to automatically increase the rate of the information provided to the application, thus fulfilling (or at least increasing the fulfilment rate of) the requirement.

The assumptions described earlier enable the SD to be simplified as only the BS has to maintain information regarding all the infrastructure and the sensing nodes only need to know about their own capabilities. Furthermore, only the BS sends queries to the network and does not need to advertise itself[9].

[9]There might be scenarios of communication between different BANs where the connection would most likely



3.10.a: In a node

3.10.b: In the BS

Figure 3.10 – SD global state machine

In figure 3.10 we define the **abstract global state machine** of the proposed SD protocol for the node and for the BS. The *start of the network* (device related) means that the node broadcasts an advertisement of its capabilities to the network and that the BS queries for available nodes. Using broadcasts suppresses the need for pre-configuring addresses. The node waits (with a time limit) for an acknowledgement from the BS after issuing the advertisement. After this initial step, the node waits for query messages and the BS for advertisement messages. If the BS misses the advertisement from the node, it will only be able to discover the node if it actively queries for nodes. Note that query and advertising messages are handled by the `WaitSDMessage` state of the node and the BS respectively. *Network down* is device related and occurs when the device turns its network interface off.

The BS issues queries if it currently does not know about a data type that is needed to fulfill an application request with its requirements. Although advertisements are acknowledged, the BS might not be aware of all the resources available. Advertisements not reaching the BS is one possible cause of this. Thus, issuing queries enables quicker and more thorough responses to the availability of data when the BS is unable to meet requirements.

Sensing nodes respond to queries and advertise their capabilities regardless of requirements. This implies that nodes state what they can produce at what settings. The BS is thus aware of what nodes can produce what information and with what capabilities. However, when trying to fulfil applications, the BS may only have partial matches regarding the requirements, which it will need to cope with. The BS can either combine partial matches or report the inability to fulfill all the requirements to the application.

An example of this combination would be regarding error on estimates for HR. Several sources

be done through the BSs. This would mandate that BSs would at least respond to queries regarding connectivity capabilities.

can be used to estimate HR, for example ECG, Arterial Blood Pressure (ABP), oximeters. Each has associated errors. If the application requested a higher level of accuracy than what could be satisfied by a single sensor, a combination of more than one should be done to accommodate the application requirement. This approach is used by Li et al. [65] to estimate HR. In chapter 4 we will expand on this.

3.6.1 Service discovery messages

The SD communication is governed by the following group of messages:

Queries allow data that is needed by the BS to be queried for. These queries can be for a specific data type, or they can be generic, allowing all the capabilities the nodes have to be queried. This is the initial query for nodes of the BS. All these queries are broadcast. They specify the code of the data profile needed.

Advertisements may be sent spontaneously or in response to a query. They refer to the node profiles described earlier by sending the associated code. For the nodes that stray from the default known node profile, the messages allow the complete profile to be sent. When an advertisement is sent in response to a query, the node could opt to just send the queried sensor profile. It is nonetheless advantageous to send the complete node profile, because if the sensor profile is missing from the BS's knowledge, it indicates that the BS does not know about the node.

Heart beats are used so that both BS and nodes know about each other's liveness. The BS sends heart beat requests periodically that must be replied to by the nodes. This means dead nodes to be removed from the usable resources of the BS. It also provides reciprocal awareness. Nodes expect these heart beat requests and as such can determine the disappearance of the BS. The existence of these messages precludes the need to have an explicit notification of *going down* by the node or BS; a message that could nonetheless be lost.

3.6.2 Other service discovery services

SD is an area that has been thoroughly researched. Some developments have focused on pervasive ubiquitous environments [34, 143]. The most common approaches gravitate around Service Location Protocol (SLP) [40], Universal Plug and Play (UPnP) [122], Jini [112] and the now defunct Salutation [97]. Although these approaches offer some insight, they lack the applicability to Sensor Networks (SNs)[10]. SNs have specific issues that warrant newer approaches, namely their: low energy powered devices, lack of computational capability (processors, memory), vast distribution, and a mainly wireless connectivity. This last point leads to two main issues. Wireless communication is usually the main *culprit* in terms of energy consumption as we saw in §2.5. The other point is that the environment in SNs is usually harsher for wireless communication than normal office/home wireless communications. In Wireless Sensor Networks (WSNs) the outdoor, unknown distances between nodes, terrain obstructions, terrain reflections are good examples. In BANs, as we saw in §2.4.1, the body is an obstacle with the added problem of water composition that absorbs 2.4GHz waves (used in 802.11, Bluetooth, 802.15.4). These problems usually lead to to a less capable connection, with limited bandwidth. As such, these restricted environments have lacked a fully fledged SD system. The wireless communication standards for low power devices have tackled this necessity by providing SD capabilities. Namely, Bluetooth [13] (with its Service Discovery Protocol (SDP)[11]) and ZigBee [144] (by adding service discovery

[10] The interested reader is referred to Ververidis and Polyzos [123] for a discussion of SD for mobile ad hoc networks.
[11] Bluetooth Low Power [14] uses the same SDP.

to the IEEE 802.15.4 communication standard) are the more widely used approaches in WSNs. Our approach *borrows* from these standards, simplifying some structures for BAN use and adding proactive advertisement by the nodes. The expected star topology also enables centralized control from the BS which eases SD.

3.6.3 Comments

From the requisites we stated for a SD service in the beginning of this section we can state the following for our framework:

Query capability: our system allows querying for the data type being sought. The advertisements enable the announcement of capabilities, which lead to divulging the available resources;

Profile flexibility: the profiles described offer detailed information regarding the node. They are easy to build on, but may be too verbose for a fresh start. When we designed them, we chose detail availability over simplicity. However, adding a capability to a node profile is just a matter of defining a new sensor hardware profile and a new data profile (or make use of one already defined). The profiles' framework also enables easy expansion of the profiles currently available by customizing some of their parameters;

Message overhead: we mitigate overhead by using a unique code to reference profiles in messages, as opposed to sending the full profile. Nodes only need to know about their own codes, but BSs need to know about all the codes they might encounter. If a node needs to send the full node profile detail, it will incur a higher overhead (from 40 bits to around 424 bits, depending on the node's capabilities to announce). This overhead is nonetheless small;

Lower Layer Interaction: this interaction is more intertwined with the implementation, where the discovery of new nodes could come from the lower layer. This has not been defined in our SD protocol and would depend on the implementation;

Energy aware: we tried to minimize the message exchange by only having messages being transmitted on boot-up and when data is needed by the BS that it cannot accommodate. The heart-beat messages are the exception. They are used for keeping the BS resource knowledge accurate. The frequency of these messages can be adapted by the BS.

3.7 – Adding a new node to the architecture

To better illustrate what our middleware addresses, in this section we describe what is needed to add a new node to the system. We describe the profiles to define and the services to implement so that the node can interact with the architecture and be recognized by it.

Starting with the profiles, we have to (recall figure 3.7):

1. **Add a new** `NodeProfile`**:** this entails defining attributes that describe the processor capabilities, available power source(s), available storage and network interface(s). Describing storage space involves stipulating the type and size of the storage. The network interface(s) involves stating the type (802.15.4, Bluetooth, serial, etc.), its maximum data rate and version. Then we need to add the node's `SensorProfile`s;

2. **Add a new** `SensorProfile`**:** a sensor profile describes the sensor hardware and the data profile of what is produced by the sensor, respectively the `SensorHardwareProfile` and the `DataProfile`;

3. **Add a new** `SensorHardwareProfile`**:** this describes the minimum and maximum rate of collection, the time it takes to have the data available and the error curve associated with measurements;

4. **Add a new** `DataProfile`**:** the data profile is associated with a data type (what is transported in messages) and an `InformationType`. The unit in which the output is given is specified (e.g. ºC (Celsius), ºK (Kelvin)). It also holds the UUID, which is instantiated per node given the code of the data profile, the code of the service profile, an access point (allowing more than one sensor/data profile in a node) and one of the node's address.

5. **Add a new** `InformationType`**:** the information type describes the data in the data profile. It defines how to interpret the value (a count, a quantity, a category, etc.) and a possible range, which may be an interval (e.g. $[-20, 50)$) or a set (e.g. $\{low, normal, high\}$). The range is defined globally per information type, but can be specialized for a specific sensor (e.g. a temperature sensor that only measures in the range $(0, 40]$ ºC, where the information type for temperature sensor states $[-20, 50)$).

Note that adding a new node may not imply defining all these profiles as they may already be defined (e.g. the network interface for 802.15.4 may be used in several nodes, the temperature sensor ADT7411 ADC for the SunSPOT may be present in other nodes, the environment temperature information type will be used in all the environment temperature sensors).

In terms of functionality, we would need to implement (recall figure 3.3):

1. **Network Itf:** this would amount to sending and receiving messages (broadcast and unicast), retrieving network related addresses and setting up reliable connections;

2. **Sensor Service:** for each of the sensors in the node an implementation of the `SensorService` would need to be done. This basically would be implementing the read operation for the sensor with an output conforming to the `DataProfile` defined;

3. **Command Daemon:** the command daemon would need to build the sensor services, turn on/off the network interfaces and change the state of the node (on/off).

As we mentioned for profiles, re-use of these implementations for the same type of components is possible. So if an implementation for the 802.15.4 radio interface was already defined for a SunSPOT we could re-use it in a Mica2 node.

Implementing only these three components assumes that all the abstraction already defined (the blue-filled boxes from figure 3.3) could be used on the new node. We discuss this when we address the implementation in chapter 6.

3.8 – Other middleware architectures

Tackling the gap between applications and sensor hardware is still not specifically addressed in BANs. Some middleware approaches address data distribution issues with greater detail than the

abstraction of information. This is the case with CodeBlue [102], a framework that addresses the mismatch between the available sensor networks and the information requirements of medical care. For this purpose, it combines hardware and software to allow for discovery, communication and dissemination of information directly from the sensor devices. In the software platform they employ a Publish/Subscribe (pub/sub) paradigm, such that information is multicast to interested parties that request the information. The system also allows for direct querying of information through a simple interface, which is similar to directed diffusion [57] and permits the specification of data rates. It considers access to individual sensors. CodeBlue is focused on the communication infrastructure and the data dissemination process among "body nodes". In other words, the network within the body is not specifically addressed and applications still need to query for the lower level data. Also, resource consumption and optimization within the BAN itself is not considered, as, in the proposal, it is assumed that most of the nodes can easily be recharged.

A more directly relevant approach is the work from Waluyo et al. [125] in Mobisense. It shares similar objectives to the ones proposed in this chapter, namely data acquisition, support for multiple sensors, plug and play functionality and resource control and management. The target is Body Sensor Networks (BSNs) and they have a prototype for classifying position, based on accelerometer data and Kalman filters. The prototype also handles ECG data. The system addresses encryption in the communication between the sensor and the BS. It also handles possible re-configuration of the system. The middleware on the sensors is a thin layer, that mostly handles the commands from the BS and sends data. It also has a power-saving mode with a *critical issues self-awake* detection. This last point works with thresholds set by the application, that trigger the node to come out of sleep mode to resume sending data. This implies that the sleep mode is mainly turning communications off, with data still being collected by the sensor node. They specifically state that no buffering is done to collected data so as to ensure timeliness. For resource management the middleware gives applications with an API to toggle the sensor node, set it to go to sleep mode (communication off) or get the battery readings. Although an API is provided, the applications still need to manage and control these functions. As such, operations like optimization of resources are the responsibility of applications. There is no mention of support for running multiple applications on top of the middleware in this article [125], although their first work did aim at that [124].

In the WSN realm, Middleware Linking Applications and Networks (MiLAN) [44] tries to cope with "*the gap between the protocol and the application [which] is often too large to allow the protocols to be effectively used by application developers*". The authors address this by tackling what they define as the features of sensor applications: distribution, dynamic availability of sensors, constraint application Quality of Service (QoS) demands, resource limitation (bandwidth and energy), and cooperative applications. This last issue relates to different applications using the same network to achieve different objectives; they must cooperate or at least not *step on each others' toes*. MiLAN tries to cope with different application requests using their QoS requirements as input. It also takes into account the network information (energy and bandwidth) and the system's information on the relevance/precedence of the different applications. The authors also propose a middleware component that resides on the network stack, so as to take into account and control network properties. The main objective is to vary the network parameters over time and maintain the QoS needs of applications. They also define a tree-like relationship for deriving data from the sensor observations. As that is the theme of the next chapters we come back to MiLAN there.

The layer we discussed in this chapter does not introduce many novelties in terms of what a usual OS, SD system would bring. However, it makes that abstraction available as *one package*. SD is provided regardless of the network used (Bluetooth, ZigBee, etc.). The hardware abstraction for the interactions proposed is also independent of the OS. The main objective is to provide an API/abstraction from what *lies underneath* as a stepping stone for the information abstraction to work with.

3.9 – Concluding remarks

Motivated by what we understand as the mismatch between applications that want to use BAN information and the current heterogeneity of hardware in BANs, we introduced in this chapter a middleware to abstract these resources to applications querying for information. Based on a set of assumptions for this type of network, we presented an architecture that gives applications the freedom to access the information they need while complying with a set of requirements and optimizing resource usage. We detailed the lower layer of this architecture, defining what we called the hardware abstraction layer. We introduced the information abstraction layer that is refined on the following chapter. This layer is responsible for the optimization and requirements compliance using the first lower layer to achieve this.

With this first layer we are able to address the following objectives set forth in the introduction:

Data collection (A): the protocol described enables requesting and receiving measurements from sensor nodes;

Answer applications (D): the SD service allows querying for the data needed to fulfill requests. Measurements received by applications include metadata regarding the data collected. We have not yet described correlation of information, so this objective is not yet accomplished for retrieving the full application request;

Optimize resource usage (E): the described layer provides the interface to control the nodes and a description of the node's capabilities; the next chapter uses it for the optimization;

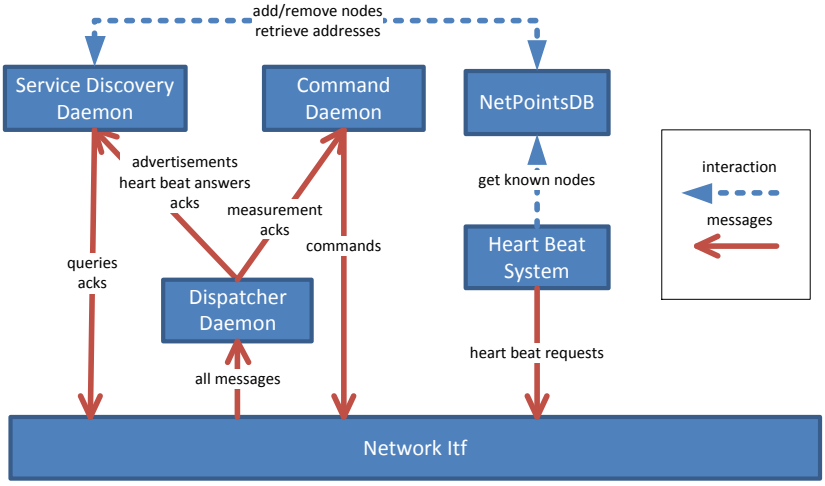
Advertisements (i), Commands (ii), Answering requests (iii) and node control (4): the nodes are able to advertise their capabilities, receive and process BS commands, and answer requests from the BS. The BS can control the nodes.

As a *pictorial* summary we have in figure 3.11 the flow of messages through the components in the system. As noted the components act differently when they are in a node or on the BS.

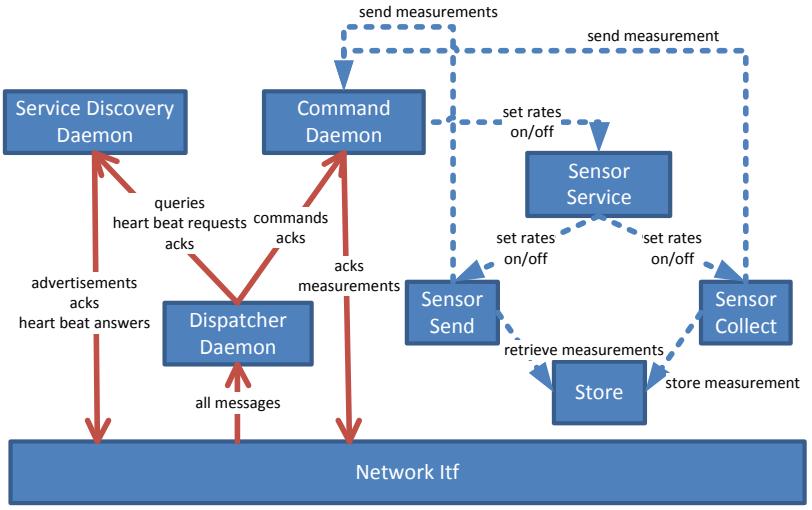
In figure 3.11.a we have the BS where measurements and advertisements are received, and queries and commands are sent. The BS also issues heart-beat requests and deals with the answers, or the lack thereof. After discussion of the information abstraction layer in chapter 4 and the information flow in chapter 5 we complement this diagram with the information layer on the BS.

In figure 3.11.b we have the node where queries and commands are received, and measurements, advertisements and heart-beat answers are sent.

This layer is used by the information abstraction layer, and not necessarily directly by applications. Some particular points still to pursue follow in the next sub-sections.



3.11.a: In the BS



3.11.b: In a node

Figure 3.11 – Message flow and component interaction

3.9.1 In-node processing

As we pointed out in §3.2.2, we assume dumb sensors. Other approaches propose that processing should be done on nodes to save energy. Our framework partially allows that option. The sensor services described in §3.3.3 can embody more functionality than just reading raw data values. One could for example, have a service that provided a moving average of the last 10 seconds of the sensed value. For this a sensor profile and a sensor service implementation should be defined to provide the functionality. It could rely on the abstraction for the collection, sending and command interface. Note that the provided abstraction only allows for setting the collection and sending rate, and to request a current sensed value. As such, it would not allow both the average period and the sampling rate to be configured.

Another opportunity for in-node processing is when collection rates greatly exceed sending rates. In these cases, compressing the data before sending it optimizes communication usage. There would be two options to accomplish this using our framework. The first option would

be similar to the moving average example. We would implement a sensor service that would compress the measurements still in storage. As before, a sensor profile would be needed so that the middleware on the BS would know about the need to decompress the data. Another option would be to define a new network interface that would compress the data before sending it. It would then use one of the physical network interfaces of the node. Similar to all network interfaces, it would be implemented in the BS and the node.

Our approach is to have this *intelligence* in the BS and for the node only to provide the raw data. Nonetheless, in some cases, it may prove useful to spare unnecessary communication channel usage, thus saving energy. In these cases processing on the node may prove advantageous.

3.9.2 What about actuators?

Most of the examples given were based on getting data from nodes, i.e. focusing on sensor nodes. The framework is designed for accessing sensor nodes, but what was described can also apply to actuators.

Profiles would apply to actuators, but some additions would be needed for full use of actuators. Profiles would need to define what the actuator can do; they would define the *degrees of freedom* of the actuator. Global actions would need to be defined such as: turn on LED, supply medication, apply defibrillation shock, etc. Some would need finer detail, e.g. *apply 0.1 units of insulin during 15 minutes*.

Advertisement of capabilities are also possible. The node profiles would be advertised similarly, and their *heart-beats* monitored in the same way;

Control of an actuator is mandatory. The current command messages would need to be extended to allow finer control of the actuator. They would use the expanded profiles mentioned above. This would assume that the actuators themselves would have the appropriate system control loops (e.g. instructing a pump to inject 0.1 units of insulin would mean that the device knew how to control the pump so that exactly 0.1 units would be delivered);

Optimization would apply partially to actuators. However, in actuators we would not see much replication. Also, as we will see, one of the optimization principles would be to try to derive the same information through less costly means. With actuators this would be less doable, as there would only be one way of doing a specific action (e.g. for cardiac resynchronization one would have only one Implantable Electronic Cardiovascular Device (IECD)). Even if multiple actuators for the same purpose existed, the objective would be to actuate a precise one. As an example, if several lights existed for warning indicator, we would want to turn on a specific one for the warning; it would not be appropriate to light one at random. Sensors of the same type will be more amenable to being used regardless of which specific one is addressed.

Actuators would also provide new challenges in the model management for applications. These application models would derive actuations based on the information received from the middleware. They could be *moved* to the middleware, letting it do all the control and actuation.

3.9.3 Virtual Nodes

The architecture allows virtual nodes to be deployed on a BS to test node interaction or new implementations. As the architecture follows a modular software abstraction approach, it is simple to instantiate a node as software as long as it obeys the stipulated protocol. It would need nonetheless, to have several components. It would need a network interface that is accessible

to the BS. Any Inter Process Communication (IPC) or a real network interface (e.g. loopback) could be used as long as the BS also had the implementation. A service implementation would be the most natural component to test, so this would be developed to run on this virtual node. It could either fake outputs or derive them from the available hardware on the BS. The command daemon and other daemons would run as normal using the implemented network interface. The implementation of the command daemon should take into account that the node is running on the BS and as such should not turn off the full BS.

3.9.4 Open issues

When we described profiles we mentioned that there is no user identifier associated with the profiles. The assumption is that nodes use a side-channel to authenticate with the BS and only nodes that are authenticated can interact with the BS. This makes nodes part of the BAN only if they can authenticate, making them all from the same user.

This approach shifts the responsibility to the authentication mechanism that must accommodate this. Approaches could include hard-coding a credential on the device that would be authorized by the BS, entering a PIN on the BS that relates to the nodes discovered, or just confirming the node join of the BAN when it is detected. Other, automatic approaches, involve a channel or knowledge sharing that can only be used/known by a node physically within the BAN. Some approaches use BCC[12] to identify nodes in the body, like the work of Falck et al. [31]. Other approaches use biometrics to generate keys for the authentication: Hanlen et al. [42] use body movement as a common source of randomess for key generation and Poon et al. [91] use ECG and PhotoPlethysmoGram (PPG)[13] as a source of entropy for generating a key for communication.

We have not yet studied these approaches and the protocol is still lacking a phase for identifying nodes within the BAN. As can be perceived, this is relevant not only in terms of identification of the correct nodes to use, but also for security and privacy.

Time stamps in measurements imply a global (within the BAN at least) time definition. As such, sensor/actuator nodes should be clock synchronized with the BS. This synchronization is not part of the middleware, although this should be its responsibility. Options for synchronization such as the work from Elson et al. [29] should be incorporated into the middleware. Some network standards, such as the current draft from Institute of Electrical and Electronics Engineers (IEEE) 802.15.6 for BAN communication, define a clock synchronization procedure between nodes and the hub. Investigation of its usability within the middleware should be taken.

Another point not addressed, which is important for conserving power, are the sleep patterns for nodes. When nodes are not being used, or when activity periods are very sparse, they should go to a low power state. This would imply turning off most hardware components including communications. For this to be able to occur and for the BS still to be able to contact them, a duty-cycle should be defined. Regular *wake-ups* from the nodes should occur to check for requests. This would imply synchronization with the BS. Different sleep patterns could be defined where nodes keep some sensing being done to control for threshold cases, similar to the Mobisense platform [125], discussed in §3.8.

[12] In BCC the body is used as the medium for communication.

[13] PPG uses the intensity of light to measure the blood volume of an area. A pulse oximeter is a device that uses PPG to determine the oxygenated haemoglobin in the blood and changes in blood volume. This is what is used in the referred paper.

4

Modelling data correlations

The model we propose for multi-parameter monitoring and correlation is described in this chapter. We also discuss an algorithm to discover optimal resource usage while satisfying applications' requests. The flow of information within this model uses a Publish/Subscribe (pub/sub) system that we present in the next chapter.

4.1 – Introduction

The concept of multi-parameter patient monitoring is re-gaining momentum [36], as the possibilities for enabling it are getting more mobile, user-friendly, and ubiquitous.

The research for digital computerized systems that are able to assess different inputs from the human body can be tracked back a few decades. The experiment by Shubin and Weil in 1965/6 [104] of using a computer to monitor Blood Pressure (BP), ElectroCardioGram (ECG), body temperature, etc. and produce records for seriously ill patients (in a shock unit) is one such example. Using several different inputs to assess physiological parameters has been a developing effort, examples are: devices that monitor several parameters as in the AMON project [4], where oxygen saturation, acceleration, skin temperature, etc. are monitored by a single device; projects to correlate data to *clean* information as Li et al. [65] do to estimate Heart Rate (HR); the use of qualitative models of multiple parameters to derive new states as the work from Dawant et al. [26] where an adaptive patient monitoring system reacts to the environment and to a predicted evolution of the patient state; severity scoring systems in Intensive Care Units (ICUs) [109] use several inputs (over 100 in some cases, some of which are diagnostic parameters) to produce a "score of illness" for predicting outcome.

Correlation of information is a key factor in producing richer information and inferring new information. Be it privacy invading correlations [35], mathematically oriented models [93], or

systems physiology [60], tapping into multiple different inputs yields richer, more complete, newer information. Although these latter correlation examples may be beyond the main focus of our work, they serve as illustrations of the power of inference and correlation. Our intent is to enable correlation of "bodily" measured data.

Figure 4.1 illustrates possible correlations of different types of information. In the diagram (blue) boxes represent producers of information; they use the information from the (yellow) circles to infer new information (another (yellow) circle). As an example we can produce Cardiac Output (CO) from HR and Stroke Volume (SV), using our known CO formula $CO = SV \times HR$. Moreover, the figure shows that CO can also be produced by the *ImpCard Reader CO* (impedance cardiograph reader) or *ElectCard Reader CO* (electrical cardiograph reader).

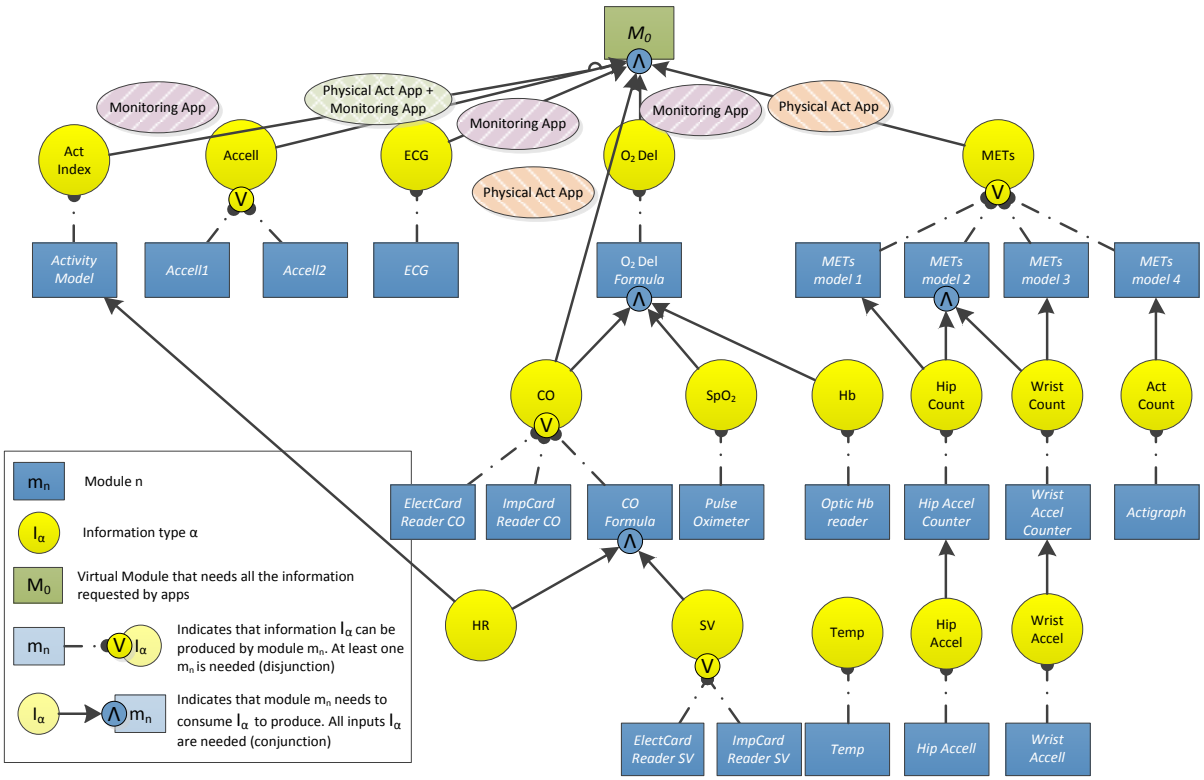


Figure 4.1 – Correlation diagram example

From the set of correlations and sensed data, the objective is to allow applications to access not only raw information from the sensors, but also information combined or inferred from the "raw" sensor information. This means accessing any (yellow) circle. The approach is to define a middleware that *hosts* how the information correlates (the model) and provides abstractions to the specificities of the hardware, communications and operating systems of the sensors.

The middleware then deals with the requests from the applications to determine how to provide the information requested. Support for several different applications requesting different information, where some of the correlations may be similar, is another aim of the middleware. In figure 4.1 ellipses represent applications requesting information. There is a *Monitoring App* for a post-operative surveillance and a *Physical Act App* for tracking exercise performance. As examples of similarities we have that both are requesting *Accell* and that the *Monitoring App* is requesting *O_2 Del* that is produced from *SpO_2* and *CO*. *CO* is also being requested by the *Physical Act App*.

The middleware holding the knowledge of what information is needed and how this informa-

tion is derived/collected helps in optimizing resource usage (the sensor network) while enabling all applications' requests. Our aim is to allow this multi-parameter correlation by using a human model where these correlations are described. Figure 4.1 is an example of such model.

An architecture that enables applications to input the model that they want to use, stating how information is correlated, provides a flexible approach. These models should be nonetheless framed such that the middleware is able to: **i)** unequivocally identify the information and its source, **ii)** infer commonalities between different models so as to optimize resources, **iii)** clearly interpret how the metadata correlates in the model.

4.1.1 Problem statement

The scenario we assume is: several applications running on the Base Station (BS) that request information in order to function (providing it to the user, for monitoring purposes, etc.). These needs may have requirements "attached", e.g. frequency of input, maximum error of the information, latency on getting it. There are costs associated with this retrieval, energy spent by a hardware node, processing power, error introduced by the sensor or correlation, etc. We discuss these metrics in §4.2.3.

To answer this, the middleware uses a model that defines how different inputs correlate to produce new information. There may be several ways to produce the same information. The middleware's objective is to optimize resource usage (the sensor system) while providing applications with the requested information. Our approach is to try to discover commonality between requests. To improve the chances of finding it, we seek to have correlation **modules** in the middleware to enable "intersections" between requests. All requests' requirements must be met, so the most stringent of the intersecting requests is the one that must prevail.

From figure 4.1, if $MonitoringApp$ requested $O_2\ Del$ with a frequency of update of 5 Hz and $Physical\ Act\ App$ requested CO with a frequency of update of 10 Hz, the middleware should calculate CO with a frequency of 10 Hz. This would satisfy the strictest requirement[1].

We propose an approach to: **(I)** compartmentalize the correlation between information in reusable software components; **(II)** define how components depend on other components (information needed as input) using a graph analogy; **(III)** optimize component usage by discovering intersecting needs; **(IV)** aggregate different requests so as to optimize according to a defined cost; **(V)** define an algorithm for the optimization.

4.2 – Model

In this section we describe the model approach we took and how to optimize resource usage using the model.

As mentioned, applications impose requirements on their requests. The middleware uses metrics to check if the requirements are being met. These metrics are also used for calculating the cost of using a specific configuration. This cost is the metric to optimize the system for. As we mentioned these metrics are frequency of updates, error, delay, energy consumption, processing, etc.

[1]We discuss in the final section the problems with this approach for frequencies of collection.

The control that is available to the middleware is which modules to use (the decision variables) for the correlations. The end result is a configuration of modules to use to meet the requests while minimizing the cost. This general definition is enough for the following subsections; we give further details in §4.2.3.

From figure 4.1 the middleware would need to compute which modules it should use so that $MonitoringApp$ and $Physical\ Act\ App$ would get the information that they are requesting, taking into account their requirements. $MonitoringApp$ needs activity index (a measure of the activity of the subject), acceleration, ECG, and oxygen delivery (describes the quantity of oxygen that is transported to cells). The $Physical\ Act\ App$ needs: acceleration, CO (measures the blood output of the heart) and Metabolic Equivalent (MET)[2].

4.2.1 Framework description

We represent information correlation in **modules**; they use **information** as an input to produce different information. In figure 4.1, (blue) boxes represent these software modules. The (yellow) circles represent information. All of a module's inputs must be available for it to be able to produce its "unique" output. This is represented by the arrows leading to the boxes (to a $\wedge$ indicating the conjunction). The output produced by a module is represented by the "slash–dot" line directed to an information circle. The same information may be produced by different modules. This is represented by the "slash-dot" lines leading to the information circle (and to the $\vee$ indicating the disjunction). The diagram is a Directed Acyclic Graph (DAG) (no cycles are allowed by design) with two types of nodes. The direction is indicated by the arrows and the circles at the end of the "slash–dot" lines, which is bottom to top in figure 4.1.

We use the text notation of $\overline{overline}$ to represent information nodes and $\underline{underline}$ for module nodes. Although both information and modules are nodes in the graph we refer to *information nodes* and just *modules* from this point forward. When we want to address both we use *nodes*.

As an example, the $\overline{O_2\,Del}$ information node can be produced by the $\underline{O_2\,Del\ Formula}$ module. This module needs $\overline{CO}$, $\overline{SpO_2}$ and $\overline{Hb}$ to produce $\overline{O_2\,Del}$. The top module, $\underline{M_0}$, represents the information requests made by the different applications to the middleware. As such it does not produce anything, but it needs all of its inputs, as it needs to answer all applications. From figure 4.1 $\underline{M_0}$ needs $\overline{Act\ Index}$, $\overline{Accell}$, $\overline{ECG}$, $\overline{O_2\,Del}$ and $\overline{METs}$.

These correlations embedded in the modules are either defined by a particular application or globally accepted by the community, the latter improves reuse. In figure 4.1, METs calculation from activity counts models 1, 2 and 3 are from Swartz et al.'s approximations [114] and model 4 is from Crouter et al.'s work [24]. Moody's [74] estimation for activity indices from HR is used in $\underline{Activity\ Model}$. CO is derived from SV and HR using the Windkessel model as described by Sun et al. [111]. We use the oxygen delivery model discussed by Law and Bukwirwa [64].

As can be seen in the figure, some modules do not need any data input (e.g. $\underline{Accell1}$). These modules represent the sensors, which produce their outputs from the physical world.

Some other observations (illustrated in the figure):

- as mentioned modules only have a single output;
- there is only one information node per information type in the model;

[2] MET is a measure to compare the metabolic expenditure of physical activities. It is a ratio to the resting activity, which has MET =1 [86].

- some modules and information present in the system might not be needed (e.g. $\underline{Temp}$ and $\overline{Temp}$);
- if a module lacks some (or all) of its input it is not able to produce an output, thus invalidating the modules that depend on the information it produces (e.g. no $\underline{HR}$ producing module is available, thus $\underline{CO\ Formula}$ and $\underline{Activity\ Model}$ cannot produce their outputs);
- the same physical sensor may have different modules associated with it (e.g. $\underline{Hip\ Accell}$ and $\underline{Accell1}$ are provided by an acceleration sensor on the hip);
- different applications may request the same information with different requirements (for example $\overline{Accell}$). Different requirements may also occur for information that is commonly needed (e.g. $\overline{HR}$);
- different modules producing the same output may have different requirements, cost and output quality (e.g. for the $\overline{METs}$ information there are four available modules with different accuracy on their outputs).

Application requirements mandate modules' requirements. A maximum error of 2% for the $\underline{O_2\ Del\ Formula}$ may impose a maximum error of 1% on $\underline{CO}$, 2% on $\underline{SpO_2}$ and 0.5% on $\underline{Hb}$. The influence of the inputs on the output of a module are defined by the module based on its correlation function.

This framework assumes that applications can let the middleware perform the correlations, where the definition of what and how it is done can be defined by the application developer, by deploying new modules. However there might be some applications that want/need the "raw" data for specific control reasons. As an example we have the closed-loop insulin delivery for diabetic treatment as discussed by Renard et al. [95]. The framework does not preclude this, as this would mean the application requesting "raw" information as for $\underline{ECG}$ in figure 4.1. However, the whole system would gain if this knowledge could be materialized in a module of the diagram. Nonetheless, the framework would always be advantageous when more than one module is available for the "raw" request, as the middleware could deal with the request on behalf of the application (e.g. $\overline{Accell}$ with $\underline{Accell1}$ and $\underline{Accell2}$).

4.2.2 Optimization algorithm

The previous section defined the structure that describes the possible combinations to meet the requests from applications. As said, modules to be used need to have all their inputs available. Information nodes can be produced by any module capable of producing the information. Furthermore, information nodes can use more than one module for production in order to improve the metrics and meet the requirements. For example, $\underline{Accel1}$ and $\underline{Accel2}$ could be used for lower error or greater frequency. Any combination of one to all the modules is possible.

In table 4.1 we provide examples of metrics to be satisfied by the middleware. We note that requirements (metrics) have inter-dependencies. Some of the metrics are merely cumulative (latency), but others may have more complex influence on the nodes that are using the lower nodes (e.g. error propagation between input and output depends on the function utilized).

Given this, we decided to define an algorithm to search all possibilities, discarding, as we search, the ones that do not meet the requirements. This is very close to a "brute force" search, where we cut branches that do not satisfy requirements and use a cache for re-visited information nodes.

A forward note to the algorithm: it does not use directly the diagram from figure 4.1. The algorithm progresses as if building the diagram, but as we will see, for each type of information it needs it finds (from a registry information that we discuss in the next chapter) what modules are available to produce it. Modules know about their required inputs, which they state as the algorithm runs. This *feeds* the algorithm and makes it follow the diagram without building it. As the diagram is not built, information nodes do not exist in the infrastructure, only modules that correlate the information. Modules produce information that is delivered to the modules that need it.

Algorithm 1 – OPTIMIZEPRODUCTION(`infoRequestSet`)

```
1: possibilities ← CHECKPOSSIBILITIES(infoRequestSet)
2: if ISEMPTY(possibilities) then
3:     return NOTPOSSIBLE
4: else
5:     bestPossibility ← MINCOST(possibilities)
6:     return bestPossibility
```

The algorithm's main goal is to discover all the possibilities for producing all the information requested while satisfying the requirements attached to each request. This is represented in OPTIMIZEPRODUCTION(`...`) (algorithm 1), where `informationRequestSet` is a set with the requests. Each request holds the information requested and its requirements. After all the possibilities are calculated, the algorithm chooses the least costly.

Algorithm 2 – CHECKPOSSIBILITIES(`informationRequestSet`)

```
possListOfSets ← []
unmetRequestsSet ← {}
for each infoRequest in informationRequestSet
    infoPossSet ← GETINFOPOSS(infoRequest.information,infoRequest.requirements)
    if ISEMPTY(infoPossSet) then                          ▷ unable to produce Info
        ADDTO(unmetRequestsSet, info)
    else ADDTO(possListOfSets,infoPossSet)
ERRORHANDLING(unmetRequestsSet)
rModule ← CREATEROOTMODULE()
possResultSet ← CHECKPOSSIBLESOURCES(possListOfSets, rModule, 0)
return possResultSet
```

For assessing the possibilities we recursively follow the diagram starting from the point of view of M_0. The call to (CHECKPOSSIBILITIES(`...`) (algorithm 2)) is with a list of requests for M_0. This part of the algorithm checks for the possibilities of building the information request within the requirements set. If no possibility is found for some of the requests, we save the request in an unmet requests set. This unmet set is handled after all information is checked for production possibilities. This handling is not part of the optimization; a simple approach for handling could be to report an error back to the application. As mentioned, for some information there might be several possibilities of producing it within the requirements. When all the possibilities for all the information requests are discovered (in `possListOfSets`) they are combined in CHECKPOSSIBLESOURCES(`...`). We will see further down that CHECKPOSSIBLESOURCES(`...`) (algorithm 5) checks for requirements being met. However, for this part of the algorithm

(CHECKPOSSIBILITIES(...) (algorithm 2)) requirements have already been checked in GETINFOPOSS(...). The node $\underline{M_0}$ is a virtual node to combine all the requests. These two facts imply that requirements are set to zero on the call CHECKPOSSIBLESOURCES(...).

Algorithm 3 – GETINFOPOSS(info, requirements)

```
possibilitiesCached ← GETFROMTABLEPOSSIBILITIESFOR(info)
if not ISEMPTY(possibilitiesCached) then
    return possibilitiesCached
possModulesSet ← DISCOVERMODULESPRODUCERSOF(info)
if ISEMPTY(possModulesSet) then
    return {}
possModListOfSets ← []
for each possModule in possModulesSet
    possOfModSet ← GETMODULEPOSS(possModule, requirements)
    if not ISEMPTY(possOfModSet) then
        ADDTO(possModListOfSets, possOfModSet)
if ISEMPTY(possModListOfSets) then
    return {}
possResultSet ← CHECKPOSSIBLESOURCES(possModListOfSets, info, requirements)
return possResultSet
```

As CHECKPOSSIBILITIES(...) calls for the information nodes production possibilities, we move down the DAG. On information nodes (GETINFOPOSS(...) (algorithm 3)), we search for modules capable of producing the information. All the possibilities (in possInfoListOfSets) are combined and checked for requirements.

In the process of ascertaining possibilities, every module producer of this information is queried for its possibilities of producing it (GETMODULEPOSS(...)). Some modules might not be able to produce the information due to unmet requirements or lack of inputs.

There is a check for cached results so as not to "re-visit" information nodes. As an example, in figure 4.1 there are two ways to get to $\overline{CO}$. This check, also precludes revisiting the same modules. If we know about an information node, we visited all the module nodes that produce it. Note that the requirements that we now need may be different from the ones that were used to assess when the possibilities were cached. A check should be made before returning the cached result. If requests are now stricter we can just re-evaluate the cached possibilities to filter the result. However, if the requirements are looser we need to traverse the node's descendants again, as more possibilities might exist.

When modules are called to assess their production capability for the specified requirements (GETMODULEPOSS(...) (algorithm 4)), they need to check for the production of their required inputs. They combine the possibilities of getting them assessing the cost and influence on requirements. Recall that modules needs all their inputs, so a single absence means a failure.

The "leaf" modules, do not require any input and the only limit is the node's metrics against the requirements. If no possibility meets the requirements an empty set is returned.

In CHECKPOSSIBLESOURCES(...) (algorithm 5) every set of possibilities of each node's descendants is combined with all others. Combinations depend on the node type. For modules, as every input is needed, we have a Cartesian product of the possibilities. For information nodes we can have any combination of the modules producing the information, ranging from a single

Algorithm 4 – GETMODULEPOSS(`module, requirements`)

```
neededInfoSet ← module.neededInfo
if ISEMPTY(neededInfoSet) then                                   ▷ leaf module node
    if not SATISFIES(module.metric, requirements) then
        return {}
    possId ← GENPOSSIBILITYID(module)
    possibility ← (possId, module, module.metric, module.cost, {module})
    return {possibility}
possInfoListOfSets ← []
for each neededInfo in neededInfoSet
    possOfInfoSet ← GETINFOPOSS(neededInfo)
    if ISEMPTY(possOfInfoSet) then                               ▷ not possible to get the info
        return {}                                                ▷ exit, no point in continuing
    ADDTO(possInfoListOfSets,possOfInfoSet)
possResultSet ← CHECKPOSSIBLESOURCES(possInfoListOfSets, module, requirements)
return possResultSet                                             ▷ could be empty
```

Algorithm 5 – CHECKPOSSIBLESOURCES(`possListOfSets, node, requirements`)

```
possComboSetOfSets ← COMBINEPOSSIBILITIES(possListOfSets, typeOf(node))
possForNodeSet ← {}
for each possComboSet in possComboSetOfSets
    (newPossibilityId, newMetric, newCost, nodesOfPossibleTree) ←
AGGREGATENODES(possComboSet, node)
    if SATISFIES(newMetric, requirements) then
        possibility ← (newPossibilityId, node, newMetric, newCost,
nodesOfPossibleTree)
        ADDTOTABLE(possibility)
        ADDTO(possForNodeSet, possibility)
return possForNodeSet
```

one to all of them, i.e. $\bigcup_{i=1}^{\#descendants} \binom{descendantsOf(infoNode)}{i}$. Each of these combinations is then tested for requirements.

Algorithm 6 – AGGREGATENODES(`possibleComboSet, node`)

```
possibilityIdSet ← GETPOSSIBILITYIDSFROM(possibleComboSet)
nodesOfPossibleTree ← GETNODESFROM(possibleComboSet)
newPossibilityId ← GENPOSSIBILITYID(possibilityIdSet, node.id)
metricsSet ← GETMETRICSFROM(possibleComboSet)
newMetric ← AGGREGATEMETRIC(metricsSet, node.metric, node)
                                  ▷ a module needs to know how each metric applies to each info
costSet ← GETCOSTFROM(possibleComboSet)
newCost ← AGGREGATECOST(costSet, node.cost, node)
ADDTO(nodesOfPossibleTree,node)                  ▷ adds the node to the set, removing repetition
return (newPossibilityId, neMetric, newCost, nodesOfPossibleTree)
```

In AGGREGATENODES(...) (algorithm 6) metrics and cost from the combination are combined for the new possibility. All the relevant data is retrieved from the combination and aggregated for the new possibility. The calculation for metrics and cost depends on the node type (see §4.2.3). A module needs to know how each metric applies to each information, as it may have a different impact on the calculation (e.g. for error). The node set (`nodesOfPossibleTree`) contains all the nodes that are part of the solution, up to the current point in the search. The possibilities that

meet the requirements are added to the cache table in CHECKPOSSIBLESOURCES(...).

In §3.6, we discussed Service Discovery (SD) and partially matching requirements from applications with the capabilities of current nodes. Algorithms 5 and 6 are the basis for this matching. In algorithm 5 the possibilities for producing a specific information are combined, by arranging all hypotheses. In algorithm 6 the relevant part occurs, as capabilities of the same type are combined to produce a single assessment of said capability. This result is then checked for compliance with the requirements, in algorithm 5. Combining the capabilities is the real issue; we will provide some insight for this in the next sub-section.

4.2.3 Metrics

The requirements that an application may use for its requests can be varied. In table 4.1 we have examples for some of the more likely ones. *Frequency* is the period at which a module can produce its output. In the case of modules representing sensors this may be limited by the rate of collection and communication. *Latency* represents the time that it takes for a module to produce its output after receiving all of its input. In a sensor it is the delay between sampling and output availability, including processing time if any [16, Annex C]. *Energy*, *CPU* and *IO* usage account for the respective spending/usage at the modules. *Error* represents the error associated with every node. In sensor modules this could be the error inherent to the sensor (measuring, integration, digitalization, etc.). On other nodes of the diagram it represents the error introduced in calculation (for modules) or while aggregating sources (information nodes).

Table 4.1 – Metrics examples

Metric	Func in Info (I_α)	Func in Module (m_i)
frequency	$\sum_{i\in\check{D}_\alpha} Met_i^{\text{freq}}$ ◊	$\min\big(\{\alpha\in\check{D}_i : Met_\alpha^{\text{freq}}\}\cup\{C_i^{\text{freq}}\}\big)$
latency	$\min\{i\in\check{D}_\alpha : Met_i^{\text{lat}}\}$	$\max\{\alpha\in\check{D}_i : Met_\alpha^{\text{lat}}\}+C_i^{\text{lat}}$
energy, CPU, IO	$\sum_{i\in\check{D}_\alpha} Met_i^{\text{energy}}$	$\sum_{\alpha\in\check{D}_i} Met_\alpha^{\text{energy}}+C_i^{\text{energy}}$
error	$f_\alpha^{\text{error}}(\check{D}_\alpha\cup\{I_\alpha\})$	$f_i^{\text{error}}(\check{D}_i\cup\{m_i\})$

◊ this would imply some form of synchronization and phase shift in the timings for the leaf modules (thus in the sensors) (e.g. the work of Elson et al. [29]);
– Met_ξ^μ is the calculated metric μ on node ξ (an information or module);
– C_i^μ is the capability μ of the module node i, this is a characteristic of the module and independent of its descendants. Information nodes (being virtual) do no have capabilities;
– $\check{D}_\xi$ = { direct descendant nodes of node ξ part of the possibilities};
– $f_\xi^{error}()$ is the error function which calculates the output error based on the inputs used and their errors, for node ξ.

The table shows that the calculation is different for the two node types. Not all metrics from table 4.1 can accommodate duplicate accounting as per the formulae given. As an example, $\overline{CO}$ is needed for $\underline{O_2\ Del\ Formula}$. $\underline{M_0}$ needs $\overline{CO}$ and $\overline{O_2\ Del}$. When calculating cost and requirements the production of $\overline{CO}$ should not be accounted for twice.

In cumulative metrics (like energy) this would cause the error of counting twice the metric from a node, when it is used only once. As such, for energy, CPU and IO it would be better to calculate the metric at the top information nodes, the ones representing the applications' requests,

with $\sum_{i \in \check{S}_p} C_i^{\text{energy}}$, where $\check{S}_p$ is the set of modules used for the given solution p. For the other metrics, the use of $\min$ or $\max$ removes the duplication. For the error, the defined error function (f^{error}) should take this into account when evaluating. Note that in each of these cases the $\check{S}_p$ would represent the set of modules that satisfy the requirements for the request, at each of the levels of the diagram.

Table 4.2 – Cost examples

Metric	Duplicates Allowed	Cost Func
frequency‡	NA	$\min\big(\{\alpha \in \check{D}_{M_0} : Met_\alpha^{\text{freq}}\}$
latency‡	NA	$\sum_{i \in \check{D}_{M_0}} Met_i^{\text{lat}}$
energy, cpu, io	No	$\sum_{i \in \check{M}} Met_i^{\text{energy}}$ ◊
error/accuracy	Yes	$f_0^{\text{error}}\big(\check{D}_0 \cup \{M_0\}\big)$

‡ sum, min or max functions could be applied; here we have min of frequencies for producing information requested and sum of all the delays to get the information requested;

◊ $\check{M}$ represents the modules that are the solution for the current requests from the applications. There can be more than one $\check{M}$, depending on the possibilities available for M_0.

– $\check{D}_{M_0}$ = { information nodes descendants from M_0};

In terms of cost, the same metrics may apply (see table 4.2). The cost is seen from the M_0 node as a global parameter to optimize across current requests. For frequency and latency several functions can be used for calculating cost, minimum of frequency or latency from all the information nodes descendant of $\underline{M_0}$, or maximum or the sum of. For energy, CPU and IO the global formula from the previous paragraph should be applied, i.e. $\sum_{i \in \check{M}} C_i^{\text{energy}}$, where $\check{M}$ is the $\check{S}_p$ from above but from the point of view of M_0. It would be the result of calling CHECKPOSSIBLESOURCES(`possListOfSets, rModule, 0`), where the possibilities would be combined without node repetitions. There will be several $\check{M}$s if there are several possibilities to fulfil the applications requests. The least costly is the one to be chosen. Error could also be seen as a global optimization criterion, where after calculating the error for each request one would use $f_0^{\text{error}}(\check{D}_0)$.

4.2.4 Complexity analysis

For analysing the proposed algorithm we used a worst-case analysis as discussed in the CLRS book [21, chap. 2.2]. Although an average-case [21, chap. 2.2] or the Spielman and Teng's smoothed analysis [107] would provide more accurate and realistic (and lower complexity) approximations, it is currently difficult, not possible even, to assess the probability of having more or fewer modules and/or which ones.

The average or smoothed analyses for our algorithm imply having some knowledge (probabilistic, statistical, etc.) on the number of modules available and the requests from the applications. We would need to be able to answer questions like: how many modules are present, how many are repeated, how many requests are made, how many can be satisfied given the modules in the

system, etc. Remember that these modules are not only the sensors, but also the modules for correlating data.

As we do not have information to make a realistic assessment, we opted for doing a worst case analysis that is more general although leading, in our case, to bad complexity results.

In this setting, the worst case scenario is when all the current modules in the model are requested to produce. This is the limiting factor. Even if requests from applications increase, if there is no way to produce the information, the impact is small, as we cut off non valid branches. Taking the number of modules (M) in the system as the input, without considering how they correlate, we have that the limiting part of our algorithm is CHECKPOSSIBLESOURCES(...) (algorithm 5).

As there is a dependency from the first parts of the algorithms on the latter, we start the analysis from the last to the first.

Table operations

A table is used in GETINFOPOSS(...) (algorithm 3) as a cache for already visited nodes. We use a table with indexes. The indexing uses re-hashing and open addressing, where the number of elements in the hash is always smaller than the possible set of keys (the cardinality of), with α as the table load factor [21, chapt. 11]. There are no deletions on the table (apart from a complete clear). On the worst case, an insertion causes a re-hashing, meaning an insertion of the current entries plus the new one (total of M entries). For the worst case every one of these insertions causes a collision and thus we have $\sum_{n=1}^{M} n$ insertions, i.e. $M\frac{M+1}{2}$. As such in the worst case, an insertion is $O(M^2)$. Note however that the average case is $O(\frac{1}{1-\alpha})$ with double hashing [21, p. 274]. Looking up an entry is $O(M)$ for the worst case as every lookup implies a collision. Again we have that the average case is $O(\frac{1}{1-\alpha})$.

$$\text{ADDTOTABLE}(\ldots) = O(M^2) \tag{4.1}$$

$$\text{GETFROMTABLE}(\ldots) = O(M) \tag{4.2}$$

aggregateNodes

AGGREGATENODES(...) (algorithm 6) is $O(M)$ as at most it involves aggregating all modules.

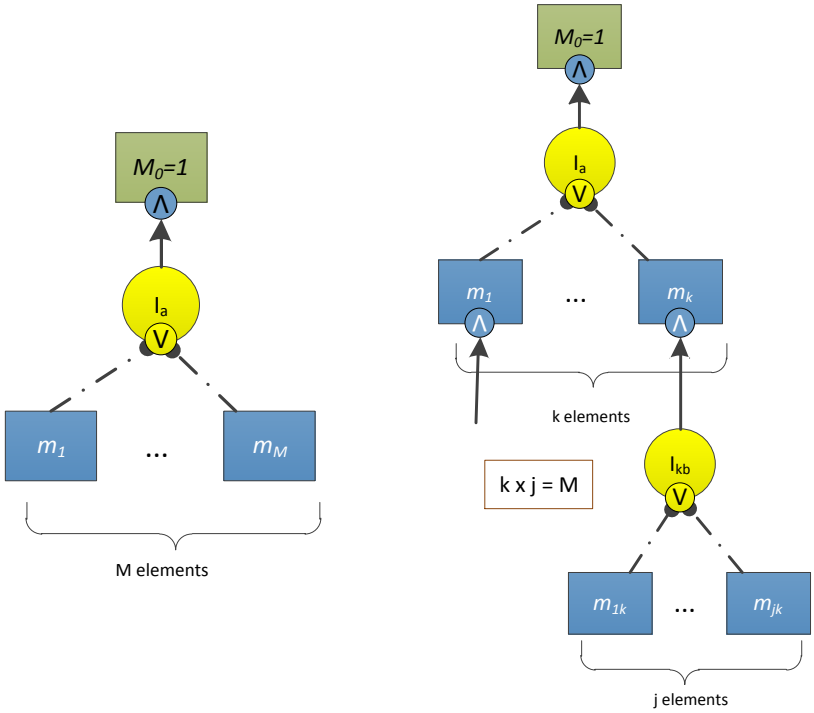
$$\text{AGGREGATENODES}(\ldots) = O(M) \tag{4.3}$$

combinePossibilities

COMBINEPOSSIBILITIES(...) is used in CHECKPOSSIBLESOURCES(...) (algorithm 5). For analysing it, we define Worst Case Number of Descendants (WCND) and Worst Case Possibilities (WCP) for a node. In the case of a node module, as it needs all of its inputs, the worst case for a combination is WCP^{WCND}. An information node can have any combination of the modules. We have in that case $\sum_{k=1}^{WNCD}\left(\binom{WCND}{k}\cdot WCP^k\right)$. The binomial series has $\sum_{k=0}^{\infty}\left(\binom{\alpha}{k}x^k\right) = (1+x)^\alpha$, so we can say

$$\sum_{k=1}^{WCND}\left(\binom{WCND}{k}\cdot WCP^k\right) < (1+WCP)^{WCND} \tag{4.4}$$

The worst case for number of descendants of an information node is having all M modules producing the information, as illustrated in figure 4.2.a. In this case the WCP is 1 as each module just provides one possibility. From equation 4.4 we get 2^M.



4.2.a: Worst case number of sons

4.2.b: Worst case possibilities

Figure 4.2 – Worst case scenarios

For the worst case number of possibilities we take as example a two level diagram, illustrated in figure 4.2.b. M_0 has only an information node ($\overline{I_a}$) that in turn has k modules for producing it. Each of these modules needs one input ($\overline{I_{kb}}$ for module k) that is produced by j modules. We have that $k \cdot j = M$. The modules for producing $\overline{I_{kb}}$ are leaves and we can combine them for production in $((1+1)^j - 1)$ ways[3]. At $\overline{I_a}$ we have $((1+(2^j-1))^k - 1)$ ways. This leads to a complexity of $O(2^M)$.

If we had more levels equally distributed it would lead to the same $O(2^M)$ as the product of the number of elements in each level would need to be M. We analyse the case of two levels with k elements in the first level. This level has $k-1$ leaves and the other module using one information node with j module nodes available to produce it. This would lead to $O(2^M)$ as we would converge to $2^{k-1} \times 2^j$ which results in 2^M as $k + j = M$ in this case. Thus:

$$\text{COMBINEPOSSIBILITIES}(\ldots) = O(2^M) \tag{4.5}$$

checkPossibleSources

CHECKPOSSIBLESOURCES$(\ldots)$ (algorithm 5) has its worst case running time as:

$$\text{COMBINEPOSSIBILITIES}(\ldots) \times \left(\text{ADDTOTABLE}(\ldots) + \text{AGGREGATENODES}(\ldots)\right)$$

and from the results above (equations 4.1, 4.3 and 4.5) we have

$$\begin{aligned} \text{CHECKPOSSIBLESOURCES}(\ldots) &= O\left(2^M\right) \times \left(O(M^2) + O(M)\right) \\ &= O\left(M^2 \cdot 2^M\right) \end{aligned} \tag{4.6}$$

[3]The binomial starts at $k = 0$, whereas we combine from $k = 1$.

getInfoPoss

GETINFOPOSS(...) (algorithm 3) has a table lookup, discovering modules and then for the modules discovered extracting their possibilities for production. At the end, it calls CHECKPOSSIBLESOURCES(...). Discovering modules is at most going through all M modules. Getting all modules for production leads to $O(M)$ and for the possible combination of all $O(2^M)$, as we saw on combining possibilities. And from equations 4.2 and 4.6, we have:

$$\begin{aligned} \text{GETINFOPOSS}(\ldots) &= O(M) + O(M) + O(M) \times \left(2^M\right) + O\left(M^2 \cdot 2^M\right) \\ &= O\left(M^2 \cdot 2^M\right) \end{aligned} \tag{4.7}$$

getModulePoss

GETMODULEPOSS(...) (algorithm 4) adds to the table and then iterates through the information needed getting all possibilities. At the end it calls CHECKPOSSIBLESOURCES(...). The information needed is at most of size M. If it was more then some information was not being produced and the cycle would terminate earlier. However, for this case GETINFOPOSS(...) would only provide one result for each call. As we discussed for COMBINEPOSSIBILITIES(...), if we needed only one information, produced by every other module we would have $O(2^M)$. Note that the other operations in GETMODULEPOSS(...) are O(1), not dependent on M. Thus from equation 4.1 and equation 4.6:

$$\begin{aligned} \text{GETMODULEPOSS}(\ldots) &= O\left(M^2\right) + O(1) \times \left(2^M\right) + O\left(M^2 \cdot 2^M\right) \\ &= O\left(M^2 \cdot 2^M\right) \end{aligned} \tag{4.8}$$

checkPossibilities

CHECKPOSSIBILITIES(...) (algorithm 2) iterates through all the requests testing for the possibilities of answering each one in each cycle. Then it calls CHECKPOSSIBLESOURCES(...). The size of the set of requests is unrelated to M as we use a cache for getting the information. The only problem would be requesting information that can not be produced by a lack of inputs on the leaf nodes. If so, the cache would mean that we would quickly discover the impossibility of production. The possibilities as discussed above are limited by the modules and thus are $O(2^M)$. From equation 4.6 we have:

$$\begin{aligned} \text{CHECKPOSSIBILITIES}(\ldots) &= O(1) \times O\left(2^M\right) + O\left(M^2 \cdot 2^M\right) \\ &= O\left(M^2 \cdot 2^M\right) \end{aligned} \tag{4.9}$$

optimizeProduction

The main loop of the algorithm OPTIMIZEPRODUCTION(...) (algorithm 1) calls CHECKPOSSIBILITIES(...) and calculates a minimum from a set. The set has WCP and as such for a non-ordered set it leads to $O(2^M)$ for sorting. From equation 4.9:

$$\begin{aligned} \text{OPTIMIZEPRODUCTION}(\ldots) &= O\left(M^2 \cdot 2^M\right) + O\left(2^M\right) \\ &= O\left(M^2 \cdot 2^M\right) \end{aligned} \tag{4.10}$$

Observations

It is easily seen that this is a very bad worst case scenario, however it is very unlikely (if at all possible) to happen. Nonetheless, it gives an understanding of how the algorithm works.

Worst case space analysis leads us to a similar pessimistic result of $O(M \cdot 2^M)$. In this case the cache table is the *hogging* factor where we have 2^M possibilities (thus table entries) each having M modules as data (which in a real scenario would not happen for every entry).

Note that the algorithm discards possibilities that do not meet the requirements. However, in the worst case analysis all possibilities meet the requirements.

As an anecdotal example of performance from figure 4.1, which has 22 modules, we have 282 operation calls performed to find 135 possibilities (worst case with $M = 22$ is $22^2 \times 2^{22} = 2030043136$). This took an average of 0.4 ms in a laptop with an Intel 2.66 GHz CPU and 4 GB of memory, using an implementation in Java, where the possibilities table was implemented as a `HashMap` with chaining[4].

4.2.5 Model notes

The middleware uses a pub/sub architecture for distributing the information, which we describe in the next chapter. Modules subscribe to their inputs when they are called for production. At startup they register their production so that this can be discovered by the middleware. Required inputs are stated when searching for a configuration. The flow of information between modules imposed by the pub/sub framework follows the diagram that arose from the modules selected for the solution found.

The diagram does not have cycles because we mandate that modules do not use their outputs as inputs or any other information derived from their output. This also prevents feedback loops in the pub/sub system.

The search can be run as soon as a new request comes to the middleware. If the system was running a configuration for previous requests, we could search for the new request and then re-do the last step of the algorithm, i.e. combine at the M_0 node with the new request. This would imply that the previous requests' solutions be cached.

4.3 – Conclusion

In the next chapter we provide related work for the subjects addressed here and in chapter 5.

In this chapter we defined a model for defining correlations of different inputs in the middleware. The objective is supporting a variety of applications and easing their access to the data they need, be it a raw sensor value or a derived value. This objective must be achieved while optimizing resource usage. This correlation and calculation is done through software modules. Some examples of formula-based modules that calculate new information based on known mathematical correlations of physiological data were given. The distribution of this information through the model is based in a pub/sub system that will be described on the next chapter.

[4]Confidence interval of 95% leads to a top value of 3.8ms.

We also described an algorithm for optimizing resource usage according to applications' requests. The algorithm was analysed regarding its complexity, in a worst case scenario. As we saw, this gave a bad result that however will not occur in practice as the worst case studied is very unlikely to occur.

This flexibility of optimization algorithms/objectives gives broader options to applications and developers. It would be possible to define a request that would drain resources in a few hours for getting the most reliable and up to date information. An example from the opposite direction, would be to have lower quality information being sent for years.

4.3.1 Open issues

Regarding conflicts, we described that intersection of requests is a possibility in the system, where the strictest requirements are used so as to fulfill the strictest request. There might exist some cases where it is not clear what the strictest requirement is. With a similar example as from §4.1.1, if for example application $\mathcal{A}$ requested a frequency of 4 Hz and another $\mathcal{B}$ 5 Hz, there could be several solutions. A 20 Hz frequency could be one option, where both applications would receive more than asked for. Another option, 5 Hz frequency, would satisfy $\mathcal{B}$, but would require some adaptation from $\mathcal{A}$. The module could also sample at 4 Hz and 5 Hz. The first solution could not be possible if the module/sensor was not capable of that frequency, which would also undermine the third for the same reason. The current implementation of the system notifies applications that a request is not able to be met with an identification and description of the reason why. However, it does not allow us to send extra data regarding which requirements were not met. It would be advantageous if applications were informed of such discrepancies as above, so as to be able to advise the middleware of their flexibility in the requirements. In the given example, application $\mathcal{A}$ could decide to change its required frequency to 5 Hz after being informed by the middleware of the conflict.

This last issue, also raises the problem of synchronization between Body Area Network (BAN) nodes and the BS. As we mentioned in §3.9.4, the sensor/actuator nodes should be clock synchronized with the BS, for frequency controls to work out.

A point we did not address but may lead to interesting research is the development of modules that incorporate fuzzy logic and/or qualitative reasoning, as described by Forbus [32]. The latter enables inferring state and state transitions using only partial knowledge, and non-numerical descriptions of systems and their behaviour. It would allow us to define the expected evolution of a system based on common knowledge. Fuzzy logic also enables actions to be based on incomplete knowledge, without having total confidence in knowledge of the current state of the system. The approach is however different in that it uses statistical models or artificial intelligence models such as neural networks or clustering means. Sugeno and Yasukawa [110] propose the combination of both fields to provide fuzzy logic based qualitative models.

5

Information flow

After describing the model for correlation and aggregation of data, in this chapter we focus on the distribution of the information and its flow between the different modules. This flow uses a Publish/Subscribe (pub/sub) paradigm that is based on the model from chapter 4.

5.1 – Pub/Sub system

Recalling some of the objectives set in chapter 3, we have: **(B)** convert raw sensor data into information in the human model and **(C)** correlate, according to the models, information and its metadata. With the modelling approach described and the pub/sub system, we aim to address these goals.

The previous chapter led to a better understanding of where the commonalities between different applications can be devised. As we discussed in §4.1, (see also figure 4.1) the *Physical Activity application* and the *Monitoring application* are both requesting acceleration. Indirectly, both may need Cardiac Output (CO). The figure also illustrated how the same information can be built using different inputs, e.g. Metabolic Equivalents (METs) calculation. The requirements for acquiring and correlating that information along with the costs of doing so were also mentioned. These factors are incorporated in a pub/sub architecture [30] to disseminate the information through the different modules.

In a pub/sub system, publishers send events to brokers that in turn forward the events to subscribers. That, of course, implies that subscribers need to "subscribe" to events. In these systems, when subscribing, there are usually some selection criteria available. There are possibilities to filter on the type of event (e.g. alarm, new price), on the topic of the event

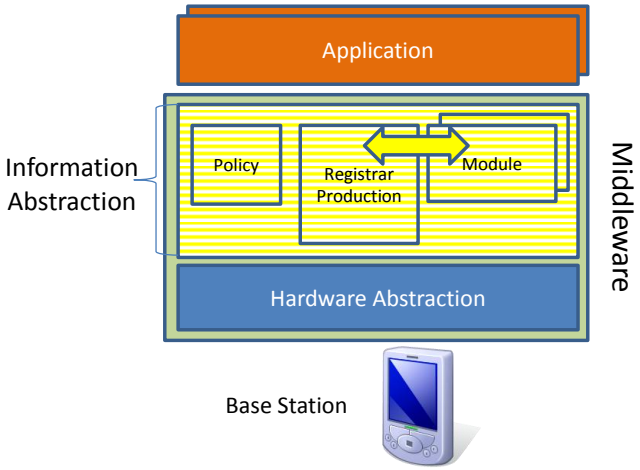


Figure 5.1 – Pub/sub architecture

(e.g. sport news, medical information) or on the content of the event (e.g. events that mention Body Area Network (BAN)).

In our system, modules (the Software (SW) components that correlate information) subscribe to a type of event and publish to the system the information they produce. This is portrayed in figure 5.1. A specific component acts as the broker (`RegistrarProduction`) and a `Policy` component is responsible for finding the best module configuration (i.e. the set of modules that will produce the information requested while satisfying requirements) for the current application requests, using the models provided. The hardware abstraction layer enables requesting and receiving the "raw" sensor data.

As such our pub/sub system has:

- software modules that correlate information by publishing their outputs and subscribing to their required inputs;
- a framework to allow optimization of resource usage, taking into account modules' "costs";
- different types of modules providing different functionalities.

In the next sub-sections we describe the different modules and brokerage.

5.1.1 Modules

Modules are the components that process information. They embed the principles for correlating inputs to produce a specific output. They may use several inputs, but only produce one output. For example, module $O_2DelFormula$ in figure 4.1 uses three inputs (CO, SpO_2 and Hb) to produce one output (O_2Del). Module production is published in the central `RegistrarProduction` that sends it to the modules that subscribed to that information.

This structure enables the composition of results by "chaining" modules together (connecting their outputs with inputs) to produce complex calculations/dependencies between data. As expected, outputs can be sent to different modules (multicast), enabling different interpretations of the same produced value in different contexts. Associated with the data produced, there are metadata fields that describe the accuracy and the time of production. These can be used by modules or applications for more detail about the data.

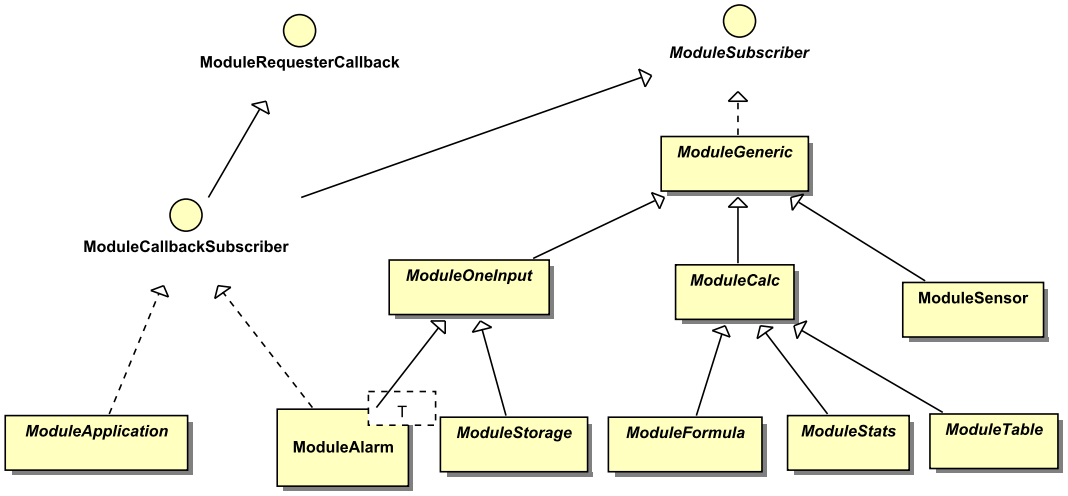


Figure 5.2 – Module types

As we discussed, a restriction imposed is that the dependency graphs (e.g. figure 4.1) do not have any loops. This is so as not to cause feedback loops in the pub/sub system and to facilitate the optimization process. As such, the graphs are Directed Acyclic Graphs (DAGs), where the direction is given by the information flow.

A module has some capabilities and a cost that describes how it is capable of producing a specific output. Modules can (and should) be implemented by the application developer if a specific correlation or functionality is not present. There are several types of modules already developed. These are abstracted in an inheritance tree as shown in figure 5.2.

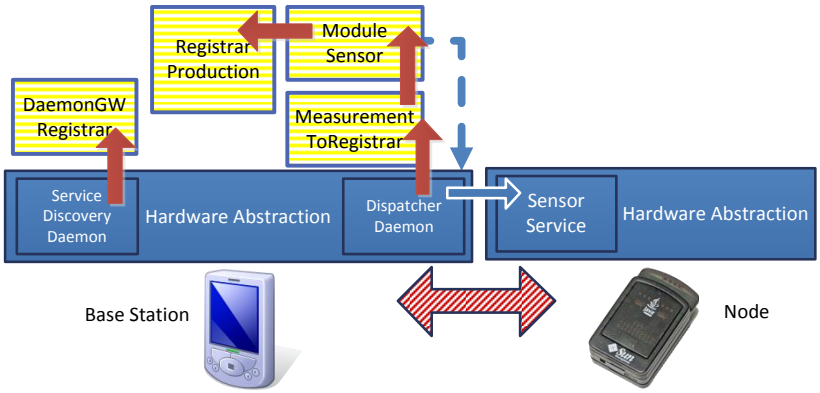


Figure 5.3 – Module sensor data flow

Some notes:

- A module is a “dumb” component, in that it does not take any action regarding optimization. Modules always assume that some outside component coordinates the different modules for the production of the cascading outputs. As such they only subscribe to inputs needed, assuming that their frequency, accuracy, etc. have been set by this other component (in §5.2.2 we provide more detail);
- A module knows about the requests made to it that are currently active. This is used for *un-subscriptions* sent to it (it changes to the most stringent requirement that it currently holds).

The first module we discuss, is the one that represents the sensors from the BAN. Figure 5.3 shows the interaction of the different middleware layers for discovering a new node and receiving data from nodes. When a new node appears in the system it advertises itself using the Service Discovery (SD) function, as described in §3.6.1. This is received by the SD daemon on the Base Station (BS) which notifies a `DaemonGWRegistrar`. This component then creates as many sensor modules as the sensors the node advertised. The information regarding the node and the sensor characteristics is kept in the structures pictured in figure 5.4. The location of a node is defined as a point with set coordinates in the body. A sensor is positioned according to this coordinate system[1]. A `DataValue`, or a measurement, is related to a sensor as we mentioned in chapter 3.

A `ModuleSensor` maps the requests made to it to commands for the sensor. When the sensor produces data it flows to the BS using the specific communication system. The `DispatcherDaemon` component is responsible for de-multiplexing the messages received by the network, in this case a message with a measurement. This is sent to the `MeasurementToRegistrar` component that directs it to the module that is responsible for this sensor. The module then publishes the data on the `RegistrarProduction`.

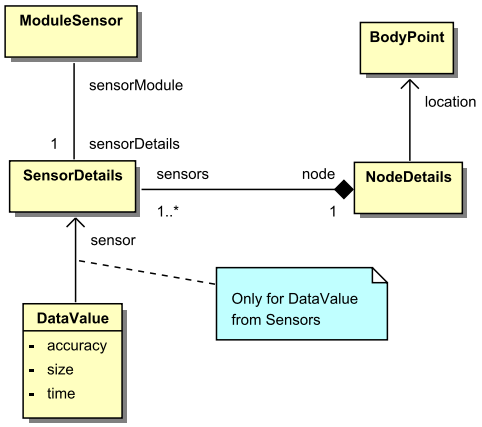


Figure 5.4 – Module sensor

The `ModuleCalc` abstracts a type of module that does calculations. It can be a formula-based module (`ModuleFormula`) where formula-based calculations are done (e.g. $O_2DelFormula$); a statistics-gathering module (`ModuleStats`) or a table-based output module where the output results from a table lookup (`ModuleTable`).

`ModuleApplication` is the counterpart of the application. This is used by the application to interact with the architecture, where it subscribes to the information requested.

The `ModuleStorage` is a module that enables storage of values produced outside the module and access to that data.

`ModuleAlarm` checks the value of the information subscribed to and sees if it is within some specified limits. It is a simple check of being within a range. If an average or other statistic over time is needed, a combination with `ModuleCalc` could be used (discussed further in §5.2.4).

The interfaces represented in figure 5.2: `ModuleSubscriber`, `ModuleRequesterCallback` and `ModuleCallbackSubscriber` define respectively the interface for subscribers, requesters (where a callback is defined for error information) and both subscribers and requesters.

Data structures

The relevant data structures used in the information abstraction layer are:

- `DataValue`: this is the structure that carries the information produced. It entails an accuracy attribute that is updated as the modules compute the values in the modules' chain. Part

[1] Although we have a data structure for this we have not yet defined the coordinate system to use. We come back to this in §5.4.1.

of the metadata is also a timestamp that defines when the data was produced. For *raw* sensor data, the sensor details are also included together with the location of the sensor (see figure 5.4); in this case they contain the measurements from the sensors;

- `Requests`: a request contains the information needed and the requirements (delay, error, etc.) for getting it. It is used when subscribing or asking for a specific output. A `ModuleRequest` is used by the `Policy` class (see §5.2.2) when calculating the modules needed for producing a specific `DataValue`.

Modules are in essence programmable components that take inputs and produce an output. As another *module* example, they could re-order out of order measurements/data values received. In §3.5.2, we mentioned that only if a reliable connection is established messages arrive in-order. A module could, for the case where the channel is unreliable, check the timestamp of said measurements and re-order them. In most cases, what is more natural is to discard out-of-order or delayed measurements. This illustrates that a module can provide a varied set of functionalities. However, its strength is also its ability to be re-used. As such, a module should be develop to provide a single functionality.

Modules can be instantiated and managed by other modules, the registrar or an application. When modules create other modules some attention is due, as this may lead to problems if, when the owner module stops or is destroyed, it also stops/destroys its owned modules (that in the mean time were being used by others).

5.1.2 Brokerage

The `RegistrarProduction` component from figure 5.1 is the central component that controls the subscription, production and optimization. The component needs to know about producers of `DataValue`s, which thus need to register, and the subscribers to those values, which need to subscribe. This enables one of its main tasks: distributing information.

For storing these two lists of producers and subscribers, the `RegistrarProduction` uses `RegistrarInfo`. Another list stored is the pending requests list, which holds the requests that could not be fulfilled due to a producer not being available for a needed `DataValue` in the production tree. `RegistrarInfo` also holds the current known policies for using the resources.

The `Policy` component is responsible for choosing which modules are used to satisfy the applications' requests while optimizing for a *defined* cost. The `Policy` is an *abstract* class that defines the interface that implementations of a policy need to conform to. This is so as to give flexibility of defining different policies that optimize to different costs. The preferred algorithm to use is the one presented in §4.2, but it is possible to define a policy that uses a different one.

5.2 – Component interactions

This section describes some of the interactions between components that are supported by the architecture. We use Unified Modelling Language (UML) [83] sequence diagrams in some cases to better illustrate said interactions.

5.2.1 Requests

Requests are done by modules that need information to function: application modules for the application, other modules for storage, calculations, etc. There are two main ways of receiving data: pull and push.

As expected, *pull* is a one-time query that is made as needed. The requesting component may need to get a list of producers from which to request the information it needs. The pull can lead to a push based request if the value is not available; e.g. the producer module does not have the inputs needed to produce, or has not yet received them. The pull request can also have a minimum "freshness", where the requester can restrict the "age" of the information. This type of request is mainly used for getting information that is already being produced for a *push* request, as it does not entail any of the "information flow building" process.

The "push" request implies a subscription of the type of information requested and can have requirements associated. As illustrated in figure 5.5, this request leads to a chain of requests to an *optimum* calculated module tree. This tree is calculated by the `Policy` component so as to produce the requested information. The `RequestOutputSeqDiag` checks the modules to verify they are able to produce the requested values. This serves as a double check because the `Policy` component should have already enquired about this (see next section, §5.2.2). It also encompasses starting production, if not yet started, or changing the settings if this request is stricter than the currently served one. Subscription to the needed inputs is also done by the module at this point.

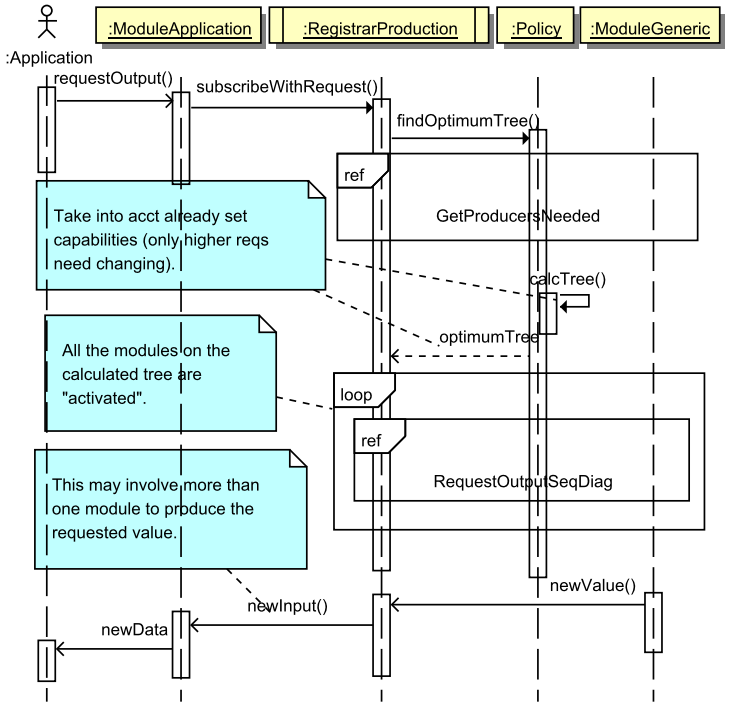


Figure 5.5 – Request push value – subscribe (using an application as example)

5.2.2 Optimization

The `Policy` component is responsible for calculating the modules and their requirements to be used to comply with the subscription request. This calculation also involves optimizing the

cost of the solution according to the internally implemented policy. This "implementation-based" policy enables the use of different strategies for optimization (battery life, higher accuracy, etc.).

The `Policy` component is thus responsible for discovering the tree of modules needed and the settings (capabilities) they need to use. Modules are oblivious to settings of other modules, they must however state their requirements for answering a specific request. When asked to produce within specific requirements, they subscribe to their required inputs and assume that the modules that produce them have been requested with (at least) the requirements the module needs. This centralization enables the `Policy` component to have a wider view and control of the whole set of requests and modules being used.

Each capability has an associated cost (see figure 5.6) that is used in the optimization problem. Capabilities are also used to describe the requirements set by the requester. As seen in figure 5.6, examples of parameters of these requirements/capabilities are accuracy (error), frequency of updates and sending rate. For cost we have processing, delay, energy and Input/Output (IO). For sensor modules some of these capabilities and costs are directly related to the underlying characteristics of the sensor's hardware (e.g. accuracy, energy, etc.).

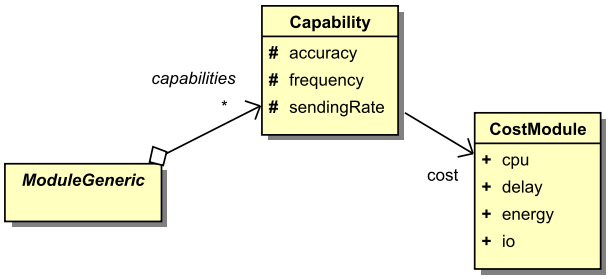


Figure 5.6 – Module's capability and associated cost

In figure 5.7 we illustrate retrieving the producers for the inputs needed and assessing the inputs they need for the request/requirements and their current details. When evaluating a new request the `Policy` takes into account the current settings for every module and the requirements set by the request. Since the new request can only increase or maintain the capabilities needed it only issues stricter or the same requirements. The result of the `Policy` optimization call is which modules to change settings (including starting their production), the ones to stop and the ones with no change (they are used in the request, but do not need to change their current settings).

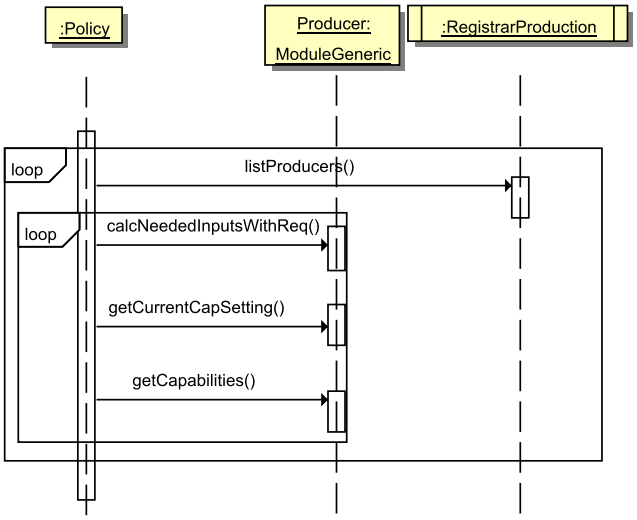


Figure 5.7 – Get producers

5.2.3 Producer un-registering

A producer may decide to stop producing due to change in sensing behaviour, detected malfunctioning[2], battery draining[3], etc. The module should inform the `RegistrarProduction` of this action. This leads to a re-assessment of the solution used. We illustrate this in figure 5.8. The modules that were currently subscribing to the information being produced by the stopping module are found and the strictest request that each was serving assessed. After this, a search for another producer module (or combinations of modules) for substitution is done. If it is not possible to find a replacement module, the requests are all re-done as if they were new requests. This re-request is done after an un-subscription to the previous requests, so as to clear the affected requests.

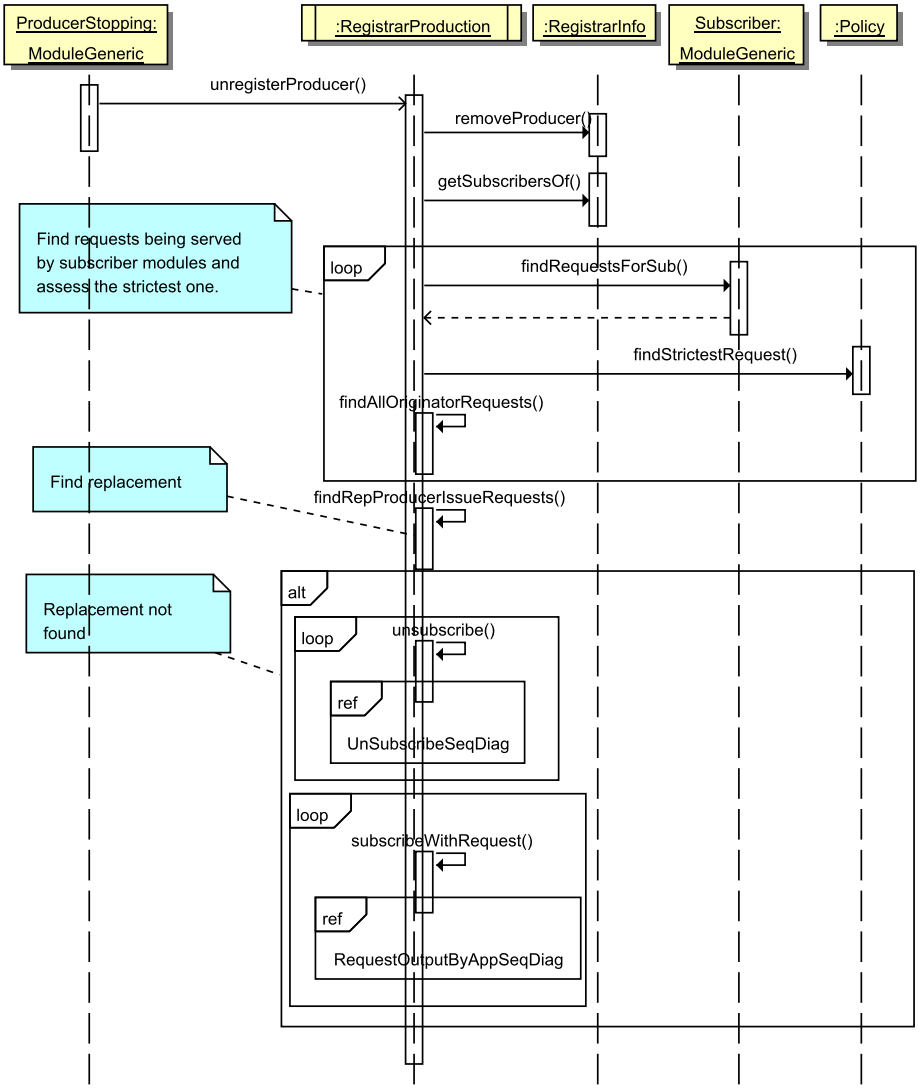


Figure 5.8 – Producer un-registering

[2]Note that the middleware can detect communication losses with nodes (described in §3.6.1), but hardware malfunctioning is not part of current functionality. As such this would be implemented internally by the Operating System (OS) on the node. The middleware provides a call to notify of this stoppage.

[3]This function could be implemented in a policy and lead to a re-assessment of the modules to be used.

5.2.4 Alarms

In figure 5.9 we show the sequence for instantiating an alarm for notification of a value going out of specific bounds (e.g. seeing if Blood Pressure (BP) is within certain limits). The sequence illustrates a module creating the `ModuleAlarm` and setting the bounds and type of value to be watched. The `ModuleAlarm` subscribes to the `DataValue` to be monitored, with the requirements needed according to the request from `AlarmedModule`. This triggers the request for production of this `DataValue` as was discussed in §5.2.1. The `AlarmedModule` then subscribes to the value being produced by the `ModuleAlarm`, this type is returned when setting the bounds. The `ModuleAlarm` receives the new inputs of the `DataValue` being watched. When the value gets out-of-bounds the `ModuleAlarm` publishes the alarm. The `RegistrarProduction` then sends it to the `AlarmedModule`. Removal of the `ModuleAlarm` is, of course, optional. This implies that `ModuleAlarm` is no longer needed and thus `ModuleAlarm` unsubscribes to `DataValue`.

Note that the `ModuleAlarm` did not register its production. As such it can not be queried by other modules. The rationale is that setting bounds is very specific, and *polluting* the `RegistrarProduction` with this production type is unnecessary[4].

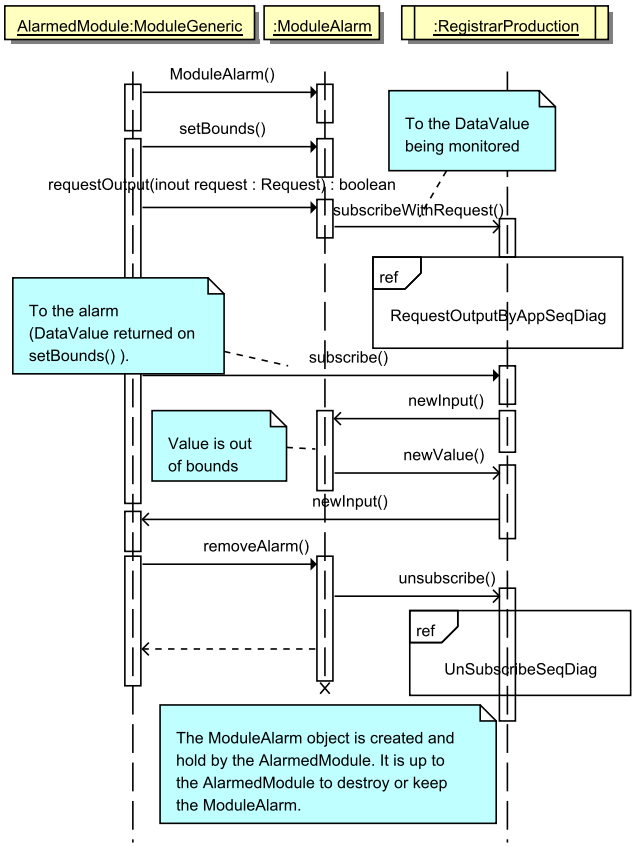


Figure 5.9 – Alarm example

5.2.5 Producer unavailable

When the `Policy` tries to find the modules' tree to fulfil a request it may be unable to find a producer of a needed `DataValue`. In this case the requester should be notified that the request

[4]An argument could be made for the `ModuleAlarm` directly calling back the `AlarmedModule` instead of publishing it. We opted for maintaining the defined flow of data per the pub/sub architecture.

was unsuccessful. The request may additionally enter a “pending-requests” list, so that when a producer capable of producing the needed information registers, the request may be fulfilled. In figure 5.10 we see the associated sequence.

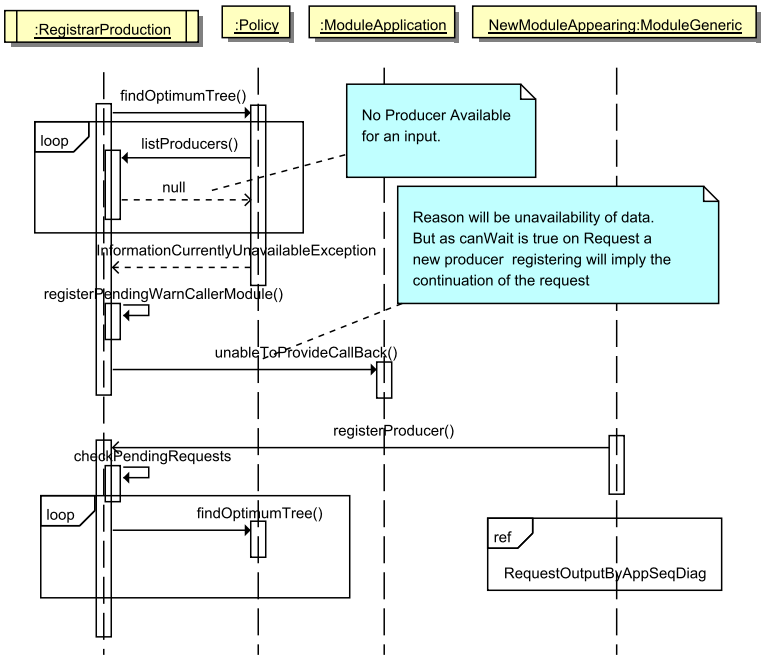


Figure 5.10 – Producer unavailable

5.2.6 New information/value

In figure 5.11 we can see the sequence of messages when a new value is produced with the chaining of modules shown as the loop of `newinput()`s. In the example sequence an application is the final destination of the correlated data. We will see the case for when a sensor produces the information in the next chapter.

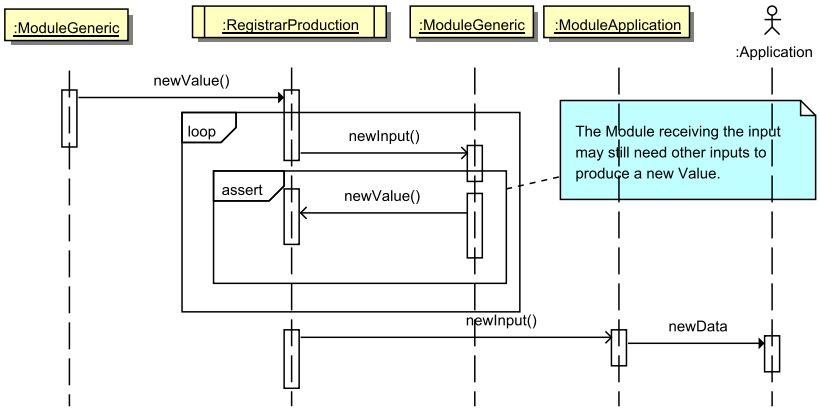


Figure 5.11 – Data new value

5.2.7 Un-subscription

When a module unsubscribes to an input (e.g. a `ModuleApp`) it may trigger several un-subscriptions from the tree of modules working for the unsubscribed information. Each producer module re-assesses its requirements according to the new strictest request, which might not

change if the un-subscription did not pertain to it. The `RegistrarProduction` centralizes the un-subscription and informs all the producers affected by this change. The modules may need to change their subscriptions; make new and/or unsubscribe to current subscriptions. Note that the `Policy` component always informs the modules of the requests they are serving even if they are not stricter[5]. As such, the modules know what they are servicing. The system is always tuned to the strictest request according to the optimization criteria used.

5.3 – Related work

As mentioned in the introduction, in 1965/6 Herbert and Weil [104] ran an experiment in a hospital shock unit where they introduced a computer (IBM 1710) to gather and assess patient real-time data. The objective was to increase the frequency of tests and their assessments; previous statistics showed poor efficiency in routine manual methods. They monitored arterial and venous blood pressure, ElectroCardioGram (ECG), body and air temperatures and urinary output. There were also derived/calculated values such as respiratory rate, Heart Rate (HR) and pulse rate. They did statistical analysis of the sensed data to derive mathematical relationships between the patient data to identify *physiological and biochemical relationships that are determinants of the clinical course*. The system allowed configuration of the frequency of reports. Detection of alarm conditions, such as cardiac arrests, would trigger shorter intervals.

Freye and Eberhard in 1975 state [33, Conclusion] that "*the computer [. . .] may function as an extension of the physicians' memory, for purposes of diagnosis, therapy and prognosis*". They wanted to use the computer's capability to analyse time-series of multivariate parameters and compare them to a reference. This "distance" would then give an indication of the patient's evolution/recovery. The study was for post-heart surgery patients.

Severity scores for Intensive Care Units (ICUs) [109] take into account several parameters that are, for most scores, monitored during the patients' stay. Usually the scores are taken at specific times (e.g. at 24h, 48h, 72h) and not constantly due to the effort involved.

Fusion of multi-sensor data is used by Li et al. [65] to estimate HR. The idea is to use different sources for estimating HR, namely beat detection from ECG and Arterial Blood Pressure (ABP) waveforms. The authors state that their main innovation is using signal quality assessments to characterize both estimates. They define how to assess signal quality of ECG and ABP and use this in a Kalman filter. This allows them to use the result of the Kalman innovation of the signal in a weighted fusion of both estimates. The objective is to use the better signal in the estimation, more precisely, to have a quality weighted "use" of the signals. This reduced artefacts and noise induced estimation errors in the resulting estimation.

Some projects, seeing the need for several inputs, developed devices that are able to monitor several parameters. AMON [4] by Anliker et al. is such an example. Apart from the sensing capabilities of BP, oxygen saturation (SpO_2), skin temperature, one lead ECG and acceleration (two-axis), the device can also do online analysis and emergency detection. The wrist worn device can connect to a central unit at a telemedicine centre. Locally, it does filtering and *cleansing* of signals sensed and converts some data extracted from the ECG signal. The system does an initial evaluation by comparison to reference values, taking into account patient history. Depending

[5]This does not conflict with the previous statement regarding the `Policy` not issuing less strict requests. The `Policy` always informs of the requests made.

on the divergence from the reference, it can lead to a local reassessment or an alarm sent to the centre. The centre can do a more detailed analysis of the data uploaded given its higher processing capabilities. This *high-level medical algorithm*, as described in the paper, corrects some of the values sensed based on others; it calculates pulse based on: SpO_2, BP and ECG. Time based averaging is also used. Although our goals are somewhat different, we share the idea of aggregating information. The AMON FP5 project's objective was developing a device and support centre, whereas we aim to provide a framework to enable correlation on the BAN only.

In Middleware Linking Applications and Networks (MiLAN) [44], the work from Heinzelman et al., although related to Wireless Sensor Networks (WSNs), has influenced what we propose. They address requirements of applications (that they call Quality of Service (QoS) demands), resource limitations and cooperative applications (i.e. different applications using the same network to achieve different objectives). MiLAN uses the applications' data requests with the requirements as inputs. Each sensor or group of sensors has a level of QoS that it can provide for each data requested, which is expressed in a sensor QoS graph. They define a State-based Variable Requirement Graph (SVRG), which determines the requirements, the quality needed in the measurement and what data to be sensed from the applications. This graph is based on the current application's state. The state influences the quality needed from sensed data. All the state is calculated by the application using the raw sensor data, although the SVRG indicates some relationship between the information [44, figure 6]. The intersection of the sensor QoS set and the SVRG defines the sensors to use. In MiLAN the notion of information correlation is left to the application. The metrics defined are the quality of the values, network bandwidth and energy of the system. It is not mentioned how to re-use information for different states. Optimization is seen from a single application's point of view, however extrapolation for several applications could be achieved as an extra intersection of the different application solution sets.

In Semantic Streams [127], Whitehouse et al. use a model of "composable inference" for WSN data. They use a Constraint Language Programming (Real) (CLP(R)) framework to declare how different streams of data can be composed to produce new streams. Constraints can be defined for a query: confidence, relationship between streams (co-temporal, co-spatial). The "facts", for the streams of sensors, and "rules" for composing, including unit transformation, are defined on a Prolog engine. Optimization is supported for the defined metrics. When a query enters the system, the existing inference units can be composed to *generate new interpretations of sensor data*. It allows multiple applications, users in their nomenclature, to share the same resources on the network, while resolving conflicts. The easiness of expression, CLP(R) rules, mandates that a CLP(R) engine be available (SICStus Prolog in the paper's case) with *minutes or less* for a one time (per query) composition time. From the details on the article the proposed architecture does not seem to support composition of the same type of data streams ($\overline{Accell}$ from $\underline{Accell1}$ and $\underline{Accell2}$ in our example) so as to improve the quality/properties of a stream (e.g. minimize error). Composition rules are easy to express, an advantage of the logic programming language used, but modularity and the lack of different layers of abstraction may impair re-usability and flexibility. Moreover, processing time may be a deterrent.

The work from Gravina et al. in SPINE [38] provides a framework to distribute Signal Processing (SP) in Body Sensor Networks (BSNs). The idea is supporting several applications that use SP. Each node (sensor) has the code for the SP distributed on to it and the coordinator controls the nodes and requests output from them according to the application running. Although some objectives are similar, supporting multiple applications, the approaches are different. They assume that nodes will process data whereas we centralize it on the BS. There is no concept of

correlation as a module, although one of the points of enabling the SP is to allow the correlation of different data from the sensors. The focus is more on allowing rapid prototyping and deployment of fixed correlation (in SP) enabling code.

Signal Interpretation and MONitoring (SIMON) [26] by Dawant et al. bases its work on the premise that "*alarms are too numerous to be correctly interpreted by humans, however highly trained*". To that end a system was built, capable of adapting its monitoring according to changes in the environment and on the patient's physiological state and its predicted evolution. The filtering, artifact removal and, especially, fault detection is also part of the framework's objectives. The project developed software components for a UNIX system based on Inter Process Communication (IPC) between modules for data acquisition from external medical devices. Data collection is based on either data acquired from sensors, where there is the possibility of acquiring data from several sources, or computed from different input types. These data can have associated thresholds and clinical medical ranges, which are user-friendly interval qualifiers. This is used as input to a model-based qualitative/quantitative reasoning framework. This model defines a hierarchy of objects that represent different levels of data abstraction, with defined boundaries, and artifact and fault models for data corruption detection. As such, the data abstraction component is used to: process and abstract the data; monitor/report according to the model; detect and react to possible errors in the data; adapt in real-time its behaviour according to context. The framework has a scheduler component, a modified earliest deadline first heuristic, to adjust tasks' frequency. Clearly we share objectives, but provide different functionalities, approaches and systems. We dwell in BANs with the central component being the BS. We also aim to provide a hierarchical abstraction of the data, by defining software components that process their inputs, producing new information. Our added flexibility (the hardware abstraction and information abstraction is more modular and transparent) is counterposed with the qualitative reasoning capability [32] of SIMON.

5.3.1 Declarative languages

Some work from WSN deals with query optimization using correlation between different sources of information in the network. Examples are the work from Yao and Gehrke [138] with Cougar and Madden et al. [69] with TinyDB. These approaches advocate using declarative languages to query the sensor network that is seen as a distributed database. The reasoning follows from the highly correlated nature of sensor networks, where values being sensed are usually of the same type, leading to possible communication optimization approaches. The idea is to have query proxies on the sensor nodes that are able to process queries and decrease power consumption in nodes on two avenues: aggregation of results and selective sensing.

The fact that WSNs are multi-hop networks provides several points where aggregation can be done in order to preclude sending all data to the sink node. Thus, queries that relate to averages, maximum, minimum, etc. can be done locally by collecting data from other nodes before sending the result to the sink node.

The use of a declarative query language allows for predicate re-ordering to optimize power consumption. This means that predicates (e.g. `light>0.5`) that cost more to be evaluated are only assessed first if they are very selective, i.e. if their variation is the most relevant for including or not a reading (e.g. from the paper: using a magnetometer is more expensive than a light sensor and it is not more selective for the example query). Sensing frequency can also be lowered so as to save energy by not using the sensors. The optimization algorithm will then base its cost

function on energy spent by nodes, namely on communication and sensing. This understanding of the energy cost of a specific query plan (where to aggregate and what frequency of sensing) enables to estimate the network lifetime using a specific plan. This allows application requests to stipulate the lifetime of the network they want to have and let the query planner define the nodes to use and frequency of collection. This is done in the TinyDB[6] system. One thing to note is that, while aggregation does not lose any information (delivery is at most delayed waiting for data to be aggregated), lowering frequency of sensing may lead to event detection loss or delay. It is up to the application to define its minimum.

TinyDB also offers event triggering, i.e. it is possible to define actions (that may be new queries) on event detection using the declarative language. These events are based on conditions being monitored by the sensor network, and as such can be optimized as regular queries.

Note that the data being handled is being streamed from the sensors, i.e. it is not a fixed, limited size set of data, but something that is being continuously produced. TinyDB allows us to create storage "points" that enable definition of fixed, limited size "chunks" of data, which are time windows of the data stream.

The data stream will imply that, even with collection and aggregation optimizations, data may sometimes overwhelm the communication channel. TinyDB defines some schemes that aim to "*improve the quality of the answer*", by sending the data that will most likely improve the quality of the data already sent.

Cougar defined the basis for moving aggregation processing to sensor nodes, whereas TinyDB takes this and extends it with the selective sensing described ,i.e. , when and where to collect data.

These declarative approaches do not take into account the predictability of the data being collected, error tolerance by applications or the correlation between data (spatial, in time and from different types of data). These points are used by Deshpande et al. in BBQ [27] and developed further in their Ken system in Chu et al.'s work [20].

Basically these systems take advantage of the replicated information available (collection of the same sensing information by different nodes) and assume that sensors already provide uncertainty on the data collected[7] to define approximate queries that are error bound. For this probability distribution functions (pdfs) are used to model collected data (multivariate Gaussian in the referenced articles). The rational is to use the probability distribution to predict the requested value within an allowed error range with a certain confidence interval. If the confidence is not met, the system will need to collect data samples to refine its model and prediction.

In BBQ[8] [27], the probabilistic model runs on the sink and it requests new values when the predictions deviate. In Ken[9], the model is run at both the sink and the nodes. This allows the nodes to pro-actively push the new observations when the confidence interval is not met.

The pdf can model the spatial correlation of sensed data, however the pdf can change over time. To account for this, both approaches use a transition model that allows us to compute the pdf at time $t+1$ based on the pdf for time t. This is done using the previous observed data. In

[6] TinyDB is a database view of the sensor network built on TinyOS.
[7] The error inherent in sensors is already mentioned in Cougar [138].
[8] BBQ stands for Barbie-Q: A Tiny-Model Query System.
[9] As noted by the authors in the paper [20] Ken is "*one's range of knowledge or understanding*".

this way the observed values refine the model in time and when the confidence on the result is not met.

One key difference between BBQ and Ken is the pull versus push approach respectively. In Ken the sensor node has the ground truth and can compare it to the prediction so as to refine it and communicate discrepancies. This also allows the detection of outliers, something that is not possible with BBQ. This exchanges energy savings in sensing data for outlier detection. In both systems, the models used are key to the performance of the prediction. However, in Ken only energy savings are lost in worse models, as the sensor node has the ground truth. In BBQ, worse models can lead to a worse accuracy of the prediction. As such, in both systems, it is highly relevant to have good training data to estimate the model parameters as best as possible.

Sen and Deshpande [100] push the work further still by defining the correlation between different sensor readings[10] using probabilistic graphic models to define inference rules of feasible solutions. This allows us to define how probable the existence of the combined readings is. The correlation between them is described in a factorized form, where each factor is a function that defines the pdf of correlated attributes. The product of all factors gives the joint pdf of all attributes. Therefore the models for predictions are more detailed by correlating different readings.

Comparison

These approaches share similar objectives with our work, namely optimization of queries on correlated data. However some context differences exist that drive different approaches. WSNs, as discussed in §2.3, use several similar, if not identical, sensors to collect the same data types. This setting is of fundamental value for the declarative queries, specifically for aggregation purposes. Even correlation is mostly done on the same data type. BSNs have different sensors collecting different types of data, making them less viable for these optimizations. The likely network star-topology makes them single-hop, leading to fewer (if any) places to do aggregation.

Frequency tunning can be done on our framework in the `Policy` component. The specific policy could optimize for network lifetime and use the frequency of collection and sending as a tunable parameter. Information regarding energy consumption by nodes and their sensors is defined on the node's profile.

Our approach of using dumb sensors precludes some of the opportunities for energy saving based on predicate re-ordering. This means that it is more difficult, given our framework, to conditionally sense based on another sensed measurement. Nonetheless, recall that in §3.9.1 we described some possibilities for data processing on the nodes. It would, however, break our separation and simplicity principles to use sensed data from a different sensor service.

The Ken system [20] implies that the probabilistic models for the collected data are distributed through all sensor nodes. This implies that either the nodes are identical in terms of the data types they can sense or that different models will exist for different sensors. For aggregation, this may lead to some subtleties if sensors are aggregating data from other sensors that use different models. Recall that models can incorporate different data types, and thus different sensor nodes may have different models. Maintaining the list of models being used at the sink, may also lead to some problems if the variation is very big. Our approach of concentrating this on a single point obviates these issues.

[10] More precisely, the different tuples in the distributed database.

Action triggering based on a condition can be part of a module in our framework, being done on the BS. When the module detects the condition it actuates. This is part of the future work regarding actuation.

The ability to create time windows of stream data is achieved by the `ModuleStorage`. The module can store data that can be accessed for later usage.

Correlation of different inputs is a driving point for our framework. The declarative language approach was aimed at other objectives and only with the later work of Sen and Deshpande's [100] was this accounted for. Another initial goal of our architecture is the possibility to optimize for multiple queries, so to allow several applications requesting different types of information. This is currently not done in the declarative language framework, as the optimization is done for a single query. Again, this may be a reflex of WSNs versus BSNs, the latter are more likely to have different applications running on top of its sensor network, as sensor information is more varied. The former, in most cases, has a single purpose and collects less varied information. Nonetheless, Madden et al. in TinyDB [69] raise the issue of optimizing multiple queries as future work.

5.4 – Conclusion

In this chapter we addressed a way to distribute the information according to the model for correlating different inputs. The model from the previous chapter and the information flow described on this form the information abstraction layer of our middleware.

Returning to the objectives pointed out in §3.1, we now have that:

Convert data to human model, with metadata (B): the `ModuleSensor` provides the information layer abstraction to allow for adding the sensor data to a model and specifically to the pub/sub system. The SD from chapter 3 provides the ability to discover the nodes producing these data. The metadata that accompanies the sensor and the measurements are kept in the `DataValue` structure;

Correlate data according to models (C): the pub/sub and the model described allow the correlation of data according to models defined in the system;

Answer applications based on information model (D, 2, 3): from the previous objective, applications can receive their answers based on correlated information with the metadata accompanying it;

Optimize resource usage (E): based on the interface from the previous chapter to control nodes, and the `Policy` component described in §5.1.2, it is possible to optimize resource usage according to defined parameters. The architecture allows us to define different policies and thus use different parameters to optimize for;

Maintain the model flow of information (5): as mentioned, the pub/sub system allows for the dissemination of information through the models in the system.

Infer commonalities (III from §4.1.1): the proposed optimization algorithm takes into account all the models as a DAG. This way all common nodes, from different models, are aggregated and considered as one. After deciding which modules to use the `Policy` component issues the requirements from the strictest request made by the applications;

Regarding the remaining objectives from §4.1.1, we can add that: compartmentalization and components re-use (I) is inherent to the architecture of modules that encapsulate the

correlation between different information. Aggregation of different requests (IV) is a must in all the framework described in this chapter. And in §4.2.2 we described an algorithm to optimize resource usage taken as input applications requirements (V).

We follow from chapter 3 and provide a pictorial summary of the interconnection and information flow of the different components of the information abstraction layer in figure 5.12. We can see the flow of measurements from the hardware abstraction layer to the modules, going through the information layer view of the sensors, the module sensor. The figure also shows access to the `Policy` and `RegistrarInfo` from `RegistrarProduction` to calculate the resource tree and access pub/sub data respectively. New nodes that were not known by the system are handled by the `DaemonGWRegistrar` that creates the representation in the information abstraction layer. The flow of information is seen in the `newInput`, `newValue` to/from the modules. The applications can also pull values directly from the modules.

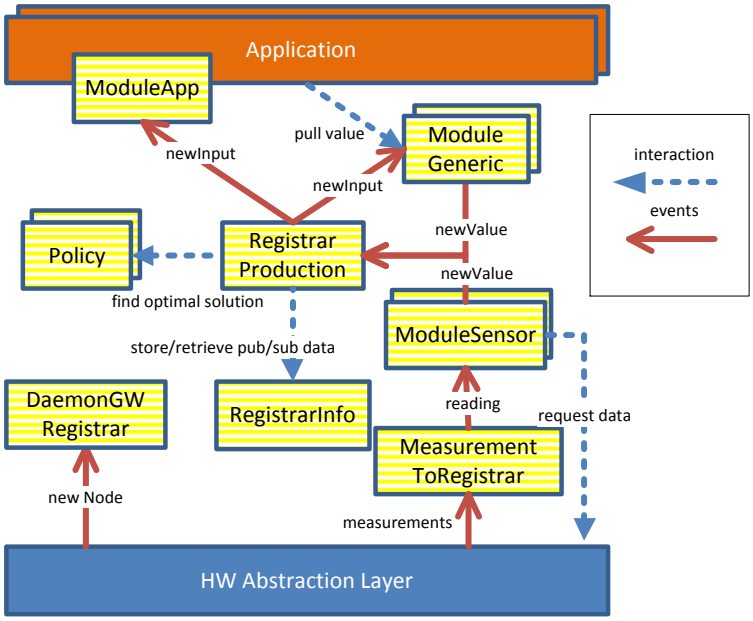


Figure 5.12 – Information layer data flow

5.4.1 Open issues

In §5.1.1 we described a structure for storing the location of nodes in the human body. However, we do not have yet a global coordinate system for it. Further research is needed to adopt a system that fits our framework, perhaps basing it on the work done by the International Society of Biomechanics [136]. Moreover, there are several medical terms databases that aim to provide a connection between the medical language and computer systems. The objective being that the latter can *understand* the former. Good examples are SNOMED-CT [55] from the International Health Terminology Standards Organisation and UMLS [72] from the USA National Library of Medicine[11]. These thesauri also define anatomy terms and thus identify body parts. Combining both systems, a coordinate accurate positioning with a recognizable body location, would provide an interesting challenge.

Signal "cleaning" modules could pose some issues for the current middleware. These type of modules would try to "cleanup" signals according to known noise models or similar approaches.

[11] UMLS combines definitions from several other databases including SNOMED-CT.

They would thus produce as output a "cleaned" version of their input, which would be the same type of information. Given the framework we described, this could cause some problems when subscribing to the information. Other modules could impose some requirements on the information that would lead the `Policy` to choose the "cleaned" version. However, if the information would not be clearly marked as a different type the "cleaning" module would receive its own output from the pub/sub system. A way to mark this functionality should be defined by the middleware so as to allow these types of modules to differentiate their own production.

The `RegistrarProduction` can have more than one `Policy`. However, when a request or a subscription is done only one policy is used to calculate the requests. The subscriber can state which policy to use for the request; this enables using different policies on different requests. A default policy is used when no policy is specified. Although this approach adds the flexibility mentioned, it also creates the problem of optimizing the different requests according to different policies (e.g. optimizing to the least error possible could conflict with energy saving). This point leads to the issue of application prioritization. Some applications might need a more detailed and "energy costly" information while others want to prolong the BAN as long as possible. Currently priorities are not handled in the framework. Priorities lead to decisions on: i) at what rate can we sense and send, ii) given current energy level what should we sense and iii) given a limited bandwidth what should we send. The centralization of the information about the network and its active functionalities enables a better planning of this. However, network usage is not easily predictable even with this centralization. Nodes, as mentioned, are dumb and will be oblivious to application priorities and data usage or its statistical significance[12]. As such, all prioritization must be done on the BS. At the optimization phase, application priority may be used to decide when conflicts occur, informing the application of the degraded quality (lower frequency of collection or sending, lower confidence interval, etc.) as we discussed in §4.3.1. When requests are already being served but there is a change in circumstances (energy going low, communication problems which lead to network congestion, etc.) the BS needs to re-evaluate the optimization based on the new context. To detect measurement losses (due to communication problems) some mechanism should be instantiated to monitor the frequency of measurements arrival. Energy thresholds can be detected with `ModuleAlarm`.

Actuators have not been addressed in this chapter. Clearly we have been focused on acquiring information as opposed to acting. Applications however, would want to act on the BAN according to the information received. Although actuators would not see a direct fit onto a model as we described, we could take the closed-loop approach mentioned in §4.2.1. In this approach, the model could be expanded to have rules to actuate on the BAN, based on the information collected, where the "output" of the modules would be commands to the actuators.

[12] In TinyDB [69], nodes prioritize data to sent based on the statistical quality improvement of sending that particular data when compared to sending other data.

6

Implementation

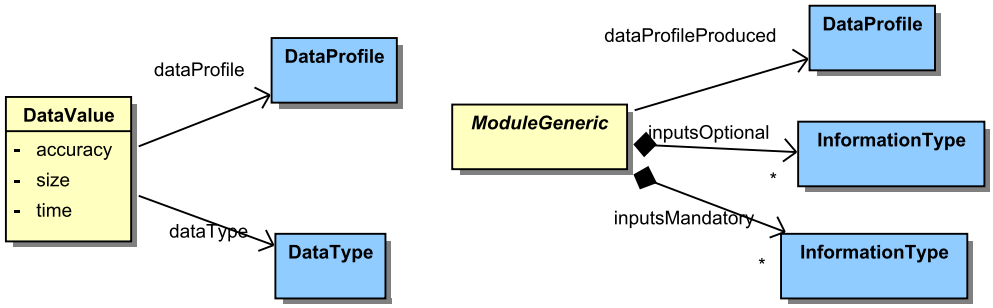
Having described the architecture, we now discuss an implementation of it. We will start by describing how the two middleware layers interact, proceed with a description of the platform used for development and discuss the Application Program Interface (API) giving some code examples to better illustrate its usage. At the end of the chapter we will make final observations on the implementation.

6.1 – Layer interaction

In §5.1.1 we saw that a `DataValue` is the structure that holds the information produced in the information layer. This value will be associated with a `DataProfile` from the hardware layer to describe it. The `DataProfile` from §3.5.1 defined the data produced. A `DataType` is the structure for handling data on the hardware abstraction layer. It allows encoding data into messages. The `DataValue` holds a `DataType` for cases where this encoding might be needed or when the `DataValue` directly results from a sensor reading, which is the case for a `ModuleSensor`. Figure 6.1.a shows these relationships and the `DataValue` attributes. Note that the accuracy will be defined by each module that processes the information, regarding the error it introduces. The time relates to the time of production. The size defines the size in bytes of the data held.

Every producer module will have associated a `DataProfile`. As described above, when producing information the new `DataValue` is associated with a `DataProfile`, thus the module needs to know the data profile it produces.

A module needs inputs to produce its outputs, they may be mandatory or optional. The optional inputs allow the module to produce outputs with different characteristics, e.g. lower



6.1.a: `DataValue` relationship with hardware layer

6.1.b: Module relationship with hardware layer

Figure 6.1 – Information layer components and hardware layer components

error. These "needs" are expressed in the dual relation to `InformationType` in figure 6.1.b. Note that some modules may not need any input, e.g. ModuleSensor gets its data "directly" from the sensors.

6.1.1 Functionality interaction

In this sub-section we will describe, through Unified Modelling Language (UML) sequence diagrams, the interaction between the two layers for some key events of the framework.

The main components of the architecture will be referred to here, so recalling from previous chapters, in the Base Station (BS):

- the `CommandDaemon` receives measurements (command reply messages) and acknowledgements of commands;
- the `ServiceDiscDaemon` sends queries to discover nodes and receives nodes' advertisements;
- the `DaemonGWRegistrar` receives information when a new node enters the system and instantiates the module sensors for the node's sensors;
- the `RegistrarProduction` is the broker for the Publish/Subscribe (pub/sub) system, where modules register their production and subscribe their inputs;
- the `MeasurementToRegistrar` receives the measurements from the `CommandDaemon` and sends them to the appropriate `ModuleSensor`.

In nodes:

- the `CommandDaemon` receives commands from the BS and sends back replies with measurements;
- the `ServiceDiscDaemon` advertises the node's capabilities and answers queries from the BS.

In the following sub-sections we will describe some of these actions.

New Node

When a new node "boots-up", it sends a broadcast message notifying of its arrival. This message is picked up by the BS and dealt with by the `DaemonGWRegistrar`. This is portrayed in figure 6.2.

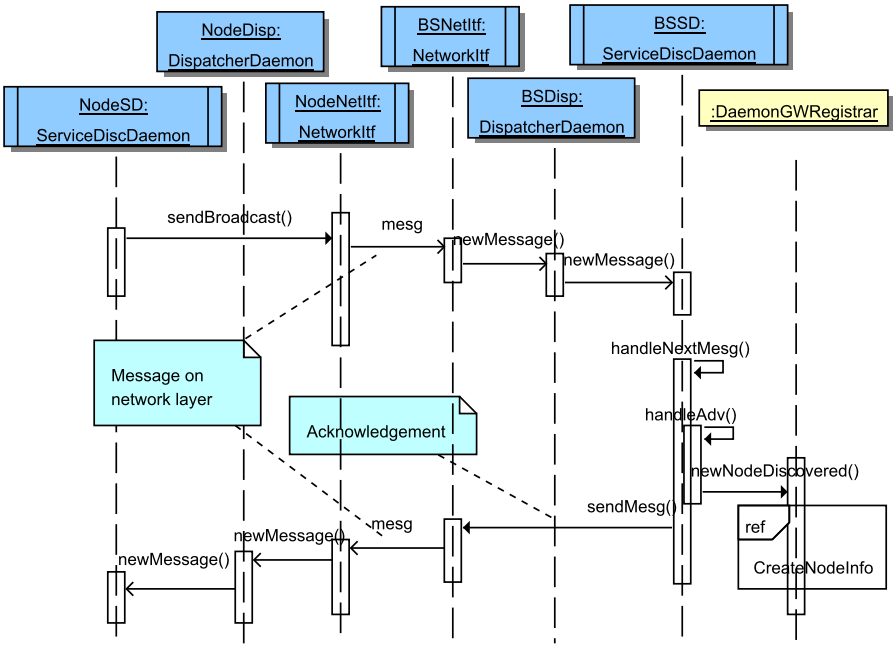


Figure 6.2 – New node Appears

In the figure we see that the `ServiceDiscDaemon` on the node is responsible for sending the broadcast message as we discussed in §3.3.2. The network interface will then send the message and deliver it to the `DispatcherDaemon` on the BS. The dispatcher will know that the `ServiceDiscDaemon` on the BS has registered for that message type and deliver the message to it. The Service Discovery (SD) daemon will, in turn, send a notification to `DaemonGWRegistrar` regarding this new node, specifying the node profile. This will allow the `DaemonGWRegistrar` to create the node representation on the information layer. We will describe this in the next sub-section. The acknowledgement is sent from the `ServiceDiscDaemon` on the BS to the SD daemon on the node.

Creation of sensor modules

After the `DaemonGWRegistrar` is notified of a new node it will instantiate its details and its sensors. Each sensor will be instantiated as a `ModuleSensor` with sensor details. The sensor module will register its production capability on the `RegistrarProduction`. These interactions are described in figure 6.3. Note that in some cases the node may already be known, e.g. if it restarted for some reason. So the `DaemonGWRegistrar` first checks this. The sensor details are always added as they might have changed, disabled, enabled, etc. The resulting relationship is the one depicted previously in figure 5.4.

Data from Node

Values produced by modules in the information layer have been depicted in §5.2 (figures 5.5 and 5.9). When the value is produced by a sensor it involves the interaction of the hardware abstraction layer with the information layer. Figure 6.4 illustrates this.

After the `SensorCollect` on the node gets its reading from the implemented `SensorService`, it will send the measurement through the command daemon. This reply message is sent by the `CommandDaemon` using the network interfaces. On the BS side the dispatcher will send the command reply message to the `CommandDaemon`. The `MeasurementToRegistrar` has

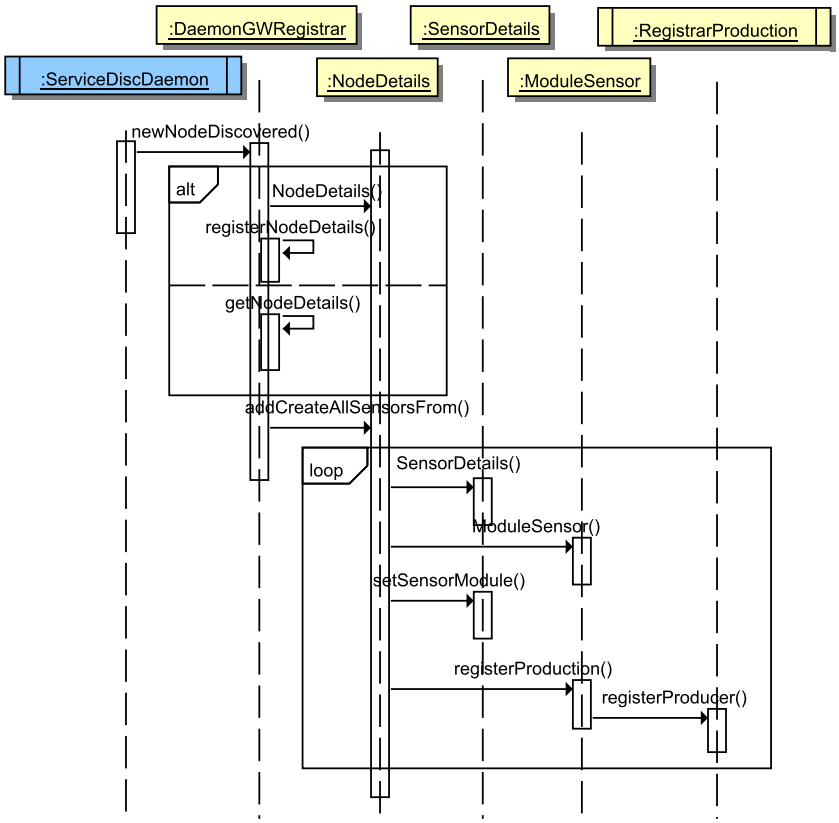


Figure 6.3 – Creation of node and sensor details

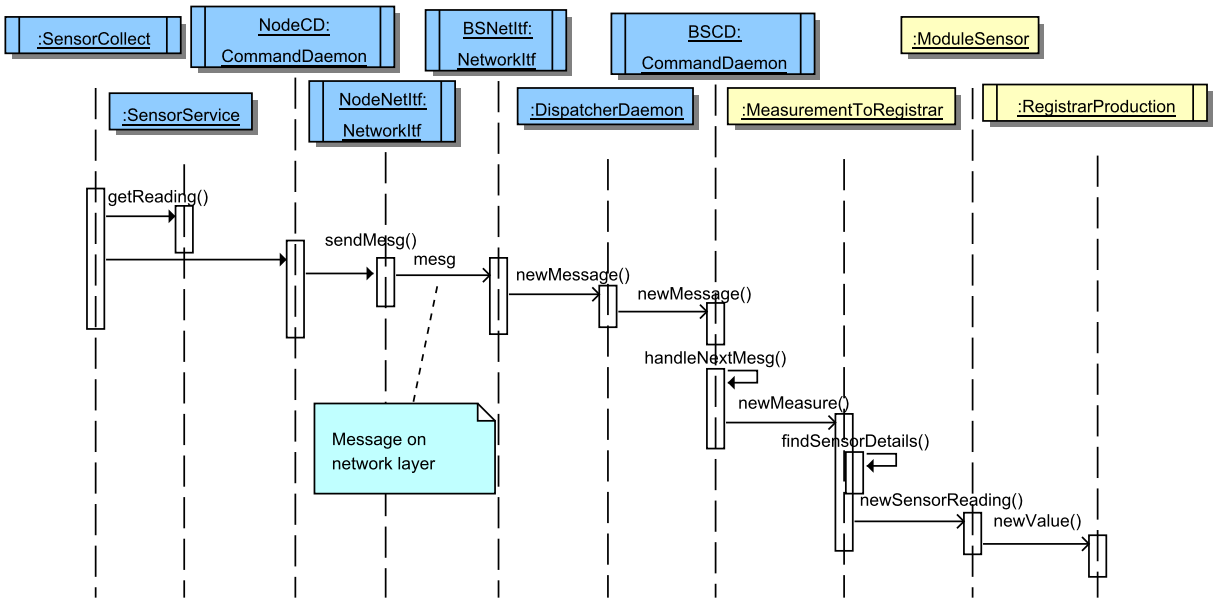


Figure 6.4 – Data new value from Sensor

registered for these messages and will thus be informed of this new measurement. After finding the sensor that is related to this measurement (recall the `UUID` on the reading) it will notify the sensor module of this new reading. The module will then publish a `DataValue` on the `RegistrarProduction` starting the information flow on the `pub/sub` system.

Bootstrap

To bootstrap the system on the BS a main process should start all required components. From the hardware layer we have `CommandDaemon` and `ServiceDiscDaemon`. From the information layer we have the `DaemonGWRegistrar`, the `RegistrarProduction` and the `MeasurementToRegistrar`.

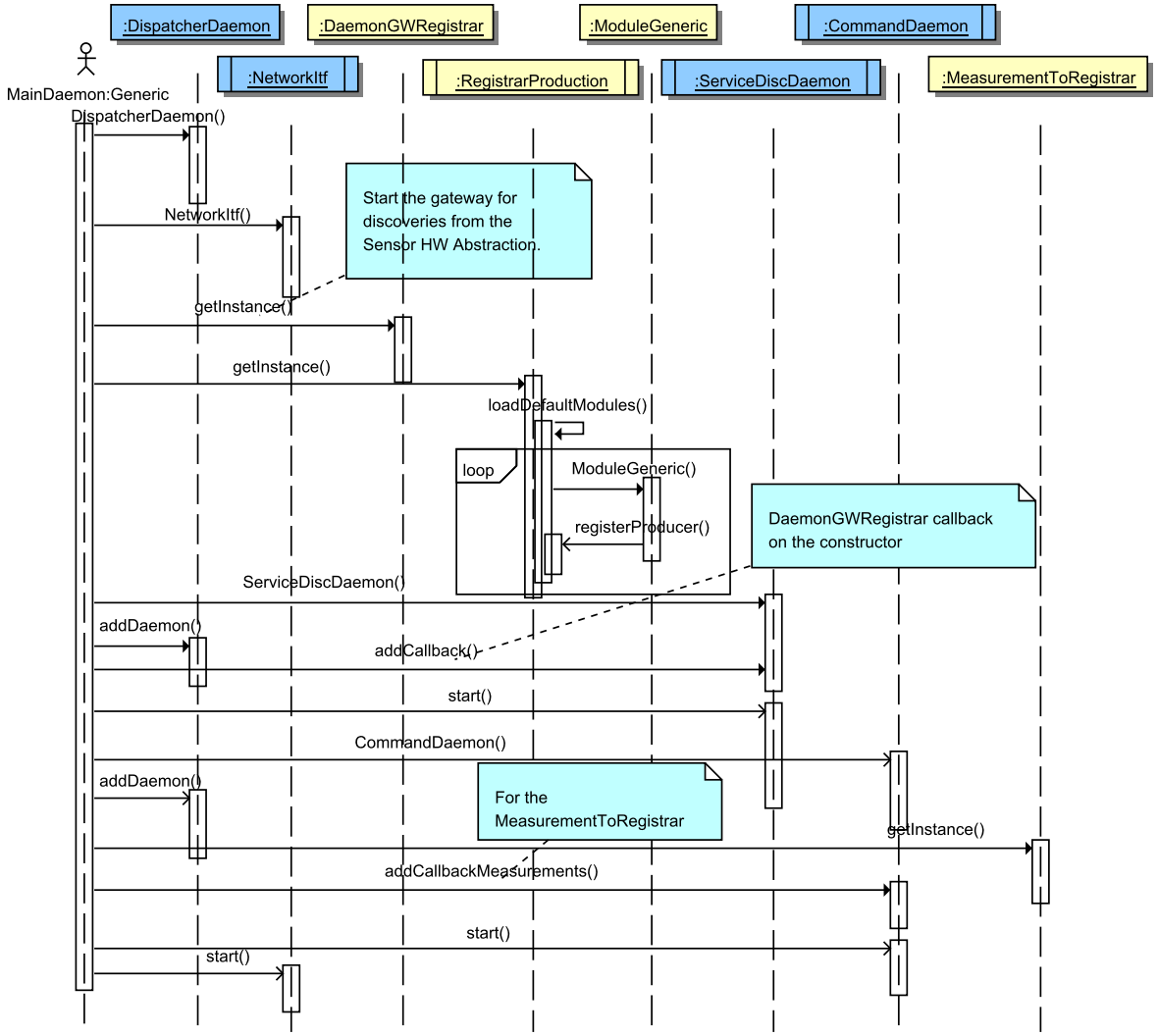


Figure 6.5 – Bootstrap

As we can see in figure 6.5, starting the daemons just involves creating and registering them on the dispatcher. This registration will enable the `DispatcherDaemon` to know which daemons are interested in which messages. The `DaemonGWRegistrar`, the `RegistrarProduction` and the `MeasurementToRegistrar` are implemented as *singletons*[1] and are instantiated by calling `getInstance()`. The `RegistrarProduction` will create the default modules it has configured when it starts. The modules, in turn, will register themselves on the `RegistrarProduction`. As we mentioned previously, the measurements received by the `CommandDaemon` will be forwarded to `MeasurementToRegistrar`. For this to happen the `MeasurementToRegistrar` is registered for a callback in the `CommandDaemon`. The interaction between `ServiceDiscDaemon` and `DaemonGWRegistrar` is similar, with a callback for new nodes discovered called on the gateway.

[1]There is only one instance of the class, which is usually accessed through a `getInstance()` method.

A node bootstrap is simpler as it only needs to start the hardware components. As such, only the instantiation of the dispatcher daemon, network interface(s), SD daemon and command daemon are needed. The callback registrations do not exist as there are no information layer components.

6.2 – Platform

As we have been discussing, the objective of our platform is to be used in a Body Area Network (BAN) setting. We built a prototype with most of the framework discussed implemented. This section will discuss some details of the platform and choices made during its development.

We did some early development on Mica2 motes. These nodes, commercialized at one point by CrossBow, can be coupled to a sensor board that adds temperature, light, magnetometer, accelerometer and microphone sensibility. They run TinyOS [121].

TinyOS is an event driven Operating System (OS) for embedded systems. The programming language for TinyOS is nesC. NesC resembles the C language in the structure and API. The architecture behind the code uses components that define an interface to be used. Components are connected to each other, so that one component can interact/use the other. Programs that run on the BS can be developed using the Java programming language.

Another option that we worked with was SunSPOTs [113]. SunSPOTs[2] are general purpose sensor nodes developed by Sun Microsystems (now Oracle) that natively run Java code (a J2ME based version) on a Virtual Machine (VM) called *Squawk*. The hardware node has already built in temperature and light sensors, a 3-axis accelerometer and connectors for adding further instruments (actuators or other sensors).

We did not evaluate other OSs (such as the Contiki multi-tasking OS [105]) as we considered the SunSPOT option as a valid one for a prototype.

The SunSPOTs provide a common development for the BS and the sensor/actuator nodes. Albeit some differences regarding the Java version (the SunSPOTs J2ME is based on Java 1.4), the code developed can be used both in the nodes and on the BS. As we will see later, this allowed us to develop common libraries that were used on nodes and on the BS.

Hardware availability was another consideration. Although, we could use the Mica2 motes, their lower specifications made them a worse choice. As an illustration, we can see in table 6.1 some of the differences of available general purpose sensor nodes. The Imote2 is a second generation platform developed by Intel, which runs an embedded version of Linux. From the table it is clear that the Imote2 has the best characteristics. However its cost made it a less attractive choice. SunSPOTs were already available at our research laboratory.

The choice for general purpose sensor nodes came from the purpose of being able to fake different types of body sensors. Being able to program a node to transmit periodically glucose values, without having a real glucometer sensor would be advantageous for testing the platform.

Using a Java platform would also mean that, in principle, it could be ported to other platforms. Most systems for a BS are able to run Java. Laptops and personal computers easily run Java, most mobile operating systems (such as Android, Symbian, BlackBerry, Maemo, Windows Mobile, see [12, 37, 70, 77, 84]) are able to directly or with some adaptations run Java.

[2]The term SPOT stands for Small Programmable Object Technology.

Table 6.1 – Node's hardware from [5, 22, 23, 62]

	Mica2	*SunSPOT*	*Imote2*
Processor	• ATM128L • up to 16 MIPS	• ARM920T 32 bit (180 MHz) • 60 to 200 MIPS	• 320/416/520 MHz PXA271 Intel XScale Processor
Memory	• 128 kB Flash • 512 kB SRAM • 4 kB EEPROM	• 4 MB Flash • 512 kB RAM	• 32 MB Flash • 32 MB SDRAM • 256 kB SRAM
Communication	868-870; 902-928 MHz IEEE 802.15.4 (TI CC1000)	2.4 GHz IEEE 802.15.4 (TI CC2420)	2.4 GHz IEEE 802.15.4 (TI CC2420)
Weight (g)	18 (exc. battery)	33	12 (exc. battery)
Dimensions (mm)	$58 \times 32 \times 7$ (exc. battery)	$71 \times 42.4 \times 23.2$	$36 \times 48 \times 9$ (exc. battery)
Cost ‡	US$ 515 for programming board + 2 nodes + 1 sensor board	US$399 for SDK with 3 nodes + base	US$990 for Imote2.Builder with 3 nodes + 2 sensor boards

‡ – prices are not all current: Mica2 from 2004; SunSPOT are current (2011) from Oracle store; Imote2 outdated, possibly from 2007.

As such we developed our middleware on the following system (recall figure 3.2):

Nodes: are SunSPOTs nodes able to run Java programs; they run the hardware abstraction layer Java library of the middleware;

BS: is a laptop where the hardware and information abstraction layers run using the Java libraries developed;

Libraries: for the hardware and information layer abstraction were developed. The hardware libraries are the same in the nodes and the BS. Nodes do not run the information abstraction library.

6.2.1 Communication

Inter process communication

The architecture for deployment, using the Java development environment, is illustrated in figure 6.6. In the figure there is a process that handles the pub/sub system of the middleware, and by extension all the BS's middleware.

The reason for this separation is to provide a centralized knowledge base and control of resources. If every application was "compiled" with all the middleware this knowledge and control would not be shared. Commonalities between requests would not be discovered, models could be different for different applications, queries would be duplicated and nodes would be independently controlled. This implies that the information abstraction components and the SD daemon should run separated from the applications' processes.

The same problem would not apply to network abstractions and the command daemon, as they do not store any knowledge, except the information regarding known nodes, which does not

conflict if duplicated.

The communication between applications and this central middleware component on the BS is done through application modules. As we will describe in §6.3.5, an application instantiates an application module (or several) that will request/subscribe to the inputs the application needs. Figure 6.6 illustrates how the Inter Process Communication (IPC) is planned to work. Applications will use the application module part of the middleware to communicate with the pub/sub system. Application modules should use a form of IPC to subscribe and get new values from the `RegistrarProduction`.

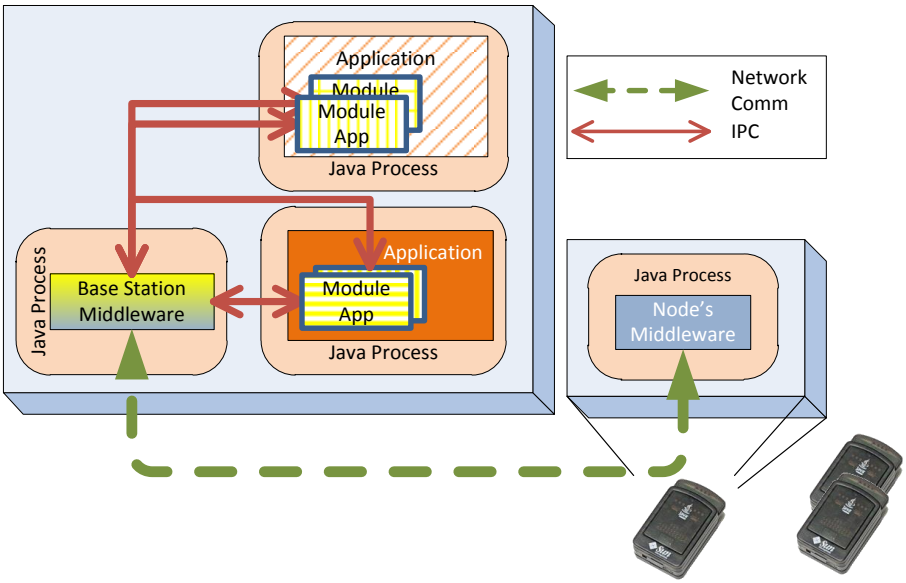


Figure 6.6 – Communication within and in/out of BS
(note that IPCs are not implemented)

Note that currently the IPC between application modules and the BS middleware is not implemented. The communication developed uses direct method calls, which means that applications are compiled with all the middleware infrastructure. This, as discussed, hinders the functioning of the central component. For testing purposes we developed an application that uses this middleware to issue several requests, simulating several applications. We will discuss this in §6.2.3.

Daemon message passing

Message passing between daemons on the same Java process (recall figure 3.4) is based on queues. The `NetworkItf` sends the raw data to the dispatcher daemon that identifies the daemon registered for that type of message. The dispatcher daemon then builds a message object according to the type and enqueues it on the registered daemon's queue. After this the `NetworkItf` goes back to listen for messages. This queue system is abstracted on a `DaemonMessaging` class, which is extended by daemons implementing messaging functionality. Currently no limit is set on the size of the queue. A timeout is defined for adding to the queue, so as not to lock the dispatcher. Receiving messages is handled in a separate thread.

In the sequence diagrams discussed, the active (running on a separate thread) components are delimited graphically by two extra vertical lines. The `DaemonGWRegistrar` and the `MeasurementToRegistrar` are not active. However, the daemons that call them spawn a thread to

call back all the registered components for the respective callback type.

6.2.2 Library details

In this prototype we have not optimized the libraries in terms of size and, as we mentioned, the same code for the hardware abstraction runs on nodes and on the BS. Therefore, the blocks from figure 3.3 on the sensor node and the BS share the same code with different instantiations in each. This leads to a current SunSPOT library size of approx 284 KB (accounting for the specific sensors of the SunSPOT). Being in Java (albeit the binary format for the SunSPOT's VM) also increases its size. The SunSPOT has 512 KB of RAM and as such the current size is not considered a problem.

As we saw in §6.1.1, when the BS bootstraps, it starts all the active components of the system. As expected those components will be "live" as long as the middleware is running. Modules instantiated in the BS will also "live" as long as the middleware is running. They will not be processing at all times as they will not be always needed. The same applies for sensor modules, they exist as long as the sensors are on the BAN, but may be idle if the sensor is not active[3]. The module alarm we discussed in §5.2.4 is an exception. As it is very specific (the thresholds are defined for a particular purpose) it will only live as long as the component that created it needs.

Messages, `DataValue`s and `DataType`s are passed by reference and the last component to use them will release the reference. Garbage collection is automatically handled by the Java VM.

6.2.3 Test application

For testing the general framework we developed a test application. The application is able to request any information produced by the available modules. Some modules will be sensor modules others will be formula based, which will need inputs from other modules. The formulae known by the middleware for this test are the ones portrayed in figure 6.7. As we mentioned previously, they are based on correlations from Law and Bukwirwa [64] for oxygen delivery (O_2Del) and the Windkessel model (see Sun et al. [111]) for Cardiac Output (CO).

For generating sensor data we developed sensor services for the SunSPOTs that produce fake values of: Heart Rate (HR), oxygen saturation (O_2Sat), hemoglobin (Hb), Stroke Volume (SV) and ElectroCardioGram (ECG) readings[4]. There is a SunSPOT representing itself, with its light, acceleration (on all axes) and temperature sensors.

There is a default policy implemented. A `Policy` is the component that performs the optimization, as described in §5.1.2. This default policy

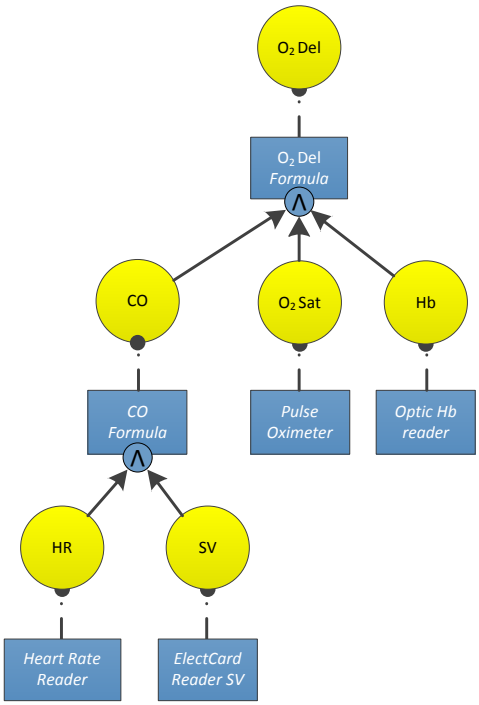


Figure 6.7 – Dependency diagram for test application

[3]Recall that the heart-beat system from §3.6.1 informs about a node's disappearance.
[4]The ECG values were based on input we collected from a wearable ECG system, Equivital from Hidalgo [46]

searches for the first modules that satisfy the requirements of a request, preferring the modules that are currently already producing. This is a simple policy that, while searching, takes into account the settings (production on or off) of the modules it needs. This is the current approach of the middleware. Policies will perform a search for the current request being able to access modules' current settings, however they do not have access to the list of active requests. This differs from the algorithm discussed in §4.2.2, which took into account all active requests. As such the implementation we made of the §4.2.2 algorithm is not yet incorporated in the middleware as a policy.

In figure 6.8 we can see a screen-shot of the test application that runs on the BS. Here we can see a list of available sensors, nodes and modules. It is possible to check the metadata for these components (the buttons *Show*) and request the output production of any of the available modules. This request will "run" the default policy for discovering the modules to produce the requested information. In the figure there are also dialogues with the values that were being requested (O_2Del, temperature and HR). Note that O_2Del is currently *Not yet Available*. This means that the middleware is lacking information that is needed for some module(s) that were on the tree found to produce O_2Del. When that information is made available (new nodes appear) the system will make the requests to the modules and start the production. The formulae for CO and O_2Del are already on the modules list. At the moment of the screen-shot, only a SunSPOT node and a (fake) Equivital node were available. This Equivital node only senses HR and temperature, so we are still missing SV, O_2Sat and haemoglobin to be able to produce oxygen delivery.

A regular application would issue the request(s) for the required information, instantiating `ModuleApp`(s) to receive the outputs. We will discuss this in §6.3.5.

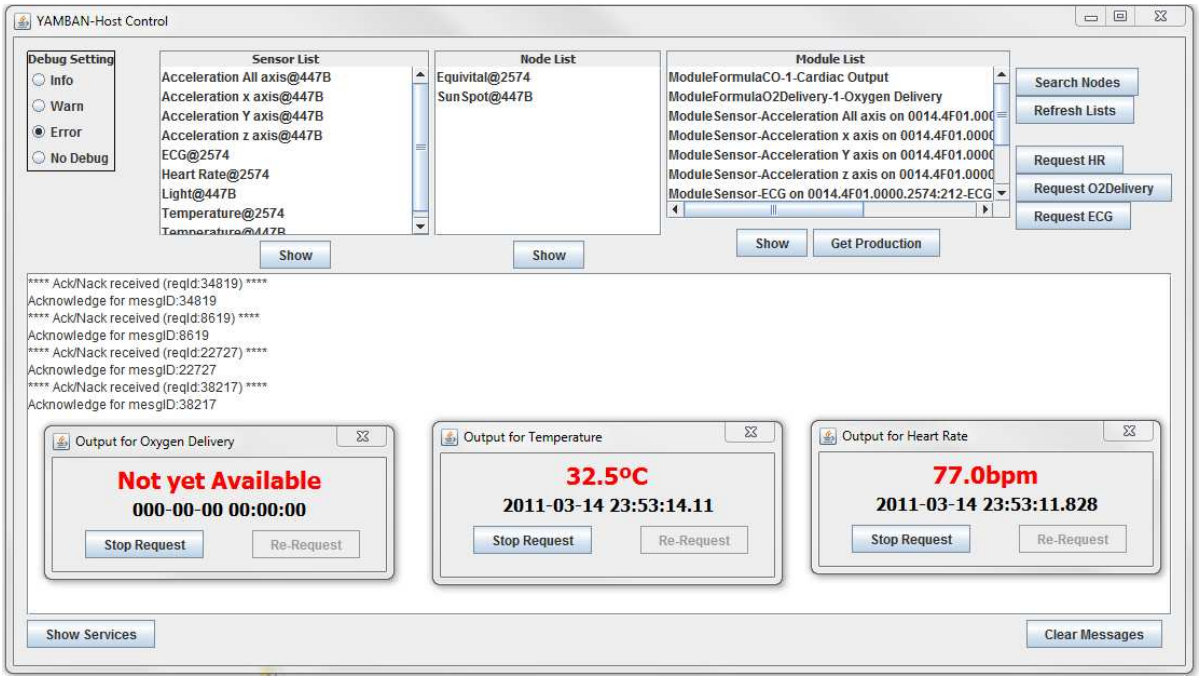


Figure 6.8 – Prototype screen-shot

6.3 – API

In this section we will describe the interfaces that need to be implemented so as to add new node types to the system and modules to the information layer; and also to develop an application that plugs into the middleware.

Depending on the hardware components of the new node and the current support on the middleware, for a new node it may necessary to develop: a new network interface, a new command daemon and new sensor services.

Adding new modules will mainly revolve around new formula correlations. This means extending `ModuleFormula`.

Last, we have the application that will have to subscribe to the information it needs using a `ModuleApplication`.

6.3.1 Network Interface

```
1  public abstract class NetworkItf extends Thread{
2     public abstract int sendMesg(String remoteAddr, byte[] bs) throws
          IOException;
3     public abstract int sendBroadcast(byte [] data) throws IOException;
4     public abstract void startListen() throws IOException;
5     public abstract void closeListen();
6     protected abstract void waitMesgsImplementation() throws IOException;
7     public abstract ReliableConnection setupReliableConnection(String
          remoteAddr) throws IOException;
8     public abstract String getLocalAddr();
9  // ...
10 }
```

Listing 6.1 – `NetworkItf` abstract methods

In listing 6.1 we have the abstract methods of the `NetworkItf` class. They define what needs to be implemented by the specific network interface.

These are the functionalities described in §3.3.1: sending messages (unicast or broadcast), receiving messages (lines 4, 5 and 6), creating reliable connections (line 7) and returning the local address of the network interface. The local address is used to define the Universal Unique IDentifier (UUID) of the sensors as described in §3.7. For the reliable connection the returned interface provides an output stream to write to, which is used to send a message reliably. It also provides an input stream to read from. These streams are backed up by the underlying reliable connection.

6.3.2 Command Daemon

Instantiating a `CommandDaemon` for a node involves implementing the methods from listing 6.2. The implementation should build the node's sensor services. It should use the sensor profile argument to instantiate a sensor service. This method is called by the `CommandDaemon`

```
1 public abstract class CommandDaemon extends DaemonMessaging {
2    protected abstract SensorService buildSensorService(SensorProfile sp);
3    protected abstract void itfsOff(NodeComm node);
4 // ...
5 }
```

Listing 6.2 – `CommandDaemon` abstract methods for a node

abstraction that knows which node profile it is instantiating and thus which sensor profiles it has (recall figure 3.7). In listing 6.3 we have an example of a command daemon for a SunSPOT. Based on the service profile code we create the appropriate sensor service passing it the sensor profile (e.g. lines 6 and 8). The `LocalStoreVector` is storage for readings that are not sent immediately by the sensor service.

```
1  public class CommandDaemonSunSpot extends CommandDaemon {
2     protected SensorService buildSensorService(SensorProfile sp) {
3        LocalStoreVector store = new LocalStoreVector();
4        switch (sp.getCode()) {
5           case SensorProfileDB.SP_CODE_SUNSPOT_LIGHT:
6              return new SensorServiceSunSpotLight(sp, store, this);
7           case SensorProfileDB.SP_CODE_SUNSPOT_TEMPERATURE:
8              return new SensorServiceSunSpotTemp(sp, store, this);
9           //...
10       }
11       return null;
12    }
13 // ...
14 }
```

Listing 6.3 – `CommandDaemon.buildSensorService` for a SunSPOT

The `CommandDaemon` also has to implement the turning on/off the node that we discuss in appendix C, to avoid detail not relevant here. Callbacks mentioned when discussing the bootstrap in §6.1.1, are also discussed in the appendix. The `CommandDaemon` extends `DaemonMessaging`, which provides methods for handling messages.

6.3.3 Sensor Service

Listing 6.4 shows an example of the code needed to implement a temperature sensor on a SunSPOT. Lines 2 and 5 are the specific parts related to the SunSPOT temperature sensor API. This would be basically what is needed for the service: to implement the `getReading(...)` method.

Note that the sensor profile is defined when building the sensor services in `CommandDaemon`. In line 9 we use the data profile from that sensor profile to set the `UUID` and check the correctness of the `DataType`.

6.3.4 Module

To see how a module receives its input we have listing 6.5, where `ModuleFormula` implements `newInput(...)`. This method is called by the registrar when a new input of the

```
1  public MeasurementBasic getReading() {
2     ITemperatureInput sensor =
          (ITemperatureInput)Resources.lookup(ITemperatureInput.class);
3     DataType dtMeas;
4     try {
5        double reading = sensor.getCelsius();
6        dtMeas = DataTypeDB.createFloat(new Double(reading));
7     }
8     /* error handling */
9     return createMeasure(dtMeas, new TimeType());
10 }
```

Listing 6.4 – `SensorService` for temperature on a SunSPOT

```
public abstract class ModuleFormula extends ModuleCalc {
   synchronized public boolean newInput(DataValue value) {
   // ...
      // if we got a DataValue we care for (is it needed)
         arrayVariable[i] = value;
      // ...
      if(!ArrayUtils.existNull(arrayVariables)){ //we got everything we need
         DataValue calc = calculateOutput();
         clearValues(); // clear our arrayVariables
         sendNewValue(calc); // send the value to the Registrar
      }
   // ...
   }
// ...
}
```

Listing 6.5 – `ModuleFormula`

information the module subscribed to is available. Remember from §5.1.1 that `ModuleFormula` calculates an output based on a correlation that is described by a formula. So `ModuleFormula` will collect its needed inputs and then calculate the output. As shown in listing 6.5, the inputs received are stored until all inputs needed are available. When that occurs the output is calculated and sent to the registrar. Note that this simple implementation does not check timestamps. It assumes that all values are good, even if they differ in time. However, it could make use of the time in the data value.

```
1  public class ModuleFormulaCO extends ModuleFormula {
2     synchronized protected DataValue calculateOutput() {
3        //CO = SV x HR
4        double svVal= (Double) arrayVariables[0].getValue();
5        double hrVal= (Float) arrayVariables[1].getValue();
6        double res = svVal * hrVal/1000; //from cm^3 to l
7        return buildValue(new Double(res));
8     }
9  // ...
10 }
```

Listing 6.6 – `ModuleFormulaCO` based on SV and HR

Listing 6.6 illustrates a module that implements the $CO = SV \times HR$ correlation. The module implements the calculate functionality of the `ModuleFormula`. Note that there is a unit conversion in line 6, but this is "hard coded". However, the module could use the relationship of `DataValue` with `DataProfile` (as in figure 6.1.a) to discover the units of the data it was handling. In line 7 a new `DataValue` is constructed associating it with the `DataProfile` of the module (see figure 6.1.b). This illustrates that modules need to associate the data profile with the values they produce so that *modules* have access to the data profile of the values they handle.

The other modules from §5.1.1 can also be extended. For example, `ModuleStorage` has an interface to store data, which should be extended for different storage types (currently there is an extension that uses a file on the file system).

6.3.5 Application

```
public class ModuleOutput extends ModuleApplication {
   public boolean newInput(DataValue value) {
      System.out.println(value.getValue().toString()+" "
         +value.getDataProfile().getUnits()+"\n");
      return true;
   }
   public void unableToProvideCallBack(int requestId, int reasonId, String
       reasonString) { /* ... */ }
// ...
}
```

Listing 6.7 – `ModuleApplication` callback methods

The application module will be the module that (as the name suggests) an application will build to subscribe to needed information. Listing 6.7 shows an example for a module that simply outputs the data it receives. The `newInput(...)` method is called when new input is available. Application modules will also implement a callback method that is called if the request is unable to be met. This allows for an asynchronous notification of the failure. What the `ModuleOutput` should also implement is the request/subscription to the information.

```
1 public boolean makeRequest(Capability capabilityNeeded){
2    Request request = new Request(this.infoWanted,capabilityNeeded);
3    this.subscriptionId =
         RegistrarProduction.getInstance().subscribeWithRequest(request, this);
4    if(this.subscriptionId == Request.INVALID_ID)
5       return false;
6    return true;
7 }
```

Listing 6.8 – `ModuleOutput` making request

Listing 6.8 gives an example of a request being made to the `RegistrarProduction`. The request needs to know what is the information needed and what are the requirements/capabilities (maximum error, delay, etc.) for getting it. In this example the information is assumed to be an attribute of `ModuleOutput` and requirements are given on the method call. A subscription identifier is returned by the subscription request, even if the request is left pending, i.e. waiting for some unavailable producer.

The application should make a request for each information type needed. It could however handle these subscriptions with the same module.

6.3.6 Moving the abstractions

The hardware abstraction used allows us to move its deployment, enabling interaction with devices with closed OSs. For example, in cases where it is not possible to program the node, it is possible to move the abstraction to the BS and make the device available to the framework. Figure 6.9 illustrates this. The node's middleware is moved to the BS and will implement the protocol to communicate with the device on the `NetworkItf`. The device will likely have a proprietary protocol. The node's middleware will then use the same approach as the rest of the middleware, serving as a proxy for all messages. The communication with the BS's middleware could use an IPC approach, if the BS has a `NetworkItf` for it.

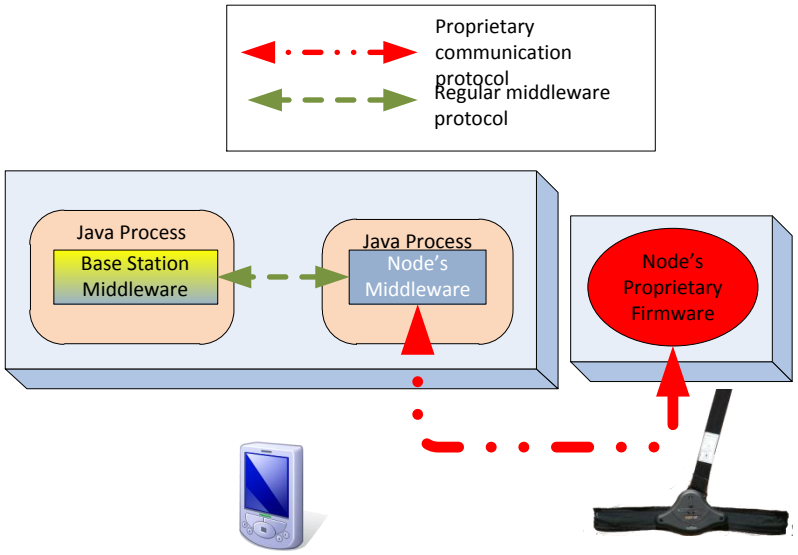


Figure 6.9 – Moving hardware abstraction

The node in figure 6.9 is an Equivital sensor unit from Hidalgo [46]. As we mentioned, this device is a strap-on with ECG, gyroscope, skin temperature and accelerometer sensors. It uses a proprietary protocol that runs over Bluetooth (serial profile) to communicate.

6.3.7 API Comments

The example from §6.3.5, gave the highest level in the hierarchy that an application developer can "plug-in" into the middleware. Nonetheless, it is possible to interact at lower layers, e.g. using the network interface or `NodeComm` for sending messages, issuing commands through the `CommandDaemon`. That is, by using the hardware abstraction directly. However, the objective of the middleware is to abstract most of that complexity from the application developer. An added bonus is centralized control and a "biasing" API that restricts applications through an interface that controls the resources, hides them even, in a centralized and optimized view.

The optimization and control of resources can be extended in the middleware by developing new `Policy`(ies) for the optimization that will imply a different usage of said resources.

The previous points imply a scenario where application developers will not have to make specific development for the nodes in the BAN. This will, of course, not be the case. Currently, in

most cases, the developer will have specific nodes to deploy for her application. Enabling the addition of new types of node is the other objective of the middleware. This allows adding new nodes that directly sense/actuate on the body, are recognized by the middleware and also to develop new modules for correlating the information sensed.

Our prototype middleware serves as a proof of concept for the need to separate both ends of the spectrum: applications on one side and resources on the other. The idea we propose is a centralization that allows correlation of different information and optimized management of the resources.

6.4 – Final Observations

In this chapter we gave some insight on the implementation of the middleware and on development using its infrastructure. Clearly several details were left untold, so we refer the interested reader to the documentation of the implementation (see appendix C).

The framework is aimed at application developers for easing their usage of BAN resources to access information related to the BAN. Moreover, it enables information correlations by re-using known and proved relationships between physiological measurements. *Re-use* is here a fundamental goal: re-use of the modules for correlation and re-use of the hardware related abstractions.

The framework also provides an easy expansion for adding new re-usable components. We gave some examples on how to do this in §6.3.

In future work, we intend to analyse performance on an optimized implementation. Tests will compare BAN's performance with and without the framework, namely: a number of requests to sensors if several applications request outputs from them, a number of different types of information available on a given model, delay for data collection the indirections introduced (for direct sensor readings), savings in resource usage with the discovery of commonality (could use the result from the number of requests). These tests involve defining what models would be used in a common setting, so as to have realistic tests. Although we suggest some possible correlations with figure 4.1, it is not completely clear what will be "a usual deployment" as the BAN community is still at the starting point of providing wider correlation opportunities to applications. A survey on regular patient monitoring, monitoring in Intensive Care Units (ICUs), sport data collection, etc. could provide some better insight on what is foreseeable for the future of applications using multi-parameter correlations on a BAN.

We will extend our prototype to cover the following points:

- when some required information is not available (e.g. for a pending request), the middleware does not query the BAN as we discussed in SD §3.6;
- known modules are currently directly instantiated by the middleware, i.e. hard-coded on the implementation. There is no provision to load them as demanded or adding new ones during run time. The Java ability to load classes during run-time could be used for applications to load new modules into the middleware without disrupting its current operation[5];
- as we mentioned in §6.2.1, the middleware should have a standalone component on the BS to enable the centralization and running applications to connect to it using some form of IPC.

[5]Apart from the re-evaluation of the optimality issues, this also raises some problems with security and application trust.

7

Conclusions and future work

We conclude the presentation of this thesis by summarizing our goals, describing what was achieved, describing some lessons learned and presenting future research and development work on several areas of Body Area Networks (BANs) related to our proposal.

7.1 – Conclusion

Our driving premise is that the healthcare proposition is evolving to a patient centred paradigm. The extant architecture will not be able to cope with the increasing costs of providing for a growing population (in terms of raw number, people affected by diseases and senior citizens). A change to a more preventive system instead of a reacting one is warranted.

We place BANs in the middle of this change, as they provide the user-focused monitoring and actuating that can lead to a system that detects anomalies sooner, with the possibility even of preventing such anomalies. Although our focus is on patient care, a BAN system is as applicable to sport monitoring instances. Specially on high profile sports that need a very accurate account of athletes' performance.

A BAN's advantage is its heterogeneous monitoring capabilities that when correlated can provide a very rich set of information. In this thesis we addressed what in our view is currently at fault in BANs: a centralized view for all the applications using the BAN to share resources and information correlation approaches. The incumbent BAN systems have mainly an "application by application" approach without considering several applications using the network in parallel. To mitigate this we propose a middleware layer that deals with this correlation.

Our architecture assumes a star topology where a central more powerful node aggregates the information. The middleware on this node (the Base Station (BS)) provides an information

abstraction. Applications will reside on this node, and use this information model to issue requests for the information they need. The middleware is responsible for correlating the information so as to respond to the applications. This involves actuating on sensor nodes to get the raw data for correlation. Actuators are also envisioned for our middleware, although current focus has been on sensor nodes.

From the introduction §1.3.1 we have developed:

Hardware abstraction: is our lowest middleware layer that provides an abstraction for the different nodes' hardware types in a BAN. This common Application Program Interface (API) eases the usage of these resources and enables a simple addition of new types. Part of this layer defines how to describe nodes' profiles. This enables a common understanding between the BS and the nodes of the resources being used. The communication protocol defined provides the abstraction in messages for a command/reply interaction. Part of this layer is a Service Discovery (SD) functionality to advertise and query the current pool of resources available to the middleware.

Models in the framework: are the focus of work. Our proposed architecture enables the usage of modules that take different inputs and correlate them to produce new information. We presented examples of current physiological formulae that allow such inference. These modules are able also to store data, calculate statistics, send alarms, i.e. handle and react to information. New modules, i.e. new correlations, storage, statistics, etc., can be added to the model with relative ease. This information abstraction layer interacts with the first to manage BAN's resources. Applications interact with the middleware through this information layer by subscribing to information needed with a set of requirements. The layer manages a Publish/Subscribe (pub/sub) system for the flow of information between modules.

Optimization: is the responsibility of the information layer when answering applications' requests. For this, it takes into account correlations known, raw sensor data availability and applications requirements. Based on this it chooses the best set of modules that produce the information requested while minimizing resource usage. These policies for optimization can be defined by the developer, and we already provide an algorithm to search the model known while minimizing a specific cost;

Prototype implementation: was developed based on Java and tested with the programmable SunSPOT nodes. It served as a proof of concept for the points discussed in the thesis, namely the ease of extension of the framework.

7.2 – Future work

As we pointed out in most chapters there are interesting subjects to be pursued and others that are needed for the completeness of the middleware. The implementation related ones are the performance tuning and evaluation, which would include some developer trials; Inter Process Communication (IPC) support in the BS; module loading at runtime and deployment on a smart phone or tablet device. Live software/firmware updates of the middleware should also be addressed, with a versioning system using the Universal Unique IDentifier (UUID) of the nodes.

Topics for further research encompass:

Models beyond formulae: are the next step for correlations. There are scenarios where even with the current parameters it is not possible to ascertain an outcome. This goes beyond the information as a quantitative value (Blood Pressure (BP) = 113/77 mmHg) to a more qualitative reasoning ("user is having an arrhythmia with 97% probability"). In this respect modules that handle fuzzy knowledge or qualitative reasoning are of interest. As an example, Otero et al. [85] use a model that relates several physiological variables over time to provide more accurate alarms in ICU. The model is defined by a network of fuzzy constraints between a set of points that describe the temporal evolution of the measured variables. Another noteworthy aspect of the work is that they provide a tool for clinicians to define the network and the variables to watch. Qualitative reasoning as described by Forbus [32] allows to define the relationships in qualitative terms and qualitative thresholds that a system can then interpret to infer some meaning. As we mention in §4.3.1 Sugeno and Yasukawa [110] propose the combination of both fields to provide fuzzy logic based qualitative models.

Actuator support: was discussed in §3.9.2. The issue pertains not only to the definition of what is the possible abstraction for an actuator, but what is the abstraction for actuating (an actuator does not need to just deliver drugs, as the artificial retina example from table 2.3 shows). An option is having categories of actuators where some fall in the drug delivery range, with quantity, duration, units, etc. as some of the characteristics and others require non-abstracted controls, like brain stimulation. Incorporating more fully actuators leads to the closed-loop control and the need to have that autonomic capability in the middleware.

Sleep patterns: influence the energy spent and the communication scheme. As we described in §3.9.4 an approach should be studied to specify when nodes can turn off most of their hardware to save energy. As communication eventually needs to occur, a duty-cycle should be defined. This raises the question of how to define these patterns: should they be defined in the node's profile as different hardware has different gains with different duty-cycles; should the BS control turning on/off, thus defining the duty cycle; should thresholds be defined and interpreted/processed by nodes, that would then wake-up to notify the occurrence, similar to the Mobisense platform [125]. Actuators do not necessarily abide by the same rules: they might not have sensors in the node for threshold appraisal and they are prone to be needed in emergency scenarios (e.g. defibrillators). The most flexible and complete approach is a mix of the three options: the BS controls duty-cycles, based on current needs and the nodes sleep patterns from their profiles. Thresholds are defined in sensor capable nodes to allow to shut down some sensors and communication system while "keeping an eye" on parameters.

Security and privacy: are important aspects of a BAN. BANs will in most cases carry personal information. It is thus important to protect the information and identify the nodes in the BAN, i.e. sensor/actuator nodes and the BS. As we talked in §3.9.4, there are approaches to generate keys using biometrics. The idea is that these biometrics are measurable only by a node on the body. Hanlen et al. [42] use movement and Poon et al. [91] use ElectroCardioGram (ECG) and PhotoPlethysmoGram (PPG) as biometrics. This enables a source of entropy to derive keys from, where a fuzzy agreement is used to guarantee that small differences in measurements originate the same keys. Not all the nodes will have the

sensors needed for the scheme (e.g. the need for ECG)[1]. The work from Falck et al. [31] uses the body itself to convey information, assuming that it cannot be eavesdropped. This is used to define a key that can be used in other communication medium. This approach is used to protect information and authenticate end-points. There are emergency situations were the data needs to be accessed. Tan et al. [115] propose a light identity based encryption that allows generating different keys to encrypt sensed data that can be recovered by trusted entities. A BAN will need some or even all of these security settings to be able to protect privacy and allow access in emergency scenarios. An added advantage of the security setting is that enables identification of the user where the BAN is. This mitigates some of the interference between different BANs.

Location within the body: will be another source of information for correlating data. BP in a leg is about 10% to 20% higher than in an arm, acceleration measured in a shoulder provides different information than when measured in a foot. As we mentioned in §5.4.1, although we have a data structure to hold the location of a node, we do not yet have the coordinate system to describe it. Input from the International Society of Biomechanics [136] for the coordinate system combined with thesauri that describe and identify body anatomy such as SNOMED-CT [55] would be a good approach.

Inter BAN communication: can provide for further parameters. Within the context described in this thesis (multi-parameter correlations), the need for these external inputs from other BANs does not seem natural. However there is scope for usage in other contexts. The business card exchange from Zimmerman [146] could be a small example, and in sport settings the exchange of performance characteristics to compare users' achievements could be another. For this communication to occur the BS would need to advertise itself as a gateway to other BANs.

Synchronization: is an important part of the system, as it influences the coordination and interpretation of collected values (timestamps). As network layers will provide some form of synchronization (e.g. the IEEE 802.15.6 task group for BAN communication [6]), the hardware abstraction should be extended to use this functionality. In cases were the network does not provide such capability, a default fall back, like the reference broadcast from Elson et al. [29] should be defined.

QoS: is needed to ensure that critical applications will be able to use the resources with higher priority, and even precluding optimization techniques in emergency scenarios. This does not affect only the communication layer but also other resources as energy and processing. The middleware should be able to convey these priority levels and use them when managing resources. Support from the communication layer is needed, which also implies extending the network abstraction to support it.

[1] The work from Poon et al. uses the ECG and PPG to derive inter pulse interval, i.e. time between heart beats. This could be accomplished with other types of sensors, like a simple heart beat sensor.

7.3 – Lessons learned

In this section we make some considerations about the work undertaken and some of the decisions made.

7.3.1 BSN versus WSN

Body Sensor Networks (BSNs) (or BANs when we account for actuators) are different in several aspects to Wireless Sensor Networks (WSNs). We addressed this issue in §2.3, but it is worth re-stating some of the points discussed.

The difference between middleware approaches for BSNs and WSNs was one of the comments usually received in submissions to conferences not specifically from the BAN field. In our view, there are differences that are usually disregarded between both networks that bring advantages and disadvantages.

In BSNs we have a centralized component, the BS, that is the center of a one hop star network topology (with the caveat discussed in §2.4.1). Communications paths are short, albeit attenuated by the human body. These are assets for communication when compared to WSNs, which in most cases need to form multi-hop hierarchies (clusters) to communicate to one or more sink nodes. This led, in our case, to the choice of having dumb nodes that send all their data *raw* to the BS, taking advantage of the fact that although communication cost is expensive it is lower than in WSN and this cost is counterbalanced by the simplicity on nodes. The centralization and small topology also helps BSNs to have a global correct knowledge of the network (the BS has this knowledge), which provides more informed ways of optimizing resource usage.

On the other hand, BSNs collect varied types of data, produced by a multitude of distinct sensors with lower redundancy when compared to WSN. This data diversity brings forth a multitude of applications that want to collect the data. Moreover, these different types of data provide more possibilities for correlations. However, it also leads to a more difficult approach to handle the different sensor hardware and the different data types.

Pointing these differences out so as to stress opportunities for new approaches was an issue that should have been pursued earlier. This would have made the research clearer early on.

7.3.2 The cost of code

Development of prototypes or demos is usually a must for systems' research. This need coupled with, usually, perfectionist researchers can lead to an over-development of the prototype code. There are different views on the amount of work that a prototype may involve. In a blog post Matt Welsh[2] advocates that software is not science and that these prototypes should be seen as "throw away" programs that only serve to prove the ideas of the papers. However, in the comments (and even in his own work) good quality code should be the foundation to yield research results. Moreover, system publications do make it (nearly) mandatory to have an implementation of the ideas. It comes naturally that the *crux* of this is the time it takes to do good implementation code.

[2] See http://matt-welsh.blogspot.com/2011/11/software-is-not-science.html.

The development of this project on BANs fell into some of the "traps" mentioned. The hardware abstraction layer is the base that the information layer uses for its models to be able to correlate the data. The development of the hardware layer could have benefited from some already developed abstraction layer. The problem resided on the capabilities needed for the layer. Namely, we needed to address every sensor the same way (abstraction achieved through the communication protocol), have service discovery announcements that used the profiles defined for the nodes, a basic functionality of sending/collecting in the sensors and to control the sensors. To the best of our knowledge, none of the architectures for BANs or WSNs provided the capabilities as we needed them. As such the development of this layer to support the information layer was pursued.

Clearly, this development had associated a time cost, but the lack of a suitable framework mandated it. Also, the lack of such a framework implies that our complete architecture (both layers) is more relevant.

Regarding "throw away" code, we believe that the re-usability, repeatability and expansion are good arguments against "throw away" code. Another argument, is that "throw away" code, as per its definition, will not be available for inspection as the code is not ready for being public. Of course, these arguments may be debatable if the implementation merely stresses a small part of a system.

A middle ground must be sought so that development of ideas is not undermined by the time needed to implement them and that this implementation would not be so impaired that it would not be useful. If for not anything else, to ease the development of other research's implementation.

7.3.3 Platform

The SunSPOTs option proved a good choice as the support and the development tools available enabled a quicker development. The Java language also lends itself to easier reuse and porting as we discussed in §6.2. The flexibility and programmability of the platform enabled programming *fake* sensors using the SunSPOTs.

The last point resulted from the lack of medical sensors in our prototype platform. The Equivital sensor we described would be a good candidate to provide a more realistic BAN. Nonetheless, apart from the Equivital type of nodes, the sensors available for research in this area are still few; medical sensor development is still needed to make devices more wearable and resilient to movement.

The option to choose *dumb* sensors for the nodes provided an easier development of the fake SunSPOTs, where code re-use provided a quick development to support "new" sensors. This provides some anecdotal evidence of that choice.

Some feedback on conferences and workshops indicated that combining different types of data collected from different devices is something that is warranted and still with few initiatives from a framework point of view.

7.4 – Publications

During the course of the thesis' work the following publications were done:

- Jatinder Singh, Pedro Brandão, Jean Bacon. "Context-aware disclosure of health sensor data" in 4th International Conference on Pervasive Computing Technologies for Healthcare 2010 Munchen Germany, March 22-25, 2010, doi: 10.4108/ICST.PERVASIVEHEALTH2010.8788
  - Pedro Brandão's work was related to the BAN sensor data collection.
- Pedro Brandão, Jean Bacon. "Body Sensor Networks: Can we use them?" in M-MPAC 2009 International Workshop on Middleware for Pervasive Mobile and Embedded Computing, Urbana Champaign, Illinois, USA Dec 2009, doi:10.1145/1657127.1657131
- Pedro Brandão, Jean Bacon. *"BSN Middleware: Abstracting Resources to Human Models"* in HealthInf, International Conference on Health Informatics Porto, Portugal, 2009, (position paper), Proceedings ISBN: 978-989-8111-63-0

A Bibliography

[1] Actigraph. GT3X+ Activity Monitor. Product specification, Actigraph, 2011 [visited June 2011]. [p. 29]

[2] Amp'ed RF Technology, Co., Ltd. BT22. Product datasheet, Amp'ed RF Technology, Co., Ltd., March 2011 [visited June 2011]. [p. 40]

[3] Gerard Anderson, Robert Herbert, Timothy Zeffiro, and Nikia Johnson. Chronic conditions: Making the case for ongoing care, September 2004 [visited April 2011]. [p. 17]

[4] U. Anliker, J.A. Ward, P. Lukowicz, G. Troster, F. Dolveck, M. Baer, F. Keita, E.B. Schenker, F. Catarsi, L. Coluccini, A. Belardinelli, D. Shklarski, M. Alon, E. Hirt, R. Schmid, and M. Vuskovic. AMON: a wearable multiparameter medical monitoring and alert system. *Information Technology in Biomedicine, IEEE Transactions on*, 8(4):415 –427, December 2004. ISSN 1089-7771. doi:10.1109/TITB.2004.837888. [pp. 69 and 95]

[5] ARM. ARM920T product overview. http://www.simplemachines.it/doc/DVI0024B_920t_po.pdf, 2003 [visited June 2011]. [p. 109]

[6] Arthur Astrin (Ed.). Draft standard for body area network. Draft, IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs), May 2010 [visited June 2011]. [pp. 32, 39, and 122]

[7] Robert Baan, Yann Grosse, Béatrice Lauby-Secretan, Fatiha El Ghissassi, Véronique Bouvard, Lamia Benbrahim-Tallaa, Neela Guha, Farhad Islami, Laurent Galichet, and Kurt Straif. Carcinogenicity of radiofrequency electromagnetic fields. *The Lancet Oncology*, 12(7):624 – 626, 2011. ISSN 1470-2045. doi:10.1016/S1470-2045(11)70147-4. [p. 36]

[8] J.E. Bardram. Pervasive healthcare as a scientific discipline. *Methods of Information in Medicine*, 47(3):178–185, 2008. ISSN 0026-1270. doi:10.3414/ME9107. [pp. 18 and 19]

[9] Michael J. Barrett, Bradford J. Holmes, and Sara E. McAulay. Healthcare unbound. Forrester Research, December 2002. [p. 19]

[10] Berkeley Bionics. Company site. http://berkeleybionics.com/, 2010 [visited June 2011]. [p. 31]

[11] Philip A. Bernstein. Middleware: a model for distributed system services. *Commun. ACM*, 39:86–98, February 1996. ISSN 0001-0782. doi:10.1145/230798.230809. [pp. 21 and 43]

[12] Blackberry. Java development on Blackberry. http://us.blackberry.com/developers/javaappdev/, 2011 [visited June 2011]. [p. 108]

[13] Bluetooth SIG. *Specification of the Bluetooth System - v2.1 + EDR*. Bluetooth Special Interest Group, July 2007. [pp. 34, 40, 52, 57, and 60]

[14] Bluetooth SIG. *Core System Package 4.0*. Bluetooth Special Interest Group, June 2010. [pp. 23, 35, and 60]

[15] Elizabeth Boehm, Bradford J. Holmes, Eric G. Brown, Lynne "Sam" Bishop, Sara E. McAulay, and Jennifer Gaudet. Who pays for healthcare unbound. Forrester Research, July 2004. [p. 19]

[16] Mike Botts and Alexandre Robin (Editors). Sensor Model Language (SensorML) implementation specification. Public Available Standard OGC 07-000, OpenGIS, July 2007. [pp. 52, 53, and 77]

[17] Michael Brodsky, Delon Wu, Pablo Denes, Charles Kanakis, and Kenneth M. Rosen. Arrhythmias documented by 24 hour continuous electrocardiographic monitoring in 50 male medical students without apparent heart disease. *The American Journal of Cardiology*, 39 (3):390 – 395, 1977. ISSN 0002-9149. doi:10.1016/S0002-9149(77)80094-5. [p. 18]

[18] Haran Burri and David Senouf. Remote monitoring and follow-up of pacemakers and implantable cardioverter defibrillators. *Europace*, 11(6):701–709, 2009. doi:10.1093/europace/eup110. [p. 18]

[19] F.L. Chan, W.Y. Chang, L.M. Kuo, C.H. Lin, S.W. Wang, Y.S. Yang, and M.S.C. Lu. An electrochemical dopamine sensor with a CMOS detection circuit. *Journal of Micromechanics and Microengineering*, 18:075028, 2008. doi:10.1088/0960-1317/18/7/075028. [p. 29]

[20] D. Chu, A. Deshpande, J.M. Hellerstein, and Wei Hong. Approximate data collection in sensor networks using probabilistic models. In *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, page 48, april 2006. doi:10.1109/ICDE.2006.21. [pp. 98 and 99]

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. ISBN 0262033844, 9780262033848. [pp. 78 and 79]

[22] Crossbow. Imote2 datasheet. Technical Report 6020-0117-02 Rev A, Crossbow, September 2007 [visited June 2011]. [p. 109]

[23] Crossbow. Mica2 datasheet. Technical Report 6020-0042-08 Rev A, Crossbow, 2009 [visited June 2011]. [p. 109]

[24] Scott E. Crouter, Kurt G. Clowers, and David R. Bassett. A novel method for using accelerometer data to predict energy expenditure. *Applied Physiology*, 100(4):1324–1331, 2006. doi:10.1152/japplphysiol.00818.2005. [p. 72]

[25] Daniel Lewis (Ed). 802.15.6 call for applications - response summary. Technical report, IEEE, January 2009 [visited June 2011]. [pp. 30 and 32]

[26] B.M. Dawant, S. Uckun, E.J. Manders, and D.P. Lindstrom. The simon project: model-based signal acquisition, analysis, and interpretation in intelligent patient monitoring. *Engineering in Medicine and Biology Magazine, IEEE*, 12(4):82 –91, December 1993. ISSN 0739-5175. doi:10.1109/51.248170. [pp. 69 and 97]

[27] Amol Deshpande, Carlos Guestrin, Samuel R. Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 588–599. VLDB Endowment, 2004. ISBN 0-12-088469-0. [p. 98]

[28] Brian Dolan. Investors pumped $233M into mobile health in 2010. Mobile Health News, January 2011 [visited April 2011]. [p. 19]

[29] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, 2002. ISSN 0163-5980. doi:10.1145/844128.844143. [pp. 67, 77, and 122]

[30] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35:114–131, June 2003. ISSN 0360-0300. doi:10.1145/857076.857078. [p. 85]

[31] Thomas Falck, Heribert Baldus, Javier Espina, and Karin Klabunde. Plug'n play simplicity for wireless medical body sensors. *Mobile Networks and Applications*, 12(2–3):143–153, June 2007. doi:10.1007/s11036-007-0016-2. [pp. 38, 67, and 122]

[32] K.D. Forbus. Qualitative reasoning. In Allen B . Tucker, editor, *Intelligent Systems*, Computer Science Handbook, chapter VII. Chapman and Hall/CRC, 2nd edition, June 2004. doi:10.1201/9780203494455.sec7. [pp. 83, 97, and 121]

[33] E. Freye and R. Eberhard. Multivariate data analysis based on a computerized patient monitoring system. *Intensive Care Medicine*, 1:193–197, 1975. ISSN 0342-4642. doi:10.1007/BF00624438. [p. 95]

[34] Adrian Friday, Nigel Davies, Nat Wallbank, Elaine Catterall, and Stephen Pink. Supporting service discovery, querying and interaction in ubiquitous computing environments. *Wirel. Netw.*, 10:631–641, 2004. ISSN 1022-0038. doi:10.1023/B:WINE.0000044024.54833.cb. [p. 60]

[35] Simson Garfinkel. *Database nation: the death of privacy in the 21st century*. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2001. ISBN 0596001053. [p. 69]

[36] Global Business Intelligence Research. Multiparameter patient monitoring market to 2016, December 2010. PRLog news [visited January 2011]. [pp. 20 and 69]

[37] Google. Java on Android. http://developer.android.com/guide/topics/fundamentals.html, 2011 [visited June 2011]. [p. 108]

[38] R. Gravina, A. Alessandro, A. Salmeri, L. Buondonno, N. Raveendranathan, V. Loseu, R. Giannantonio, E. Seto, and G. Fortino. Enabling multiple BSN applications using the SPINE framework. In *Body Sensor Networks (BSN), 2010 International Conference on*, pages 228 –233, June 2010. doi:10.1109/BSN.2010.34. [p. 96]

[39] Sandeep K. S. Gupta. Safe and dependable bio-sensor networking for pervasive healthcare. ASU Biomedical Informatics Colloquium Seminar, January 2008. Talk. [pp. 18 and 19]

[40] E. Guttman, C. Perkins, J. Veizades, and M. Day. RFC2608 - Service Location Protocol, version 2. Technical report, IETF, June 1999. Published: IETF RFC. [p. 60]

[41] Keisuke Hachisuka, Teruhito Takeda, Yusuke Terauchi, Ken Sasaki, Hiroshi Hosaka, and Kiyoshi Itao. Intra-body data transmission for the personal area network. *Microsystem Technologies*, 11:1020–1027, 2005. ISSN 0946-7076. doi:10.1007/s00542-005-0500-1. [p. 37]

[42] Leif W. Hanlen, David Smith, Jian Andrew Zhang, and Daniel Lewis. Key-sharing via channel randomness in narrowband body area networks: is everyday movement sufficient? In *Proceedings of the Fourth International Conference on Body Area Networks*, BodyNets '09, pages 17:1–17:6. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009. ISBN 978-963-9799-41-7. [pp. 67 and 121]

[43] M.A. Hanson, H.C. Powell, A.T. Barth, K. Ringgenberg, B.H. Calhoun, J.H. Aylor, and J. Lach. Body area sensor networks: Challenges and opportunities. *Computer*, 42(1): 58–65, January 2009. ISSN 0018-9162. doi:10.1109/MC.2009.5. [p. 39]

[44] W. B Heinzelman, A. L Murphy, H. S Carvalho, and M. A Perillo. Middleware to support sensor network applications. *Network, IEEE*, 18:6–14, February 2004. doi:10.1109/MNET.2004.1265828. [pp. 52, 63, and 96]

[45] Marco Hernandez, Haruka Suzuki, and Ryuji Kohno. HARQ for High QoS Applications of BANs. Presentation, IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs), November 2009 [visited June 2011]. [p. 38]

[46] Hidalgo. Equivital life monitoring. http://equivital.co.uk/, 2011 [visited May 2011]. [pp. 33, 111, and 117]

[47] Peter Hunter, Peter Robbins, and Denis Noble. The IUPS human physiome project. *Pflugers Arch*, 445:1–9, 2002. doi:10.1007/s00424-002-0890-1. [p. 52]

[48] IEEE. IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Standard 802.15.4-2006, IEEE, 2006. (Revision of IEEE Std 802.15.4-2003). doi:10.1109/IEEESTD.2006.232110. [p. 34]

[49] IEEE. IEEE 802.15 TG-6 Body Area Networks (BAN), 2011 [visited May 2011]. [pp. 30, 35, and 47]

[50] IEEE Computer Society. IEEE standard for information technology— telecommunications and information exchange between systems— local and metropolitan area networks— specific requirements. Standard, IEEE, 2007. [p. 35]

[51] IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. IEEE Standard 754-2008, IEEE, August 2008. doi:10.1109/IEEESTD.2008.4610935. [p. 57]

[52] IEEE/ISO. IEEE 11073 - Personal Health Device Communication. Standard 11073, IEEE/ISO, 2004. [p. 52]

[53] IEEE/ISO. IEEE 11073 - Personal Health Device Communication - Part 10201: Domain information model. Standard 11073-10201:2004(E), IEEE/ISO, 2008. [p. 57]

[54] IEEE/ISO. IEEE 11073 - Personal Health Device Communication Part 20601: Application profile- Optimized Exchange Protocol. Standard 11073-20601-2008, IEEE/ISO, 2008. [pp. 27 and 57]

[55] IHTSDO. SNOMED Clinical terms user guide. User guide, International Health Terminology Standards Development Organisation, January 2010 [visited June 2011]. [pp. 101 and 122]

[56] Hyung il Park, In gi Lim, Sungweon Kang, and Whan woo Kim. Human body communication system with fsbt. In *Consumer Electronics (ISCE), 2010 IEEE 14th International Symposium on*, pages 1 –5, June 2010. doi:10.1109/ISCE.2010.5523268. [p. 37]

[57] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11:2–16, February 2003. doi:10.1109/TNET.2002.808417. [p. 63]

[58] International Commission on Non-ionizing Radiation Protection. Publications. http://www.icnirp.org/PubEMF.htm, 2011 [visited June 2011]. [p. 36]

[59] ISO/IEC. Information technology – Telecommunications and information exchange between systems – High-rate ultra-wideband PHY and MAC standard. Standard ISO/IEC 26907:2009, ISO/IEC, November 2009. Revision of ISO/IEC 26907:2007. [p. 35]

[60] James Keener and James Sneyd. *Mathematical physiology II: Systems physiology*, volume 8/2 of *Interdisciplinary Applied Mathematics*. Springer Verlag, 2nd edition, 2009. ISBN 978-0-387-79387-0. [pp. 22 and 70]

[61] Prajakta Kulkarni and Yusuf Öztürk. Requirements and design spaces of mobile medical care. *SIGMOBILE Mob. Comput. Commun. Rev.*, 11:12–30, July 2007. ISSN 1559-1662. doi:10.1145/1317425.1317427. [p. 19]

[62] Sun Labs. Sun SPOT Theory of Operation. Manual Red release 5.0, Sun Microsystems, 2009 [visited June 2011]. [p. 109]

[63] Benoît Latré, Bart Braem, Ingrid Moerman, Chris Blondia, and Piet Demeester. A survey on wireless body area networks. *Wireless Networks*, 17:1–18, 2011. ISSN 1022-0038. doi:10.1007/s11276-010-0252-4. [pp. 33 and 34]

[64] R. Law and H. Bukwirwa. The physiology of oxygen delivery. *Update Anaesthesia*, 10:1–2, 1999. [pp. 41, 72, and 111]

[65] Q Li, R G Mark, and G D Clifford. Robust heart rate estimation from multiple asynchronous noisy sources using signal quality indices and a Kalman filter. *Physiological Measurement*, 29(1):15, 2008. [pp. 60, 69, and 95]

[66] Joshua Lifton, Deva Seetharam, Michael Broxton, and Joseph Paradiso. Pushpin computing system overview: A platform for distributed, embedded, ubiquitous sensor networks. *Pervasive Computing*, 2414:605–614, 2002. doi:10.1007/3-540-45866-2_12. [p. 37]

[67] Yibo Ling, Terrence Pong, Christophoros C Vassiliou, Paul L Huang, and Michael J Cima. Implantable magnetic relaxation sensors measure cumulative exposure to cardiac biomarkers. *Nat Biotechnol*, 29(3):273–277, Mar 2011. doi:10.1038/nbt.1780. [pp. 29 and 31]

[68] R. Stuart Mackay. Radio telemetering from within the body. *Science*, 134:1410, 1961. doi:10.1126/science.134.3488.1410-a. [p. 20]

[69] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30:122–173, March 2005. ISSN 0362-5915. doi:10.1145/1061318.1061322. [pp. 97, 100, and 102]

[70] Maemo Community. Java on Maemo. http://wiki.maemo.org/Java, 2011 [visited June 2011]. [p. 108]

[71] J.J. Mastrototaro. The minimed continuous glucose monitoring system. *Diabetes technology & therapeutics*, 2(1, Supplement 1):13–18, 2000. doi:10.1089/15209150050214078. [p. 29]

[72] Bethesda (MD). UMLS Reference Guide. Technical report, National Library of Medicine (US), September 2009. [p. 101]

[73] Medtronic Minimed. Product site. http://www.minimed.com/products/insulinpumps/features/index.html, 2011 [visited June 2011]. [p. 31]

[74] George B. Moody. ECG-based Indices of Physical Activity. *Computers in Cardiology*, 19:403–406, 1992. doi:10.1109/CIC.1992.269499. [p. 72]

[75] Jin-Hee Moon, Dong Hyun Baek, Yoon Young Choi, Kwang Ho Lee, Hee Chan Kim, and Sang-Hoon Lee. Wearable polyimide–pdms electrodes for intrabody communication. *Journal of Micromechanics and Microengineering*, 20(2):025032, 2010. doi:10.1088/0960-1317/20/2/025032. [pp. 37 and 38]

[76] A. Natarajan, B. de Silva, Kok-Kiong Yap, and M. Motani. To hop or not to hop: Network architecture for body sensor networks. In *Sensor, Mesh and Ad Hoc Communications and Networks, 2009. SECON '09. 6th Annual IEEE Communications Society Conference on*, pages 1 –9, June 2009. doi:10.1109/SAHCN.2009.5168978. [pp. 46 and 47]

[77] Nokia. Java on Symbian. http://www.developer.nokia.com/Devices/Symbian/#article4, 2011 [visited June 2011]. [p. 108]

[78] Nonin Medical Inc. Onyx II® Model 9560 Bluetooth® Fingertip Oximeter. OEM Specification and Technical Information 6470-000-01, Nonin Medical Inc, 2008 [visited June 2011]. [p. 29]

[79] A.V. Nurmikko, J.P. Donoghue, L.R. Hochberg, W.R. Patterson, Yoon-Kyu Song, C.W. Bull, D.A. Borton, F. Laiwalla, Sunmee Park, Yin Ming, and J. Aceros. Listening to brain microcircuits for interfacing with external world–progress in wireless implantable microelectronic neuroengineering devices. *Proceedings of the IEEE*, 98(3):375 – 388, March 2010. ISSN 0018-9219. doi:10.1109/JPROC.2009.2038949. [p. 29]

[80] E. Nuxoll and R. Siegel. Biomems devices for drug delivery. *Engineering in Medicine and Biology Magazine, IEEE*, 28(1):31 –39, January-February 2009. ISSN 0739-5175. doi:10.1109/MEMB.2008.931014. [p. 31]

[81] National Institute of Mental Health. Brain stimulation therapies. http://www.nimh.nih.gov/health/topics/brain-stimulation-therapies/brain-stimulation-therapies.shtml, November 2009 [visited June 2011]. [p. 31]

[82] J. Olivo, S. Carrara, and G. De Micheli. Energy harvesting and remote powering for implantable biosensors. *Sensors Journal, IEEE*, 11(7):1573 –1586, July 2011. ISSN 1530-437X. doi:10.1109/JSEN.2010.2085042. [p. 40]

[83] OMG. OMG Unified Modeling Language (OMG UML), Infrastructure, February 2009. [pp. 24 and 89]

[84] Oracle. Java on Windows ME devices. http://www.oracle.com/technetwork/java/javame/javamobile/download/sdk/, 2009 [visited June 2011]. [p. 108]

[85] Abraham Otero, Paulo Félix, Senén Barro, and Francisco Palacios. Addressing the flaws of current critical alarms: a fuzzy constraint satisfaction approach. *Artificial Intelligence in Medicine*, 47(3):219 – 238, 2009. ISSN 0933-3657. doi:DOI: 10.1016/j.artmed.2009.08.002. [p. 121]

[86] Oxford University Press. The Oxford Dictionary of Sports Science & Medicine. Oxford Reference Online, 2007. [p. 72]

[87] M. Pacelli, G. Loriga, N. Taccini, and R. Paradiso. Sensing fabrics for monitoring physiological and biomechanical variables: E-textile solutions. In *Medical Devices and Biosensors, 2006. 3rd IEEE/EMBS International Summer School on*, pages 1–4, September 2006. doi:10.1109/ISSMDBS.2006.360082. [p. 37]

[88] J.A. Paradiso and T Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4(1):18–27, Jan-Mar 2005. ISSN 1536-1268. doi:10.1109/MPRV.2005.9. [pp. 34 and 40]

[89] M. Patel and Jianfeng Wang. Applications, challenges, and prospective in emerging body area networking technologies. *IEEE Wireless Communications Magazine*, 17(1):80–88, 2010. doi:10.1109/MWC.2010.5416354. [p. 34]

[90] PwC Pharmaceuticals. Medical technology innovation scorecard. Technical report, PriceWaterhouseCoopers, January 2011. [p. 19]

[91] C.C.Y. Poon, Yuan-Ting Zhang, and Shu-Di Bao. A novel biometrics method to secure wireless body area sensor networks for telemedicine and m-health. *Communications Magazine, IEEE*, 44(4):73 – 81, April 2006. ISSN 0163-6804. doi:10.1109/MCOM.2006.1632652. [pp. 67 and 121]

[92] E.R. Post and M. Orth. Smart fabric, or "wearable clothing". *Wearable Computers, 1997. Digest of Papers., First International Symposium on*, pages 167–168, October 1997. doi:10.1109/ISWC.1997.629937. [p. 37]

[93] L. Rabiner and B. Juang. An introduction to hidden Markov models. *ASSP Magazine, IEEE*, 3(1):4 – 16, January 1986. ISSN 0740-7467. doi:10.1109/MASSP.1986.1165342. [p. 69]

[94] Redpine Signals, Inc. RS9110-N-11-02 802.11bgn WLAN Module. Product Datasheet Versiion 1.46.1, Redpine Signals, Inc., February 2011 [visited June 2011]. [p. 40]

[95] Eric Renard, Guy Costalat, Hugues Chevassus, and Jacques Bringer. Closed loop insulin delivery using implanted insulin pumps and sensors in type 1 diabetic patients. *Diabetes Res Clin Pract*, 74 Suppl 2:S173–S177, December 2006. doi:10.1016/S0168-8227(06)70026-2. [p. 73]

[96] Shad Roundy, Dan Steingart, Luc Frechette, Paul Wright, and Jan Rabaey. Power sources for wireless sensor networks. In Holger Karl, Adam Wolisz, and Andreas Willig, editors, *Wireless Sensor Networks*, volume 2920 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin / Heidelberg, 2004. doi:10.1007/978-3-540-24606-0_1. [pp. 34 and 40]

[97] Salutation Consortium. Salutation architecture specification (part 1), version 2.0c. Technical Report 2.0, The Salutation Consortium, June 1999. [p. 60]

[98] Brent Schlender. Intel's Andy Grove On The Next Battles In Tech. Fortune, May 2003 [visited April 2011]. [pp. 18 and 19]

[99] J. Scott, F. Hoffmann, M. Addlesee, G. Mapp, and A. Hopper. Networked surfaces: a new concept in mobile networking. In *Mobile Computing Systems and Applications, 2000 Third IEEE Workshop on.*, pages 11–18, 2000. doi:10.1109/MCSA.2000.895377. [p. 37]

[100] Prithviraj Sen and A. Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 596 –605, april 2007. doi:10.1109/ICDE.2007.367905. [pp. 99 and 100]

[101] Shimmer-Research. Wireless GSR Sensor. Product specification, Shimmer-Research, 2011 [visited June 2011]. [p. 29]

[102] Victor Shnayder, Borrong Chen, Konrad Lorincz, Thaddeus R. F. Fulford-Jones, and Matt Welsh. Sensor networks for medical care. Technical report, Division of Engineering and Applied Sciences Harvard University, 2005. [p. 63]

[103] Feng Shu and Guido Dolmans. Qos support in wireless bans. Presentation, IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs), November 2008 [visited June 2011]. [p. 38]

[104] Herbert Shubin and Max Harry Weil. Efficient Monitoring with a Digital Computer of Cardiovascular Function in Seriously Ill Patients. *Annals of Internal Medicine*, 65(3): 453–460, 1966. doi:10.1059/0003-4819-65-3-453. [pp. 69 and 95]

[105] SICS. Contiki OS. http://www.sics.se/contiki/, 2001 [visited June 2011]. [p. 108]

[106] Smartex. Company site. http://smartex.it/, 2011 [visited June 2011]. [p. 37]

[107] Daniel Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 296–305, New York, NY, USA, 2001. ACM. ISBN 1-58113-349-9. doi:10.1145/380752.380813. [p. 78]

[108] J. Steffan, L. Fiege, M. Cilia, and A. Buchmann. Towards multi-purpose wireless sensor networks. In *Proc. Systems Communications*, pages 336–341, 2005. doi:10.1109/ICW.2005.77. [p. 34]

[109] K. Strand and H. Flaatten. Severity scoring in the ICU: a review. *Acta Anaesthesiologica Scandinavica*, 52(4):467–478, 2008. ISSN 1399-6576. doi:10.1111/j.1399-6576.2008.01586.x. [pp. 69 and 95]

[110] M. Sugeno and T. Yasukawa. A fuzzy-logic-based approach to qualitative modeling. *Fuzzy Systems, IEEE Transactions on*, 1(1):7, Feb 1993. ISSN 1063-6706. doi:10.1109/TFUZZ.1993.390281. [pp. 83 and 121]

[111] J.X. Sun, A.T. Reisner, M. Saeed, and R.G. Mark. Estimating cardiac output from arterial blood pressure waveforms: a critical evaluation using the mimic ii database. In *Computers in Cardiology, 2005*, pages 295 –298, September 2005. doi:10.1109/CIC.2005.1588095. [pp. 21, 22, 41, 72, and 111]

[112] Sun Microsystems. Jini specifications, v. 2.1,. Technical report, Sun Microsystems, 2005. [p. 60]

[113] SUN/Oracle. SunSPOT. http://www.sunspotworld.com/, 2004 [visited June 2011]. [pp. 24, 48, and 108]

[114] A.N.N.M. Swartz, S.J. Strath, D.R. Bassett, W.L. O'Brien, G.A. King, and B.E. Ainsworth. Estimation of energy expenditure using CSA accelerometers at hip and wrist sites. *Medicine & Science in Sports & Exercise*, 32(9):S450–S456, 2000. ISSN 0195-9131. [p. 72]

[115] Chiu C. Tan, Haodong Wang, Sheng Zhong, and Qun Li. Body sensor network security: an identity-based cryptography approach. In *Proceedings of the first ACM conference on Wireless network security*, WiSec '08, pages 148–153. ACM, ACM, 2008. ISBN 978-1-59593-814-5. doi:10.1145/1352533.1352557. [p. 122]

[116] Texas Instruments. CC2420 - 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver. Product specification, Texas Instruments, March 2007 [visited June 2011]. [p. 40]

[117] Texas Instruments. CC2540 2.4-GHz Bluetooth® low energy System-on-Chip. Product specification, Texas Instruments, May 2011 [visited June 2011]. [p. 40]

[118] Texas Instruments. MSP430x2xx Family. User guide, Texas Instruments, April 2011 [visited June 2011]. [p. 39]

[119] Textile Wire. Technical brochure. http://www.textile-wire.ch/fileadmin/download/Techn_Brosch_TW_en_def_may2011.pdf, 2011 [visited June 2011]. [p. 37]

[120] Textronics. Energy-activated fabrics. http://www.textronicsinc.com/products/, 2011 [visited June 2011]. [p. 37]

[121] TinyOS team. TinyOS. http://www.tinyos.net/, 2000 [visited June 2011]. [pp. 50 and 108]

[122] UPnP Forum. UPnP device architecture. version 1.0. Technical report, UPnP Forum, July 2006. [p. 60]

[123] C.N. Ververidis and G.C. Polyzos. Service discovery for mobile ad hoc networks: a survey of issues and techniques. *Communications Surveys Tutorials, IEEE*, 10(3):30 –45, quarter 2008. ISSN 1553-877X. doi:10.1109/COMST.2008.4625803. [p. 60]

[124] Agustinus Borgy Waluyo, Isaac Pek, Song Ying, Jiankang Wu, Xiang Chen, and Wee-Soon Yeoh. LiteMWBAN: a lightweight middleware for wireless body area network. In *Medical Devices and Biosensors, 2008. ISSS-MDBS 2008. 5th International Summer School and Symposium on*, pages 141–144, June 2008. doi:10.1109/ISSMDBS.2008.4575038. [p. 63]

[125] Agustinus Borgy Waluyo, Wee-Soon Yeoh, Isaac Pek, Yihan Yong, and Xiang Chen. Mobisense: Mobile body sensor network for ambulatory monitoring. *ACM Trans. Embed. Comput. Syst.*, 10:13:1–13:30, August 2010. ISSN 1539-9087. doi:10.1145/1814539.1814552. [pp. 63, 67, and 121]

[126] J.D. Weiland, W. Liu, and M.S. Humayun. Retinal prosthesis. *Annu. Rev. Biomed. Eng.*, 7: 361–401, 2005. doi:10.1146/annurev.bioeng.7.060804.100435. [p. 31]

[127] Kamin Whitehouse, Feng Zhao, and Jie Liu. Semantic streams: A framework for composable semantic interpretation of sensor data. In Kay Römer, Holger Karl, and Friedemann Mattern, editors, *Wireless Sensor Networks*, volume 3868 of *Lecture Notes in Computer Science*, pages 5–20. Springer Berlin / Heidelberg, 2006. doi:10.1007/11669463_4. [p. 96]

[128] Bruce L. Wilkoff, Angelo Auricchio, Josep Brugada, Martin Cowie, Kenneth A. Ellenbogen, Anne M. Gillis, David L. Hayes, Jonathan G. Howlett, Josef Kautzner, Charles J. Love, John M. Morgan, Silvia G. Priori, Dwight W. Reynolds, Mark H. Schoenfeld, and Panos E. Vardas. HRS/EHRA Expert Consensus on the Monitoring of Cardiovascular Implantable Electronic Devices (CIEDs): Description of Techniques, Indications, Personnel, Frequency and Ethical Considerations. *Europace*, 10(6):707–725, 2008. doi:10.1093/europace/eun122. [p. 18]

[129] World Health Organization. The global burden of disease: 2004 update, 2004 [visited April 2011]. [p. 17]

[130] World Health Organization. Working together for health, the world health report 2006, 2006. Overview. [p. 18]

[131] World Health Organization. Life expectancy at birth, 2008 [visited April 2011]. [p. 17]

[132] World Health Organization. Fact sheets: chronic diseases, 2008-2011 [visited April 2011]. [p. 17]

[133] World Health Organization. Global health atlas, 2009. Data Statistics of Global Health [visited April 2011]. [p. 18]

[134] World Health Organization. Global health risks, mortality and burden of disease attributable to selected major risks, 2009 [visited April 2011]. [p. 17]

[135] World Health Organization. Electromagnetic fields. http://www.who.int/peh-emf/en/, 2011 [visited June 2011]. [p. 36]

[136] Ge Wu and Peter R. Cavanagh. ISB recommendations for standardization in the reporting of kinematic data. *Journal of Biomechanics*, 28(10):1257–1261, 1995. ISSN 0021-9290. doi:10.1016/0021-9290(95)00017-C. [pp. 101 and 122]

[137] Guang-Zhong Yang, Omer Aziz, Benny Lo, Ara Darzi, Bhavik A. Patel, Costas A. Anastassiou, Danny O'Hare, Anna Radomska, Suket Singhal, Tony Cass, and Heny Higgins. *Body Sensor Networks*, volume XXVIII of *User Interfaces, HCI and Ergonomics*. Springer, 1st edition, 2006. ISBN 978-1-84996-569-9. [p. 33]

[138] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31:9–18, September 2002. ISSN 0163-5808. doi:10.1145/601858.601861. [pp. 97 and 98]

[139] T. Zasowski, F. Althaus, M. Stager, A. Wittneben, and G. Troster. Uwb for noninvasive wireless body area networks: channel measurements and results. In *Ultra Wideband Systems and Technologies, 2003 IEEE Conference on*, pages 285 – 289, November 2003. doi:10.1109/UWBST.2003.1267849. [p. 35]

[140] Zephyr Technologies. Company site. http://www.zephyr-technology.com/, 2011 [visited June 2011]. [p. 29]

[141] Bin Zhen, Maulin Patel, SungHyup Lee, EunTae Won, and Arthur Astrin. TG6 technical requirements document. Trd, IEEE P802.15 Working Group for Wireless Personal Area Networks (WPANs), September 2008. [pp. 32, 34, and 36]

[142] Gang Zhou, Jian Lu, Chieh-Yih Wan, M.D. Yarvis, and J.A. Stankovic. BodyQoS: Adaptive and Radio-Agnostic QoS for Body Sensor Networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 565 –573, April 2008. doi:10.1109/INFOCOM.2008.105. [p. 38]

[143] F. Zhu, M. W Mutka, and L. M Ni. Service discovery in pervasive computing environments. *Pervasive Computing, IEEE*, 4:81–90, December 2005. doi:10.1109/MPRV.2005.87. [p. 60]

[144] ZigBee Alliance. Zigbee specification version 1.0. Technical Report 1.0, ZigBee Alliance, June 2005. [pp. 23, 34, 52, and 60]

[145] T.G. Zimmerman. Personal area networks: near-field intrabody communication. Master's thesis, Massachusetts Institute of Technology, September 1995. [p. 20]

[146] T.G. Zimmerman. Personal area networks: near-field intrabody communication. *IBM Systems Journal*, 35(3.4):609–617, 1996. ISSN 0018-8670. [pp. 20, 37, and 122]

B

Acronyms

ABP	Arterial Blood Pressure
API	Application Program Interface
BAN	Body Area Network
BAWSN	Body Area Wireless Sensor Network
BCC	Body Coupled Communication
BP	Blood Pressure
BS	Base Station
BSN	Body Sensor Network
CAN	Controller Area Network
CLRS	Cormen Leiserson Rivest Stein
DAG	Directed Acyclic Graph
DALY	Disability Adjusted Life Year
CLP(R)	Constraint Language Programming (Real)
CO	Cardiac Output
COPD	Chronic Obstructive Pulmonary Disease
CPU	Central Processing Unit
ECG	ElectroCardioGram
FEC	Forward Error Correction
GPS	Geographical Positioning System
GW	Gateway
HBC	Human Body Communication
HCI	Human Computer Interface
HR	Heart Rate
I^2C	Inter-Integrated Circuit
IARC	International Agency for Research on Cancer
ICNIRP	International Commission on Non-Ionizing Radiation Protection
ICU	Intensive Care Unit
IDE	Integrated Development Environment
IECD	Implantable Electronic Cardiovascular Device

IEEE Institute of Electrical and Electronics Engineers

IO Input/Output

IP Internet Protocol

IPC Inter Process Communication

IT Information Technology

J2ME Java 2 platform Micro Edition

JVM Java Virtual Machine

LED Light Emitting Diode

MAC Media Access Control

MET Metabolic Equivalent

MiLAN Middleware Linking Applications and Networks

MDER Medical Device Encoding Rules

MRI Magnetic Resonance Imaging

MTU Maximum Transmission Unit

OS Operating System

P2P Peer to Peer

PAN Personal Area Network

PC Personal Computer

PDA Personal Digital Assistant

PHY Physical

PPG PhotoPlethysmoGram

pub/sub Publish/Subscribe

QoS Quality of Service

RAM Random Access Memory

SAR Specific Absorption Rate

SD Service Discovery

SDK Software Development Kit

SDP Service Discovery Protocol

SIMON Signal Interpretation and MONitoring

SLP Service Location Protocol

SN Sensor Network

SNOMED-CT Systematized Nomenclature of Medicine Clinical Terms

SP Signal Processing

SPI Serial Peripheral Interface

SPOT Small Programmable Object Technology

SVRG State-based Variable Requirement Graph

SV Stroke Volume

SW Software

pdf probability distribution function

PDR Packet Delivery Ratio

PIN Personal Identification Number

USA United States of America

UML Unified Modelling Language

UMLS Uniform Medical Language System

UMTS Universal Mobile Telecommunications System

UPnP Universal Plug and Play

URL Uniforme Resource Location

UUID Universal Unique IDentifier

UWB Ultra Wide Band

VM Virtual Machine

WBAN Wireless Body Area Network

WHO World Health Organization

WCND Worst Case Number of Descendants

WCP Worst Case Possibilities

WPAN Wireless Personal Area Network

WSN Wireless Sensor Network

C

Implementation details

This appendix describes some of the implementation specifics that, although relevant, fell outside of the level of detail for the main part. We describe some aspects of the Application Program Interface (API) and then mention the software used in the development of this thesis, including typesetting this document.

JavaDoc documentation for the code can be seen at http://www.cl.cam.ac.uk/~pb405/implementation/doc/.

C.1 – API

This section discusses some details that were not addressed in §6.3 for the API.

When in §6.3 we discussed the `CommandDaemon` abstract interface we did not address the details for turning the node on/off. This functionality is done by implementing the method `cmdStateCh_impl(...)` (listing C.1, line 2). The parameter `on` defines if it is to set it on (`true`) or off (`false`). The parameter for `NodeComm` identifies the communication component used for sending the acknowledgement after the on/off, so the interface associated with this `NodeComm` should stay on for the acknowledgement. After the acknowledgement is sent, the last interface should be turned off. For this `itfsOff(...)` is used; the `NodeComm`'s interface is now turned off. This is not a two phase approach; this separation is just so that the `cmdStateCh(...)` on the `CommandDaemon` abstraction can control the off process acknowledging it or not depending on the result and only then turning the last part off.

Although not needed for implementing a `CommandDaemon`, it may be relevant to know the interface for callbacks we mentioned when discussing the bootstrap in §6.1.1. In listing C.1,

```
1 public abstract class CommandDaemon extends DaemonMessaging {
2    protected abstract void cmdStateCh_impl(boolean on, NodeComm nodeComm)
         throws IllegalStateException;
3    protected abstract SensorService buildSensorService(SensorProfile sp);
4    protected abstract void itfsOff(NodeComm node);
5 // ...
6    public void addCallbackMeasurements(CallBackCmdMeasurements toCall) { /* ...
         */}
7    protected void cmdStateCh(boolean on, MessageStream mesg) { /* ... */}
8 // ...
9 }
```

Listing C.1 – `CommandDaemon` abstract methods for a node

line 6 we show a method for adding a callback for getting measurements when they are received; note that it is not filtered by information type, all measurements received are notified.

Listing C.2 shows the interfaces to implement to register for these callbacks. The `CallBackCmdMeasurements` allows aggregated (bulk) measurements and single measurements to be received. The command daemon also allows for command replies to be received, namely acknowledgements.

```
public interface CallBackCmdMeasurements {
   public abstract void newMeasure(MeasurementBasic measure, int reqId);
   public abstract void newMeasuresAggSimp(MeasurementBasic[] measuresAggSimp,
       int reqId);
}
```

Listing C.2 – `CommandDaemon` callback interfaces

As expected, the `ServiceDiscDaemon` implements the same procedure, where a `CallBackSD` interface allows for notification on node discovery and removal. The daemons maintain a list of the registered callbacks and call them on a separated thread.

C.2 – Software Used

The work of this thesis relied on several packages of software. A word of thanks is due to the people involved in their development, as most were developed in open source and without any company backing.

We list the software used for the development and for typesetting the thesis.

C.2.1 Development

Development was mostly done in Windows 7 64 bits and Java SunSPOTs with some excursions to Linux. Software used was

- Eclipse IDE
- Tortoise SVN client for windows and Eclipse plug-in

- Tigris SVN server for Linux
- BOUML UML software
- Sun's Java 64 bits
- SunSPOT's SDK
- Serial port classes for Java 64 bits
- Google's Window Builder Pro for Eclipse
- Doxygen documentation generation tool
- Foswiki (Twiki at first) for wiki documentation

C.2.2 Typesetting

The following programs were used for typesetting the thesis document:

- Vim as text editor
  - Latex-Suite for vim
  - tags plugin for vim
- JabRef for bibtex reference managing
- SumatraPDF and Acrobat Reader
- Microsoft Office suite: MS Word, MS Visio, MS Excel
- ctags program for windows
- Perl 2.10 for scripting help
- Tex configuration/style from Markus Kuhn, Sriram Srinivasan and also from tex.ac.uk and eng.cam.ac.uk/help/tpl/textprocessing/

The thesis was typeset in MS Windows using the MikTex 2.9 latex distribution. The following packages were used:

- acronym
- algorithm
- algorithmicx
- algpseudocode
- amsbsy
- amsfonts
- amsgen
- amsmath
- amsopn
- amstext
- array
- atbegshi
- atveryend
- babel
- backref
- bigintcalc
- bitset
- caption
- caption3
- charter
- color
- colortbl
- courier
- etexcmds
- fancyhdr
- filecontents
- fix-cm
- float
- fontenc
- gettitlestring
- graphics
- graphicx
- hobsub
- hobsub-generic
- hobsub-hyperref
- hopatch
- hycolor
- hyperref
- ifluatex
- ifpdf
- ifthen
- ifvtex
- ifxetex
- imakeidx
- infwarerr
- inputenc
- intcalc
- keyval
- kvdefinekeys
- kvoptions
- kvsetkeys
- letltxmacro
- lettrine
- lipsum
- listings
- lstlang1
- lstmisc
- ltxcmds
- microtype
- multicol
- multirow
- nameref
- natbib
- needspace
- paralist
- pdfescape
- pdftexcmds
- refcount
- rerunfilecheck
- rotating
- snapshot
- soul
- subfig
- subfiles
- suffix
- tabularx
- threeparttable
- titlesec
- trig
- uniquecounter
- url
- verbatim
- wrapfig
- xcolor-patch
- xkeyval
- xspace

D

Index