JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

Distributed Activity Recognition from Acceleration Data

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Masterstudium

PERVASIVE COMPUTING

Eingereicht von:
Michael Josef Haslgrübler-Huemer, Bakk. techn.

Angefertigt am:
Institut for Pervasive Computing

Beurteilung:
Univ.-Prof. Mag. Dr. Alois Ferscha

Mitwirkung:
Dipl.-Ing. Mag. Dr. Clemens Holzmann

Linz, Mai 2011

Affidavit

I hereby declare that the following master thesis "Distributed Activity Recognition from Acceleration Data" has been written only by the undersigned and without any assistance from third parties.

Furthermore, I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself. Linz, at the 15th June 2011

Michael Haslgrübler

Acknowledgment

I would like to say thank you to Professor Alois Ferscha and my adviser Clemens Holzmann for enabling me to work on this thesis and their support through the writing and implementation of this work.

I would like to thank Matthias and Vanessa for proof-reading. Johannes and Reinhard for fruitful discussions about the content of this work. My family, my friends and my colleagues for their support throughout my studies.

Zusammenfassung

Mit der steigenden Anzahl an Sensoren in unserem Umwelt steigt der Wunsch diese Sensoren wiederzuverwenden. Traditionelle Verfahren, welche mehrere Sensoren verwenden und deren Daten kombinieren, sind dazu aber nicht in der Lage, weil die typischerweise verwendete statische Konfiguration dies nicht zulässt. Ein Sensor-System, welches diese Limitierung überwinden möchte, muss aus Sensoren bestehen, welche ihre eigenen Fähigkeiten vermitteln können und sich über die Fähigkeiten (bei / an/ über) anderen Sensoren informieren können. Dadurch wird es möglich, dass Sensoren ein gemeinsames Ziel, wie das Erkennen von Bewegungsaktivitäten, erreichen können, indem sie sich zusammenschließen und ihre Fähigkeiten kombinieren.

Diese Arbeit beschreibt ein durch ein Sensorziel initiiertes Aktivitätserkennungsverfahren, welches sich auf die Verwendung von mehreren Sensoren stützt und wie dieses Verfahren auf einem eingebetteten System implementiert ist. Weiters wird erklärt und festgehalten wie Selbst-Organisation, Selbst-Management und Selbst-Adaption in diesem Verfahren dazu beitragen und dazu führen, dass das Sensor-System in der Lage ist, vorhandene Sensoren bestmöglich wiederzuverwenden, effizient Daten zu sammeln und zu transportieren, die Datenverarbeitungkapazitäten der Sensoren nutzt und das Sensor-System gegen spontan auftretende Fehler in, und Ausfällen von, einzelnen Sensoren schützt, ohne dabei die Aktivitätserkennung zu unterbrechen.

Die Implementierung dieses Verfahrens wurde im Framework **D**arSens realisiert, welches in der Lage ist Aktivitätserkennung auf einen von Menschen getragenen Sensor System durchzuführen, während die Klassifizierung und Feature Extraktion auf den einzelnen Sensoren durchgeführt wird, ohne dabei auf die Datenverarbeitungsfähigkeiten eines Anwendungsgerät zurückzugreifen zu müssen. Es wurden drei Experimente durchgeführt, welche die Machbarkeit des Verfahrens bestätigen und das Laufzeitverhalten des Systems in typischen Aktivitätserkennungsszenarien untersuchen.

Abstract

As the amount of sensors in our environment grows, it is feasible to reuse existing sensors. Traditional multi-sensor fusion strategies in place in todays sensor systems are not capable for this as they use a static configuration which cannot be reused. In order to create a sensor system which can overcome this limitation, it is essential that a sensor node can convey its capabilities and inquiry about other sensor node's capabilities. In turn, sensor nodes can then cooperate to achieve a common sensing goal by combining their capabilities and e.g. recognize locomotion activities.

This work describes the sensing goal initiated multi-sensor activity recognition approach and how it is implemented on an embedded system platform. As well as how mechanisms of self-organization, self-management and self-adaptation are incorporated in the approach, and the implementation and how they benefit the sensor system in terms of making best use of available sensors, efficient data gathering and delivery, using the processing capabilities of sensors and protecting the sensor ensemble against spontaneous and occasional sensor faults or breakdowns without disrupting the activity recognition process. The implementation of this approach is implemented in a framework called **D**arSens which is able to perform activity recognition on a body sensor network with the feature extraction and classification performed within the sensor network without needing to revert to the processing capabilities of a client device using the sensor ensemble. Three experiments have been conducted to show the feasibility and performance of the approach in a typically activity recognition task.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter motivates the work by showing the central theme ranging from the early days of personal computing over to the present and suggesting a way for the future. The introduction also outlines the goal and the research questions of this work and gives an overview about this thesis.

1.1 Motivation

When computers started to be accessible for personal use in the late 1980s more and more people started to use them e.g. for gaming purposes, household financing or simple replacing typewriters and writing then documents digitally. In the early days a profound knowledge was necessary to operate a computer. Over the following years a lot of effort was made to make computers more usable for laymen.

Nowadays computers are usable for laymen but they still depend a lot upon the knowledge of their users. The human being has to actively work with and focus on the computer in order to achieve a goal. To advance further, it is necessary to shift the focus of attention away from the computer and from operating it towards achieving a goal implicitly. E.g. when entering a conference room it is "socially" expected to turn off the sound of a mobile phone – now a user has to manually get his mobile phone out of his pocket and set his device accordingly. The user has to interact actively with his mobile phone to achieve this goal.

Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. [1]

However if the mobile phone would recognize the context the user is in, it could accomplish this task without interaction of the user. Recognizing the context of a user is a challenging task which may involve using multiple types of sensors, multiple computing devices and, to be more general, involves accessing information from a highly heterogeneous environment.

1.2 Activity recognition

Context recognition composes of different types of sub fields, one of this is activity recognition. Which in turn is a broad research field in itself. This work focuses on human activity recognition. Which is a stepping stone to build computer system that can implicitly perform tasks which would otherwise require human interaction. In order to recognize the activity a human is performing, we need sensors. There are a vast amount of different sensor types available. However most research projects incorporate and emphasize the use of acceleration sensors. While early research work in this field started out using single sensor systems the trend is to use more than one sensor.

When looking at nowadays multi-sensor systems, they either consist of multiple sensors of the same type [22] or use multiple sensors of different types [24]. The more sensors are incorporated in such systems, the bigger is the need to coordinate the data flow in such systems, especially if the used communication channel is wireless. Typically, the systems are centralized, having one coordinator with centralized control, although there is the beginning of a paradigm shift towards decentralized control or more distributed approaches [13].

1.3 Goal and research question

The main task of this work is to create a distributed activity recognition system which should be composed of sensor nodes which are equipped with an acceleration sensor.

These sensor nodes should be attached to the human body and recognize the activity the wearer is performing. The recognition process should use the data obtained by the acceleration sensors of those sensor nodes and an exchange of their data should happen wirelessly and support the recognition task.

Furthermore the system should make best use of available sensors in the environment and manage and use their resources (e.g. processing power or battery) without tailoring down the system to a special activity recognition problem.

The system needs to perform sensing, processing and communication of activities recognized. The sensing goal specifies what needs to be recognized and the system will take care of translating this into a machine-readable format, which will be used to self-organize the available sensors.

Derived from this the research question is at follows:

RQ: What are the potentials and (observable) benefits of a sensing goal initiated multi-sensor activity recognition approach as opposed to traditional multi-sensor fusion strategies?

a) What are appropriate policies for spontaneous sensor ensemble configurations, and how can they be derived from sensing missions?

b) How can mechanisms of self-organization (as seen from an individual sensor) and self-management (as seen from sensor ensembles) be embedded into opportunistic sensing strategies?

c) How can the run-time operation of goal driven, opportunistic sensing ensembles be secured and protected against spontaneous and occasional sensor faults?

1.3.1 Hypotheses

ad RQ) A major benefit when using a sensing goal initiated multi-sensor activity recognition approach is, that the system is capable of recognizing different sets of activities at runtime, by only changing the sensing goal unlike traditional systems, where this has to be done at compile time. Furthermore it is also possible to detach the link between the activity recognition and the types and amount of sensors needed for the recognition task, insofar as for a sensing goal multiple sensor ensemble configurations, aka sensing missions, can be available and used based on the available sensor nodes within the environment.

ad RQ a) The sensing mission constraints the sensor to ensemble themselves to fit the configuration specified by the sensing mission. Insofar as the sensing mission specifies from which body parts sensor data needs to be gathered, how it needs to be combined and processed to fulfil the sensing goal. However, unlike static configurations used in traditional sensor fusion approaches, where the exact sensor needs to be specified, e.g. by the MAC address, only the body positions from which data needs to be collected constrains the sensor ensemble. To be more general this approach only constraints the sensor ensemble by specifying which sensor data needs to be gathered but not from which sensor.

ad RQ b) In order to perform opportunistic sensing, an individual sensor needs to be able to communicate with his environment, the other sensors. Using a shared interaction protocol its possible for a sensor to specify what 'duties' are needed to be performed by other sensors and what the sensor itself is capable of. Using the self-organization approach the sensor ensemble is able to perform self-management because upon receiving a sensing mission, a sensor node will try to satisfy the goal specified within it with the help of other sensor nodes. Within the sensor ensemble communication channels will be opened where necessary to deliver data from the sensors to the receiver of the recognition chain.

ad RQ c) As easily as opportunistic sensing ensembles can come into existence, they can also be disturbed by unforeseeable change within their loosely coupled environment. It is essential to provide a mechanism to mitigate this change, the disappearance of sensor nodes. This can be achieved by letting the system manage itself using the same self-organization technique that was used to form the sensor ensemble in the first place and

replacing the disappeared sensor nodes by other sensor nodes found in the environment if possible.

1.3.2 Key benefits

The key benefits of using a sensing goal initiated multi-sensor activity recognition approach, as implemented in this work, are:

goal change at run-time the recognition goal of the sensor ensemble can be changed at runtime

no exact sensor specification its not necessary to specify which sensor to use but only which data needs to be gathered

interaction protocol for self-organization sensors can cooperate using an interaction protocol which enables them to state their needs and their own capabilities and enables them to fulfil a sensing goal

run-time reconfigurability sensors can detect faults of other sensors and replace those sensors at run-time without disturbing the recognition task

1.4 Outline

The outline of this thesis is as follows:

- Chapter 2 provides an overview about the existing work in the activity recognition and distributed (sensor) systems.
- Chapter 3 provides a conceptual description about the framework for distributed activity recognition which incorporates sensor data acquisition, feature extraction, machine learning within a sensor network without centralized control.
- Chapter 4 provides details about the implementation, architecture and design of the framework on an embedded system platform with wireless communication facilities and different sensors.

- Chapter 5 provides an evaluation of the framework and implementation by performing three types of experiment, which show the feasibility of the approach taken.

- Chapter 6 sums up the paper by drawing a conclusion and indicating how future work could further improve the work already done this thesis.

Chapter 2

Related Work

This chapter covers an introduction to pattern recognition concepts. Additionally related work in the context of activity recognition is presented and key problems are identified which lead towards the development of opportunistic activity recognition. Furthermore parts of this chapter focus on how to self-organize components in a sensor system and what efforts have been made regarding distributed service discovery.

2.1 Pattern recognition and the recognition chain

Pattern Recognition - the act of taking in raw data and taking an action based on the "category" of the pattern - . . . [7]

The above statement by Duda et al. [7] defines what pattern recognition is. Several steps of work have to be done in order to perform this action. These steps which have to be executed sequentially are called the recognition chain and are incorporated in every activity recognition system:

- extraction of raw sensor data - sensor data needs to be sampled from a sensor – e.g. an acceleration sensor – by converting analog measurements into a digital representation

- pre processing - unnecessary information for the total outcome of the recognition chain will be stripped from the raw data gathered, e.g. removal of noise generated by an acceleration sensor by applying a low pass filter.

- feature extraction - the raw data contains certain properties or features which can be measured and used for further processing, e.g. when using a three-axis-acceleration sensor a property would be the orientation of the sensor [27]

- classification - based upon the features or measurement taken a decision will be made, e.g. based on the orientation of an acceleration sensor attached to a human body it will be decided if this person is standing or lying.

- post processing - based upon the classification results and a-priori knowledge, post processing can be applied to smooth the overall outcome of a recognition chain. E.g. a person will not be able to switch from lying to standing and back within a few milliseconds therefore a system will omitting such unreasonable classification results and instead alter the output of the classification from standing to lying.

2.2 Activity recognition systems

2.2.1 Wearable sensor badge and jacket

The sensor badge [9], which is one of the earliest and most successful work in activity recognition, consists of acceleration sensors which were then used to measure forward

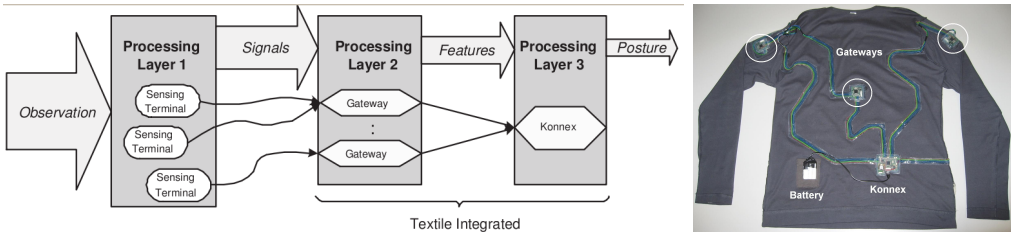
and upward acceleration. To distinguish between static postures (standing, sitting, lying) and dynamic activities (walking) the difference between minimum and maximum measured values were used over a sampling window of one second with 20Hz. To distinguish between the static postures the average of the sampling window was computed and assigned to a posture. To distinguish between walking and running the maximum and minimum on both axis was used. The vertical maxima, unlike the horizontal maxima, differs significantly and was used as an indicator for distinction. Additionally the frequency of the movement was a second indicator, which was estimated by the average crossing rate, a feature which is also used in this work. The overall detected state was visualized through a set of LEDs, whereas each LED was assigned to an activity or posture.

The sensor jacket complements the sensor badge and consists of knitted stretch sensors and knitted conductive sensors which were integrated into the fabric. It was used to detect the elbow and wrist positions of the wearer but had to be calibrated for each user. Data was sampled in real time by 50 Hz and the detected positions were displayed on a PC. Later work by [31] also continues to integrate sensors into clothing, in this case also a jacket which combines information from several inertial measurement units to track the assembly of a car.

2.2.2 From PadNet to SMASH

PadNet [17] has been developed to be seamlessly integrated into clothing while making the distribution of sensors over the whole body possible and an easy task. One significant observation – "most user activities can be characterized by certain motion and state patterns of distinct body parts" has influenced the system design. The architecture consists of multiple subnetworks, each with a master and a central master which links the subnetworks together in a hierarchical manner. This separation of the sensor network into two layers has the objective to increase local processing, minimize data transfer and reduce the computational load and requirements for the central master. This concept is also essential for the approach taken in this thesis. Namely that partial activity recognition of distinct body parts should be enforced and that this will benefit the overall system performance. But unlike PadNet, which uses serial communication, the challenge, of implementing such an approach is enhanced by the problem with wireless communication.

SMASH [13] can be seen as further development of PadNet. It consists of three hierarchically linked layers of processing, cf. Figure 2.1. At the lowest layer "sensing terminals" sample, filter and transmit data to the next level. Within the next level gateways collect data from their attached sensing terminals, fuse the data and extract



(a) layers (b) deployment

Figure 2.1: SMASH[13]

features. The highest level consists of a single unit – "Konnex" – which is the centralized control for the network and collects the data from the gateways for the last processing step which takes the data and performs an online classification with the result sent to the user. In order to evaluate the system, a user study was performed which provided a user-specific accuracy of 85% when detecting different arm positions with 10° difference whereas typically only adjacent classes of arm positions were detected wrong.

2.2.3 Acceleration

One of the most frequently used sensor modalities in activity recognition is acceleration. [3] presents a good overview of research work which use acceleration sensors. In general kinematic motion is used to detect dynamic human activities while gravitation acceleration is used to detect postures [18]. Furthermore posture and movement was discovered to be the most useful context identifier in a study examining daily activities. Location and Object of interaction where the next best context identifiers, which however were 10% less relevant. This is also a reason why this work focus on acceleration sensors although combining multiple different sensor types may yield better activity recognition performance, as presented in the next sections. In the following paragraphs issues and limitations and usage of acceleration sensors are presented.

One problem which still has to be addressed is that acceleration sensors are affected by the gravitational force and therefore the orientation of the sensor has to be known or calculated using the approach presented in [27]. Another lesser problem of acceleration sensors is that they can only detect acceleration but not velocity and that only relative movement against the body can be detected.

In [20] the impact regarding activity accuracy drop due to sensor displacement in activity recognition systems with acceleration sensors is presented and a mitigation strategy is discussed. An acceleration signal composites of the following contributions: orientation due to gravitational force, translations and rotation. Whereas only the later is

affected by sensor displacement and can be measured by a gyroscope. In order to be independent of the location of the sensor on a body part and a possible sensor shift while using it. It is necessary to separate an acceleration reading based on the dominant contribution (orientation, translation or rotation) and successively weight the location independence of the measurement. Given a measurement, where the acceleration vector is similar to the vector of a free falling object, a low measured angular velocity or the acceleration measured by a gyroscope it can be concluded that the acceleration signal is not governed by rotation and hence location independent. Therefore the data measured can be used safely in an activity recognition process.

In [4] a portable acquisition system is presented. It consists of a 450g heavy black box which integrates a digital signal processor and can use Bluetooth or GPRS for communication with a PC, which can acquire signal from up to 16 sensors simultaneously. In an experimental setting the system was used to measure inclination of human body segments using 2-axis acceleration sensors. The sensors where aligned so that the gravitational force was fully measured on one axis and the other axis measured zero in this calibration and reference setting. When the inclination changed – e.g. when the shank is moved to be parallel with the floor the first axis measures zero while the other measures full g-force – a shift from one axis to the other is measured and the angle of the falling gradient can be computed. To test the system in use on a jacket and special limb racks 16 sensors were placed on all body segments except the head. The measurements were fed into a LabView program via Bluetooth which in turn calculated the angles and visualized using a 3D human model. In order to do so a 30 seconds calibration phase is needed to adjust the program to mitigate positioning errors. The update of the human model is performed in real-time mode as long as the person is not moving too fast because then the angle of the body segment can not be calculated correctly.

In [8] a body posture recognition system based on the body area wireless sensor network, WiMoCA is introduced. Using three sensor node, whereas each features a tri-axial accelerometer, placed on thigh, chest and shank the system can separate seven different activities – variants of standing, sitting and lying. For recognition each sensor node computes the average acceleration on a sample window for each axis and computes the orientation of the sensor. The tilt information in respect to gravity and the location of the sensor is then sent towards a base station. The base station combines the tilt information from each sensor node and based on that looks up in a table which tilt configuration belongs to which activity. The activity information is then relayed to an application residing on a PC. Using a collision free MAC implementation which additionally handles synchronisation the system is able to continuously monitor a user with a sampling rate of 30 Hz for 19 hours. This star topology approach is frequently related research.

Jeoung et al. [16] assembled a sensor board with a 3-axis acceleration sensor and a communication facility which implements the IEEE 802.15.4 wireless communication standard for personal area networks. The system is using TinyOS, a small memory footprint and low power event-driven operating system[10]. The system is trimmed to recognize lying, sitting, standing, walking, running and falling. For the latter three activities the system computes the signal vector magnitude and the differential signal magnitude vector, over a 0.5 seconds window size – 50 samples for the used 100Hz sampling rate, and based on a threshold decides the "activeness". Under a certain threshold the user is not performing those dynamic activities and the system decided by a simple threshold algorithm based on mean value of each axis over the same window size as before. In an performance evaluation the system could recognize falling, standing and lying with 100% accuracy, running with 96% and walking with 98% – a total recognition rate of 99.2%.

2.2.4 Acceleration and sound

In order to further increase the accuracy of activity recognition systems other sensor types have been investigated. One of the most prominent additional type used in research is sound.

Lukowicz et al. [24] combined acceleration sensors and microphones in order to recognize activities performed in a wood workshop. For motion detection the PadNet, cf. 2.2.2 was used. As for sound two microphones were used, one located on the chest and the other located on the dominant hand, in order to use the measured intensity difference between those two location. To recognize gestures performed at the workshop HMM were used with an recognition rate of 95% on training data.

To extract features from sound the Fast Fourier Transformation (FFT) was applied to frames of several seconds length with a Linear Discriminant Analysis to reduce the feature space further. Classification was performed using a supervised learning approach where the mean value for each class was calculated and new unseen data was labelled by choosing the next class mean. With this approach a recognition rate of 90% was achieved on training data. In order to distinguish sounds further they were separated into three "intensity" levels. Sounds with high intensity difference between hand and chest microphone – e.g. produced by working with a saw. Sounds with varying intensity difference between the two microphones, the sound starts with the hand of the user close and the hand gets removed while the sound continues – e.g. while using a stationary driller. The third class are background noises and noises which don't involve the users hand. When using only frames where the intensity level was above a given threshold – frames belonging to the first and second class the recognition

rate was increased by an additional 2%. While performing the experiments it was noted that the acceleration based and sound based classification each tend to report the same set of activities when assigning the wrong class label. However these sets of activities were distinguishable from each other and could be used to rule out false positives and increase the overall recognition case in a continuous working mode, from 72% using only acceleration, 2.8% using only sound towards a 84.4% using both modalities.

2.2.5 Acceleration and RFID

Another way to enhance activity recognition is do combine it with RFID technology which leads to very promising results in terms of accuracy.

In [15] three acceleration sensors worn on thigh, waist and wrist. Whereas the sensors on thigh and waist were used to decide between the standing, lying, sitting, walking and running and the sensors on the wrist was used in combination with an RFID reader to decide what was the user doing with a given object. The wrist sensor was used to separate between lateral, vertical and rotational movement.

The RFID reader was integrated in a glove and samples with 3 Hz and has a range of 5 cm so only objects in close proximity were detected. Unlike previous research the body state, hand motion and detected object by the RFID reader was used to distinguish more precisely between a set of daily activities. E.g. only when a vertical motion at the wrist is detected and a cup is in the user hand the detected activity is drinking, unlike e.g. when you rotate the cup and empty its content. Given the combination of the three separated inputs for overall classification which are hierarchically linked the system can detect the right activity out of 18 different activities with an accuracy of 95%.

2.2.6 Acceleration and RSSI

Quwaider et al. [29] introduces a body posture identification system based upon the Mica2Dot wireless measurement system with a two-axis acceleration sensor attached to it. Four nodes where deployed in the sensor network and located on each thigh and upper arm. Using a 20 Hz sampling rate for the acceleration sensors and analysing the FFT amplitude of the average value of the signal on both axis a distinction between walking, running and less dynamic activities was made. However to separate the less dynamic activities – namely sitting and standing – the acceleration signal was found to be not good enough and instead proximity information is used. Each sensor nodes obtains a table of radio signal strength indicators (RSSI) by periodic broadcasting a

"Hello" beacon. The average RSSI tends to be high for a person sitting and rather low for standing posture however through radio blockage by clothing, other objects, sensor and antenna orientation and alignment problems a simple threshold based approach may match in up to 90% however this is highly situation dependent and not applicable for different subjects. To address this issue a HMM has been introduced which uses the current RSSI value and the relation to a peak-to-peak RSSI range as an observation vector and can achieve an accuracy up to 94% (minimum of 86%). Using an expectation-maximization algorithm they can iteratively adjust the observation probability matrix to fit a subject for a given and initial "false" observation probability matrix, this behaviour increases the accuracy of the recognition further after some time.

2.2.7 Other sensor types

Van Laerhoven et al. [22] experimented with two different types of sensors. On the one hand they used a set of 20 accelerometers assembled together on a bar, called the spine. On the other hand they used nine wireless sensor nodes with ball or tilt-switches installed on it, called the porcupine. Both sensor systems were using serial communication to collect the data from all sensors or sensor nodes which in turn were forwarded wireless to a base station. While performing the study Van Learhoven et al. first tried to use porcupine's sensor node wireless communication facilities for data collecting however through a lot of packet collisions and not reaching the required throughput they switched to wired communication. This problem was also experienced in this thesis an will be revisited later on. The activities performed during the experiments were: lying, kneeling, sitting, standing, walking, running, ascending/descending stairs, bicycling and jumping while both sensor platforms were attached to the test subject's legs. The overall classification accuracy for spine was 93.88% and 65.24% for porcupine. Interestingly was the finding that when using boosting with porcupine – classifying first locally on each sensor node and classify one with all local classification results – they reached a accuracy of 62%. This emphasizes that a distributed approach will yield promising results and be as good as a centralized approach. They also underline that neglecting or decreasing the accuracy of sensor node's data by classifying locally will still be valuable when using a good algorithm for fusing the data back together.

Another approach for context or activity recognition is presented by [5]. They created a logging application on a mobile phone which is carried by a user to record daily live activities. For activity recognition it uses the mobile phone's cell id and information about Bluetooth devices in the environment (MAC Address, major and minor device number). The user labels the data with activity information and a temporal relation model is deduced. This model is later used for predicting the activity the subject is

performing. The harmonic mean of precision and recall of the activity recognition was up to 96%. This shows once more that quite simple and at first glance "irrelevant" features can be good context identifiers.

Concluded activity recognition have evolved from single sensor systems into multi-sensor systems, whereas not only multiple sensors of the same type were used but also multiple sensor types have been integrated. The tendency is towards multiple-heterogeneous-sensor system. However these systems imply a significant challenge:

The management of such systems.

The more sensor nodes used the bigger the following problems are in such traditional multi-sensor fusion system with static configuration:

communication when adding a sensor node, the communication pattern needs to be adjusted, the software needs to be recompiled and deployed to all sensor nodes

change of the recognition goal when changing the recognition goal, the software needs to be changed and deployed to all sensor nodes

replacement when replacing a sensor node the software needs to be recompiled and deployed to all sensor nodes which interact with this sensor node

breakdown breakdown of a single sensor will result in a breakdown of the total sensor system without a recovery possibiliy

Therefore it is essential to shift towards a more dynamic approach. Where sensor nodes are more or less independent from each other. They can discover their neighbours capabilities and self-organize themselves to achieve a recogniton goal.

2.3 Distributed systems and self-organization

Collier et al. [6] investigate how to define self-organization in sensor networks and what challenges are. They define that a system is self-organizing if a collection of those system components work together and coordinate to form a collective which adopts and achieves a goal more efficient than a single component could. The following enumerates the definition into a set of features:

1. *The system is composed of units which may individually respond to local stimuli.*

2. *The units act together to achieve a division of labour.*

3. *The overall system adapts to achieve a goal or goals more efficiently.*

[6]

Despite the complex nature of the presented features wireless sensor network face further challenges like limited power, limited radio range, high density and an highly dynamic environment. Especially the coordination of communication is a problem because two sensor nodes may not access the MAC layer simultaneously but have to schedule their access and through the limited communication range they have to schedule routing of messages within the network. So in general wireless sensor networks will try to optimize data throughput, low latency between any sensor node within the network. However this may not hold as some sensor nodes may create more data or have more important data then others. Therefore are privileged and routed faster towards a data sink. Less mission critical data is delayed.

[26] proclaims that self-organization is an emergent feature of all distributed system. A system may reach a certain state where it is stable although fluctuation may occur. Changes in the environment, which will shift the system into a less stable state, will force the system to adopt in order to go back to a stable state. This adaptability will make the system able to deal with failure of components it is composed of. Self-organization may organize over time, space or both whereas they tend to be hierarchically structured and the hierarchical layers are spatial similar. This phenomena of self-organization can be found amongst others in – e.g. where homogeneous cells of embryos evolve into heterogeneous cells with specialized function, social science – e.g. where swarm of birds can manoeuvre as flocks, or economics – where supply and demand on the market change the behaviour of consumers and producers alike.

When applying this self-organization behaviour to wireless sensor networks [26] they look into the ability of the system to accomplish the following: (i) sharing physical resources, (ii) structuring, (iii) adapted behaviour, (iv) manage collective resources and (v) ensure survival.

(i) Sensor nodes can share physical resources and capabilities to other sensor nodes within the network e.g. nodes with bigger battery supply may announce their superiority and take over message relaying from devices with low power in order to maximize overall life time of all nodes. (ii) As for structuring refer to the later example of AnyBody [32]. (iii) A sensor system might adopt its behaviour in form of reconfiguring software modules, whereas a sensor node senses the need of his environment for ac-

complishing a special task and the sensor node will change his software configuration – e.g. load and unload of modules – to fulfil this task. (iv) Within a sensor network all sensors commonly share the communication channel with each other. Given a sufficient density in such a sensor network its not necessary to power on all sensor nodes to deliver information. On the contrary powering on all nodes will likely create interference and diminish the networks throughput. Therefore a sleep, wake and forward algorithm will adjust this behaviour if two neighbouring nodes fail to be able to communicate with each other without his help. (v) A sensor system should be able to deal with sensor node failure. This is important for sensor networks with a cluster structure. Cluster heads may exchange information about their fitness or existence and adjust if one of the cluster heads fails to perform proper.

AnyBody [32] is a protocol for self-organization which focus on structuring the sensor network into cluster which efficiently in terms of speed and power route messages from data sources to a sink. First each node discovers his one-hop neighbours – neighbours it can communicate to directly – by broadcasting an appropriate message and receiving responses of the neighbours. Secondly a node will discover the neighbours two-hops away by relaying the previously broadcast and received message from other sensor nodes. After complete knowledge of its neighbours a sensor node will calculate the density (ratio between number of links and number of nodes within this 2 hop neighbourhood) and broadcast it. Afterwards each node, except the node with the highest density in its neighbourhood, will send a list of 1 hop neighbours towards the neighbour with the highest density with a join request. With this join request the node with the highest density is now the clusterhead of its neighbourhood and all messages will be routed to/from it. Each cluster, respectively clusterhead, will poll through its cluster member nodes and collect information about which node has neighbouring node outside the own cluster. These nodes will be inter-cluster gateways. They will relay messages from other cluster/clusterheads to its own clusterhead. In case of a single sink, where information has to be sent to, its necessary to set up routing towards it. This is done by the sink sending out a message to its clusterhead which in turn increases a counter within the message and send it towards its neighbours. By storing where they received and sent the message to and the appropriate counter value the clusterhead can infer which is the next clusterhead. A message has to be sent to, in order to end up at – be relayed to – the sink.

In [28] object networks are described as a headstone for self-organization in order to perform distributed activity recognition. The object network is an overlay network of interconnected devices around a user. This network is hierarchically organized and contains a leader object. Each object takes care of sensing and discovering other objects and selecting a leader object, which should be the most powerful – in terms of e.g. battery supply, processing capabilities or storage – device in the network. Each object may have a role in this network like acquiring sensor data or process data of other

objects. A role may be dependent on the work of other roles. These dependencies automatically create context zones where one or more objects provide information to one object which needs the information provided by those objects to fulfil its role. This can be used to abstract the information, provided e.g. by low level sensors, which is processed and combined into a simpler representation – e.g. by classification. Context zones can be hierarchically linked however at the top of the overall network will always be the leader object. Which will use the data generated by the object network and a map of relations between activities (activity map) to infer what activity the user is performing. E.g. activity use cup and use coffee machine will let the system infer that the user is having a cup of coffee. This examples underlines once more the need to classify locally and feed preliminary classified data into the overall activity recognition process.

With the principles of self-organization it is possible to overcome the problems of multi-sensor fusion systems with static configuration. Self-organization can be used to dynamically configure a sensor network. The examples presented outline how a sensor system can be structured to fulfil a goal, e.g. like relaying messages efficiently from sink to source. Therefore in order to create a sensor system, which uses dynamic configuration and is able to change its goal at runtime, it is essential to incorporate self-organization into the system's design.

2.4 Distributed systems and service discovery

Service Discovery is proposed as a solution to the problems with high dynamics and heterogeneous environment in pervasive computing by [33] or [11]. Service discovery is needed in order to know what services are provided by an environment. This obtained information can later be used to combine several services to create an activity recognition service/system. By e.g. combing the sensor data from different sensor services. Various service discovery approaches exists, two prominent representatives are Sun Microsystem's Jini and Microsoft's UPnP. For better understanding the service discovery approaches its essentially to understand the underlying concept in service discovery approach, service registration and service infrastructure.

Two service discovery approaches exist: announcement-based and query-based. In the first case every participant in the network environment listens to service announcements by other participants which occur periodically. In the latter case a participant in a network requests service information from his environment.

As for service registration and infrastructure two different modalities exist as well: directory or non-directory based. In the first case, a centralized authority holds information of services available in the surroundings. If a new service gets available in the network it will register itself with the authority and can then be accessed by other participants. In an environment based on non-directory based service registration and infrastructure every participants holds information of services available. A new service will announce its availability and the other participants will store this information for later usage.

Those service discovery methods work quite well in a known environment. However when considering advances in ad-hoc sensor networks, which are more and more common in pervasive computing and have proven their feasibility, applying these service discovery approaches leads to certain issues. Considering pairing announcement based discovery of services and non-directory based infrastructure which would be ad-hoc the best selection for an unknown environment. The main issue here is the flooding of the network with announcement on a regular basis, which is not feasible for a network with wireless communication because this is the main power drain for such sensor systems. Furthermore the information of all available services has to be stored on every participant which may lead to memory shortage as the main memory is still limited on such devices. If using query based discovery with a non-directory based infrastructure it may take a lot of requests to find a suitable service, again draining the battery. On the other hand when using queries with in a directory based infrastructure, the central authority has to be located in the first place if it is in communication range at all. If this isn't the case although other services are available in close proximity they can't be used. Finally service composition may only occur at the requester site which leads to the fact that the data produced by all services has to be transported through the network, which will reduce bandwidth throughput and therefore increase the energy demand within the network.

Therefore, service discovery may be a valid alternative to self-organization for activity recognition in known environments, however, in unknown environments self-organization will be more suited as it's more goal oriented and will structure the sensor system to be more efficient.

Chapter 3

DarSens

In this chapter the approach for opportunistic activity recognition, the solution for the problems of traditional multi-sensor fusion systems, is extended and explained. This approach includes access to data, processing capabilities of a sensor node in a reusable way. Furthermore classification and feature extraction is performed on-site and sensor nodes can interact with each other without a-priori knowledge of each other. This approach is extended and implemented in a framework called **D**arSens .

3.1 Approach

As deferred by the related work, the challenges of wireless sensor network – communication pattern, change of recognition task, replacement and breakdown issues – are only solveable with a more dynamic approach, namely opportunisitic activity recognition, as also proposed by the European research project **O**PPORTUNITY.

The purpose of opportunistic activity recognition is that just those sensor that can provide relevant information cooperate to achieve the goal of recognizing an activity or multiple activities. These activities are combined and called a recognition goal, e.g. locomotion activities like walking, standing and running).

This recognition goal is an abstract concept, which has to be transferred into a coordinated sensing mission to be processable for a sensor system. This sensing mission specifies how and which sensors have to cooperate in order to satisfy the recognition goal. In order to translate a recognition goal into a sensing mission it may be necessary to create a database where each recognition goal has at least one sensing mission stored, in order to be able to translate automatically.

3.2 Concept

The concept of **D**arSens is easily explained with a application scenario. Consider a mobile-phone being interested in the situation the wearer is in. Additionally the wearer has a set of sensor nodes integrated into his garment. Now in order to adapt its behaviour to the user a mobile phone will be interested in certain activities of the user. E.g. if the user is currently running to catch the train it will reject a call, in order to not disturb the user, and notify the caller accordingly. To access the activity information the mobile phone will request the desired information using wireless communication. As the user is equipped with a sensor network, it will receive this request and will then self-organize to provide the mobile phone with the desired information.

To be more precise the mobile phone will transform its recognition goal – the activities it is interested in – into a sensing mission and broadcast this sensing mission. The sensor nodes will receive this sensing mission and self-organize themselves to fulfil the requirement of the sensing mission. If this requirements can be satisfied a confirmation of this status, namely a satisfied sensing mission, will be delivered to the requester which can use this confirmation to start the recognition process within the sensor network. This behaviour is illustrated in Figure 3.1.

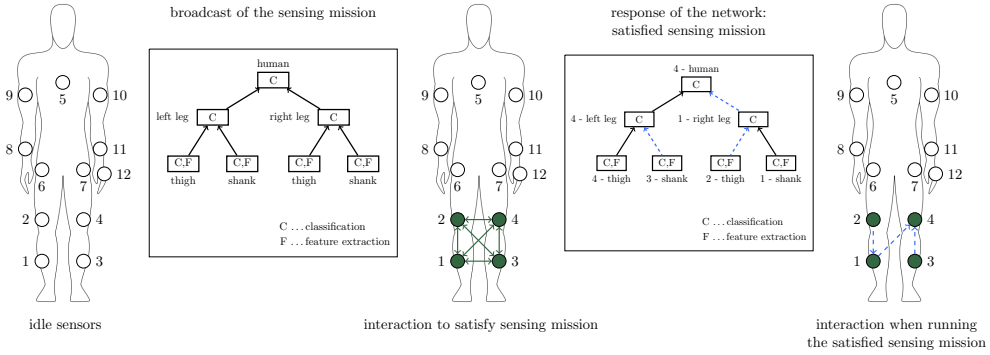


Figure 3.1: Concept

3.2.1 DarSens and Opportunity

DarSens shares the conceptionally ideas of **O**PPORTUNITY,cf. Table 3.1 and copies its nomenclature. The focus in **D**arSens lies on transforming the sensor network into a coordinated sensor ensemble by the use of a sensing mission and the communication behaviour. The recognition goal and its transformation into a sensing mission is omitted within **D**arSens. An approach for this behaviour would be a database lookup where for each activity or set of activities a corresponding sensing mission is returned and used to self-organize the sensor network.

Since the recognition goal consists of activities, whereas each activity can be detected using sensors on different places on the body or types of sensors, there are many possibilities for the layout of a corresponding sensing mission. E.g. walking can be detected on many body positions [19], like left thigh, right thigh, left shank, right shank or combinations of those positions. Therefore a recognition goal 'walking' could have 15 different sensing missions stored. Any of those sensing missions may be suitable and could be chosen to recognize the activity. Now if the environment doesn't have a sensor on a body position specified in the sensing mission another sensing mission can be chosen at runtime. This is a major benefit of this recognition goal initiated multi-sensor activity recognition approach as opposed in traditional multi-sensor fusion strategies where it is necessary that all sensors need to exist. Especially within a wireless sensor network this approach is not fit to mitigate the cessation of wireless sensor nodes, which may occur because of low running batteries or system faults.

DarSens focuses and implements the concepts of the sensing mission and the communication in a four step approach. First a requester – someone/something in need of activity information – broadcasts a sensing mission. The sensing mission will be used to self-organize the network and if successful a satisfied sensing mission – the corresponding notification of the self-organization process – will be sent to the requester. This

requester can in turn use the satisfied sensing mission to start the activity recognition process, cf. Figure 3.1.

OPPORTUNITY	**D**arSens
• recognition goal	• broadcast sensing mission
• **sensing mission**	• sensors receive sensing mission and self-organize
• **communication**	• if successful a corresponding notification will be delivered
	• approval of the requester of the requester to start the activity recognition process

Table 3.1: Comparison of **D**arSens and **O**PPORTUNITY

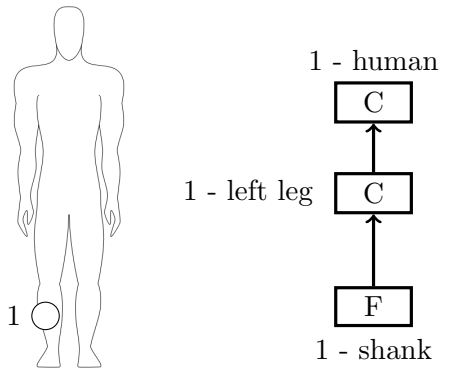
3.3 Sensing mission and satisfied sensing mission

In **D**arSens sensing missions and satisfied sensing missions are represented with the help of a self-developed containment hierarchy. It defines which sensor nodes of a sensor network take part in a sensing mission.

3.3.1 Containment hierarchy

The layout of the containment hierarchy is motivated by two ideas also found in related work, cf. Chapter 2. (i) That most activities can be characterized by characterizing sub-activities or state/motion changes in certain body parts [17] and (ii) that neglecting accuracy through local classification on sensor nodes and fusing the data back to together will result in an overall good system performance (activity recognition, accuracy, battery life-time, . . .)[22].

The hierarchy is based upon the anatomy of the human body. The root element of every containment hierarchy is always referred to *human*. The following child nodes are named *head, left arm, right arm, left leg, right leg and torso* and resemble the extremities of a human body. These body parts can be further segmented e.g. *left leg* the corresponding children are: *foot, shank, thigh*. This approach can be used to create a hierarchy that represents the position of a sensor node on the human body. E.g. for a sensor node placed on the shank of the left leg the containment hierarchy would consist of the *human, left leg and shank*, , cf. Figure 3.2



(a) sensor node (b) containment hierarchy

Figure 3.2: Sensor node placement and corresponding containment hierarchy

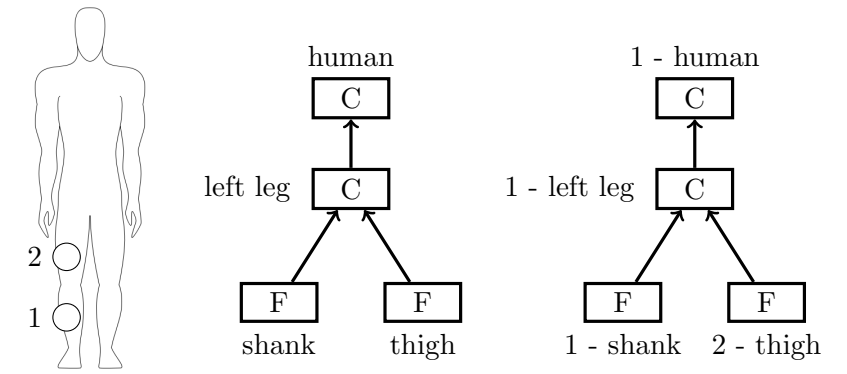
Now in **D**arSens every sensor node has to know its own position either pre-defined or by using a localization approach such as presented in [19]. This position information is stored and used for self-organization, cf. section 3.4.

Furthermore when the containment hierarchy is used to represent a sensing mission or satisfied sensing mission, it identifies the positions of sensor nodes needed for activity recognition, cf. Figure 3.3 – defining which sensor provides relevant information for the recognition goal. Additionally through the structural overlap in the containment hierarchy in a sensing mission, it also defines the recognition chain, i.e. the information flow from input devices – acceleration sensors on sensor nodes – towards an output device – a mobile phone with a corresponding interface to the sensor network.

A lot of activities can be identified by movement in respective body parts – E.g. most locomotion activities like running or walking can be identified using sensor nodes attached to the lower body parts (legs or hips) [2]. Therefore a sensing mission should use only the sensor nodes which are placed on body parts involved in performance of a given activity. Hence at the leaf nodes of the sensing mission raw sensor data will be extracted and fed into the recognition chain. This raw data can be further processed either by feature extraction, classification or both. This is specified by the 'F' for feature extraction or 'C' for classification, cf. Figure 3.2. Denote that classification and feature extraction can be performed at every node in the hierarchy however it is recommended to perform feature extraction and classification as soon as possible to reduce the amount of data that has to be transferred to next level. When performing classification over the data gained by sensor nodes on body parts, the goal should be to recognize not solely the overall activities but the distinctive sub-activities for a given body part. E.g. when using a sensing mission to detect jumping jacks, the sensor data from the legs should be used to classify the posture of the legs at the left/right leg node and combine it with the respective left/right arm posture in the human node.

Denote that a sensor node which executes the tasks – sensor data acquisition, feature extraction and/or classification – at a leaf node also executes tasks of the ancestors in the containment hierarchy. But as the leaf nodes in a sensing mission share their ancestors only one sensor node will be performing those tasks. The selection of the sensor node to execute those tasks is part of the self-organization phase and will be documented in the satisfied sensing mission.

A satisfied sensing mission is an extension of the sensing mission. It carries the same information as the sensing mission however every node within the containment hierarchy has a MAC address assigned, cf. Figure 3.3. With this assignments it its clear where the tasks of the nodes in the containment hierarchy are executed. The satisfied sensing mission is needed because multiple sensor configurations may be applicable to satisfy a given sensing mission and hence it is needed for the requester to select a set of sensors, which will then execute the tasks of the sensing mission. E.g. a sensing mission consisting solely of the human node would be satisfiable by many sensors. Now, if all of those would start sending acquired activity information, the requester would get the same information multiple times, which would in turn drain the battery of all sensor nodes and is completely unnecessary. Therefore when the sensor network sends a satisfied sensing mission to a requester, it will selected a satisfied sensing mission and the selected ensemble of sensors will start the recognition process.



(a) sensor nodes (b) sensing mission (c) exemplary satisfied sensing mission

Figure 3.3: Sensor nodes placement and a corresponding sensing mission/satisfied sensing mission

3.4 Self-organization

When performing opportunistic activity recognition it is certain that the environment is unknown, meaning there is no a-priori knowledge of the sensor nodes available in the

environment. There are but two solutions to the problem: (i) Discover the environment by inquiring information about all sensors and their capabilities and orchestrating these sensors into an ensemble for activity recognition. (ii) Be opportunistic and assume that the sensors needed for the recognition task are available, send out the sensing mission and let the sensor nodes self-organize to satisfy the sensing mission.

(i) is the traditional approach for wireless sensor networks, however, the effort for inquiring what sensors are available is quite high, especially as every sensor needs to send its capabilities to the inquirer – potentially relayed over many hops – which needs to be processed even if they are not needed for the recognition task. The opportunistic approach (ii) is better as, only necessary information from the network to the inquirer is sent, leading to potentially less traffic. Furthermore for environments with changing conditions, precisely the leaving and joining of sensor nodes, this approach is much more suited, as in the first approach, the inquiry had to be done every time the recognition goal changes, because the possibility is high that the environment has changed. The keystone for this opportunistic approach is that the sensor nodes can self-organize themselves into a sensor ensemble, able to recognize the activities specified. The following paragraphs explain how this self-organization is implemented in **D**arSens.

The self-organization process is constrained by the broadcasted sensing mission, as the sensing mission defines from which body parts sensor data needs to be gathered. The sensor ensemble can only successfully self-organize, if data from all body parts specified in the sensing mission can be gathered, unlike static configurations where the exact sensors – e.g. on MAC address basis – are specified. This has the advantage that the sensor can easily be replaced without the need to change the configuration of the whole sensor ensemble.

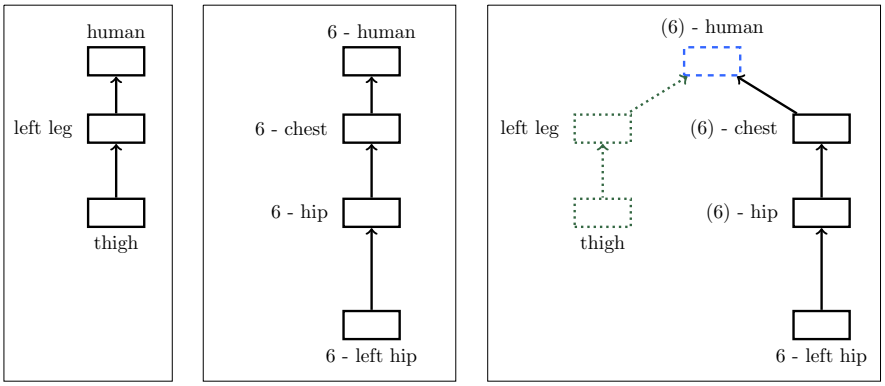
Since a sensor node knows his own position on the body and therefore the respective containment hierarchy he can compare this information with the containment hierarchy of a received sensing mission. This comparison will result in the creation of a containment hierarchy consisting of the nodes in which the sensing mission and the local containment hierarchy differ. Based upon this comparison there are several ways to processed:

- no participation
- participation
  - full overlap – difference: 0 nodes
  - almost full overlap – difference: 1 node

– partial overlap – difference: > 1 nodes

3.4.1 Participation

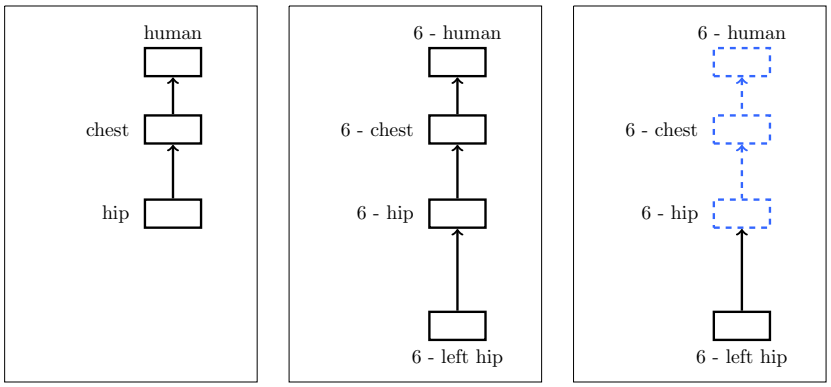
A sensor node only participates in a sensing mission, if and only if it is useful to that particular sensing mission. Usefulness in this context means that there has to be an overlap between the sensing mission and the containment hierarchy insofar as a leaf node of the sensing mission and the local containment hierarchy has to overlap. As already stated previously the goal of a sensing mission is to recognize a set of activities and the sensing mission should be composed of the containment hierarchies of the body parts involved in this activities. Therefore if you want to recognize activities which involve e.g. the thigh of the left leg you are not interested in sensor data of the left hip, hence a sensor on the left hip will not participate in the respective sensing mission, cf. Figure 3.4.



(a) sensing mission (b) local containment hierarchy (c) difference tree

Figure 3.4: Unsuccessful overlap between sensing mission and local containment hierarchy

Denote that the depth of the local containment hierarchies can be higher than the one of the sensing mission, cf. Figure 3.5. Participation is still successful because the sensor data extracted is part of the specified body part in the sensing mission. Vice-versa this would not be the case because the sensor data extracted would not be on the position specified within the sensing mission. Therefore it is either necessary to specify the position where to extract data in depth or use training data for the recognition chain on several body positions. E.g. if the goal is to recognize locomotion activity (running, walking, ...), its necessary to specify that the sensing mission uses data from the left leg thigh – for instance – or use training data for the classification from the whole body and solely specify human in the sensing mission.



(a) sensing mission (b) local containment hierarchy (c) difference tree

Figure 3.5: Successful overlap between sensing mission and local containment hierarchy

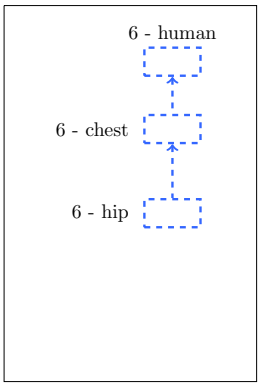


Figure 3.6: Satisfied sensing mission

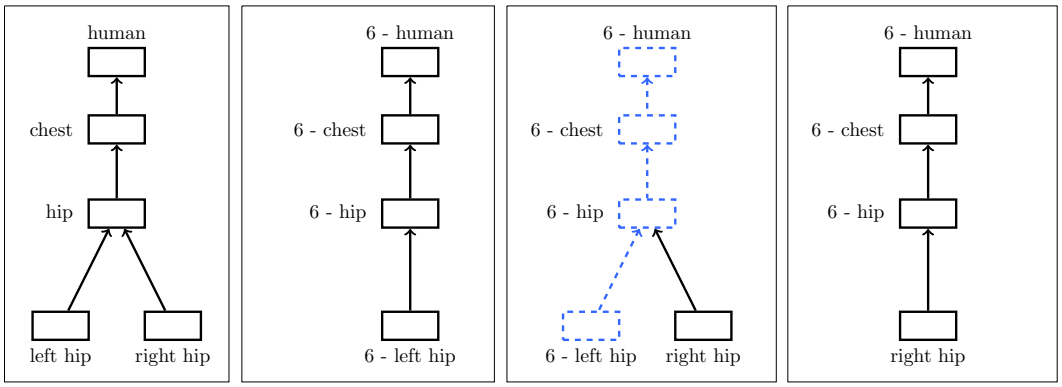
3.4.2 Fulfilment of a sensing mission

If and only if a sensor node participates in a sensing mission it will create a satisfied sensing mission by inserting its own MAC address at all the overlapping nodes within the sensing mission with no – previously assigned – MAC address assigned. If all nodes now have a MAC Address assigned like in Figure 3.5(c) the resulting satisfied sensing mission will look like Figure 3.6 and will be sent to the requester. If this is not the case a sub-self-organization process will be started.

3.4.3 Sub-self-organization

For the sub-self-organization process the difference tree between the sensing mission and the local containment hierarchy will be used as a starting point. The parts of

the tree where all the nodes overlap – where after the creation of the satisfied sensing mission all nodes have a MAC address assigned – will be pruned and the rest of the tree will be packed into a single sub-sensing mission and broadcast to the sensor network. There are two cases where this is the case either almost full overlap (difference: one node) or partial overlap difference: more than one node) between the local containment hierarchy and the sensing missions containment hierarchy.



(a) sensing mission (b) local containment hierarchy on sensor 6 (c) almost satisfied sensing mission (d) sub-sensing mission

(e) local containment hierarchy on sensor 7 (f) satisfied (sub) sensing mission (g) completed satisfied sensing mission

Figure 3.7: Self-organization process with one node difference

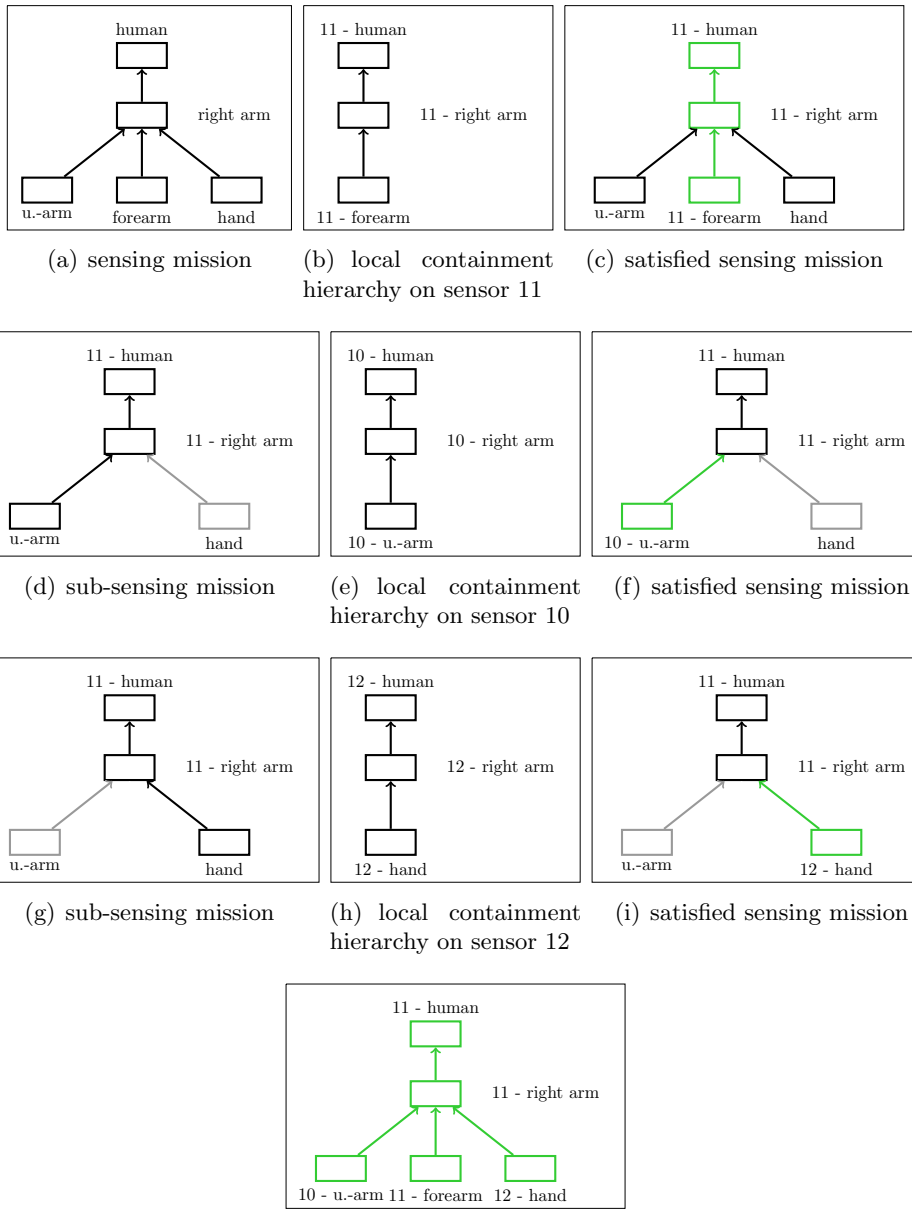
In the first case the sensor node will only need the help of one other physical node to fulfil the sensing mission. E.g. in Figure 3.7 the satisfied sensing mission, 3.7(c), misses one node to satisfy the sensing mission. Therefore it will create a sub-sensing mission, 3.7(d), out of the almost satisfied sensing mission and broadcast it. The sensor node with the MAC address 7 will fulfil the request and respond with the created satisfied (sub) sensing mission, 3.7(f), which will in turn be integrated into the almost satisfied sensing mission of the sensor node with MAC address 6. Therefore the requested sensing mission is now fully satisfied and the sensor node will send the satisfied sensing mission, 3.7(f), to the requester of sensing mission.

In the later case there will be a master selection process, where the difference in nodes between the local containtment hierarchy and the sensing mission is more than one. E.g. as shown in Figure 3.8, all sensor nodes receive the sensing mission and either participate or not. In our case the sensor node with the MAC address 10, 11 and 12 will participate. Everyone of those sensors will calculate the amount of nodes in which they differ – two in our case – and broadcast this information with an additional random number in case of equality. All sensor nodes will wait for an amount of time to receive this information and if their own difference number or random number is higher stop their own local master selection. After the time is up, or it received a message from all applicable siblings of the sensing mission (u.arm, forearm and hand), the node with the lowest difference number and random number elects itself as master and creates a incomplete satisfied sensing mission, cf. Figure 3.8(c). The sensor node will of course create a sub-sensing mission for the missing part. The process is illustrated in Figure 3.8(d) - 3.8(f) for sensor node 10 and 3.8(g) - 3.8(i) respectively. In early versions of **D**arSens the sensor node which got elected as master splitted the sub-sensing mission into respective sub-trees, in our case this would have been sub-sensing missions like the local containment hierarchy of sensor nodes 10 and 12. However for performance and to save bandwidth reasons this splitting will occur at the receiver side and only the relevant part will be evaluated for the self-organization process as outlined in 3.8(d) and 3.8(g) for node 10 and 12.

The master selection process will be performed multiple times per node within the hierarchy until the sensing mission is satisfied and the completed satisfied sensing mission can be sent to the requester. Denote that subsequent master selections will take place on different nodes and not the node which “won” the first master selection.

Because of this behaviour its possible to use this mechanism in an environment where not all nodes can communicate with each other or where this behaviour is discouraged in order to save power. E.g. consider that the sensor node 12 (hand) is to far away from the original requester of the sensing mission of Figure 3.8. The original request will be received by sensor 11 and 10 and they will start the master selection process. After the master selection process either one will be able to create a sub request for the missing nodes of the satisfied sensing mission and they can communicate with the, for the original requester unreachable, node 12 and fulfil sensing mission.

Denote that a (sub-) sensing mission can be satisfied by multiple sensor ensembles using the self-organization mechanism. Therefore the requester of a sensing mission will currently use the first satisfied sensing mission it receives to start the network setup or integrate them into a existing satisfied sensing mission. Subsequent received satisfied sensing mission will be discarded. It should also be noted that all requests are done using broadcasts and responses are directly communicated between two pairs.



(a) sensing mission

(b) local containment hierarchy on sensor 11

(c) satisfied sensing mission

(d) sub-sensing mission

(e) local containment hierarchy on sensor 10

(f) satisfied sensing mission

(g) sub-sensing mission

(h) local containment hierarchy on sensor 12

(i) satisfied sensing mission

(j) completed satisfied sensing mission

Figure 3.8: Self-organization process with master selection

Instead of using a "first come, first serve" approach the requester could also choose on a heuristic, like which sensing mission is best for routing data energy efficiently towards the requester. However this is not implemented yet but could be in future work.

3.5 Network-setup

After the self-organization within the network, the requester of a sensing mission may receive one or more satisfied sensing missions and choose the first one to start the network setup. If more than one satisfied sensing missions are returned, it would also be possible to select one of those sensor ensembles based on meta-information e.g. power left on those sensor node and selecting the sensor ensemble with the most power left, however this is future work and currently not implemented.

The requester will notify the sensor node which sent him the satisfied sensing mission. This sensor node which is also the sensor node assigned to the root node of the containment hierarchy of the sensing mission, will hence start the network setup, cf. Figure 3.9. Upon arrival of this notification the sensor node will subsequently inspect the satisfied sensing mission which is enclosed in the notification and examine the root node. E.g. classification task in Figure 3.9 at the root node needs additional data to work properly – in this case the classification model. However through the limited size of packets in the IEEE 802.15.4 specification this data has to be fetched from the requester and cannot be enclosed in the notification sent from the requester. The sensor node assigned to the root node will request the classification model (dark/light green line in figure) from the requester and afterwards relay the notification to its children so they can in turn start the network setup. The information which features to extract is also fetched from the requester in form of a matrix which enables data acquisition and feature calculation per axis. Therefore it is also recommended that feature extraction is only used at the leaf nodes otherwise the raw data has to be delivered through wireless communication channels which will in turn drain the battery of the sensor nodes quickly.

The network setup process is performed sequentially per level of the satisfied sensing mission and in parallel per children. E.g. in Figure 3.9 after the root node finished setup it will notify both children which are doing their setup in parallel. Denote that by using the satisfied sensing mission created by the self-organization process, the sensor ensemble will perform self-management insofar as it will open only the necessary communication channels on each node of the containment hierarchy, as wells as perform classification, sensor data acquisition and feature extraction if needed and deliver the data gathered or processed to the upper level in the hierarchy, where it is processed further.

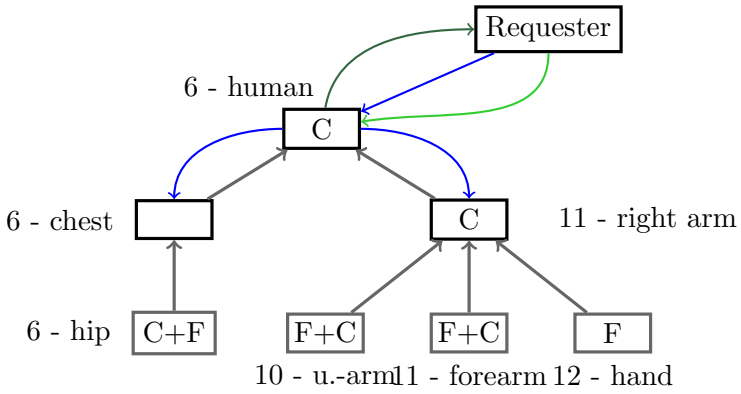


Figure 3.9: Network setup

As the network setup is top down, it has the advantage that in a sensor ensemble which is distributed over many hops, the communication routes from the root node to the leaf nodes are set-upped with the top down traversal of the satisfied sensing mission, and this routes can then be used to fetch additional information relevant for the sensing mission – e.g. classification models – from the inquirer.

3.5.1 Recognition chain

Once the network setup is completed the activity recognition can start. The recognition chain defines the way the data flows from input to output and what tasks have to be done with that data. In **D**arSens the data flow is bottom up. Therefore our input are the physical sensors of the sensor nodes assigned to the leaf nodes. Based upon the task specified in a given leaf node the data may be fed into the feature extraction chain afterwards in a local classification process and then sent to sensor node assigned to the ancestors node for further processing until the data reaches the root node and the final recognition result will be sent towards the requester. Denote that it is also possible to omit any processing at a node, e.g cf. the chest node in Figure 3.9.

3.6 Self-adaptation

A self-organizing system needs to adapt to changes in the environment [6]. In the case of a sensor system, changes in the environment mean the leaving and joining of sensor nodes, which can happen quite easily in a wireless sensor network. This may happen due to segmentation faults, drained batteries or any other kind of errors. However it is essential that in an activity recognition scenario the data flow continues because

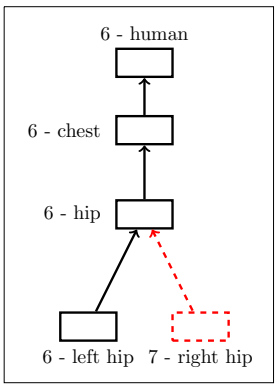


Figure 3.10: Detected error in a sensing mission

only the continuous availability of this data enables context aware applications to be successful.

Therefore **D**arSens implements a self-adaptation behaviour, which mitigates such change, the disappearance of sensor nodes, by reusing the self-organization technique. When a sensing mission is deployed all sensor nodes execute the tasks they got assigned to. Now if a sensor node leaves the network, e.g. because his battery is drained, the data generated – by the tasks previously executed – in that process is missing as input for the tasks at the respective ancestor node. E.g cf. Figure 3.10 where the tasks at the node hip needs input from node right and left hip. Whereas the tasks of left hip and hip are both executed on the sensor node 6 and the tasks of right hip are executed on the sensor node 7.

Now if a failure occurs at sensor node 7, cf. Figure 3.10, the data generated by it is missing as input at the hip node. There are three strategies to recover from this failure:

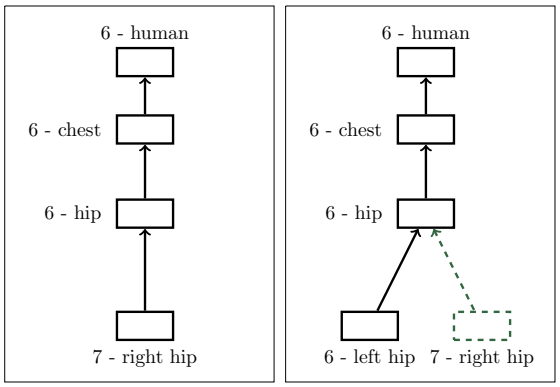
1. **notification** - notify its children to continue delivering information

2. **recreation** - create a sub-sensing mission for the failed part of the sensing mission

3. **termination** - notify its ancestors to terminate the satisfied sensing mission

3.6.1 Notification

The first strategy to recover from a sensor node failure, is that the sensor node which detected the failure notifies the sensor, which failed to deliver data. The behaviour to



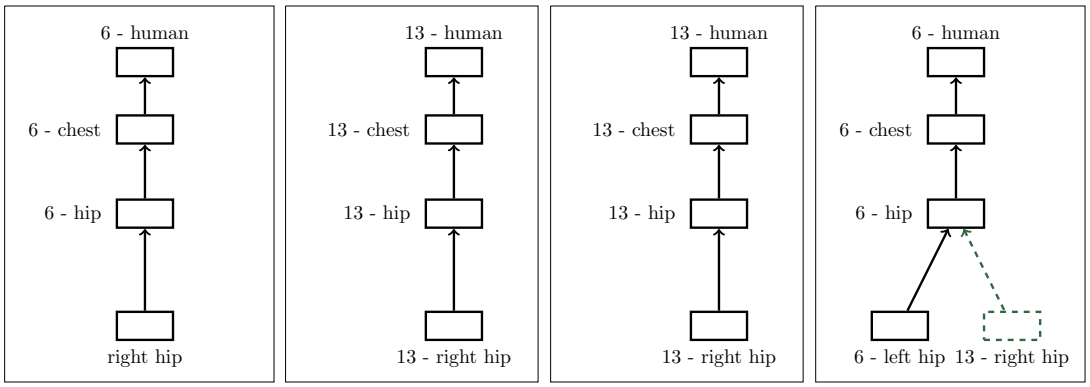
(a) notify (b) successful notification

Figure 3.11: Notification

do so is as follows: The sensor node which detected the failure will take the satisfied sensing mission, which it received to start the activity recognition process, and extract the part of the hierarchy where the failed node is the root node of and create a notification, cf. Figure 3.11, which is in fact the same type of notification, that is used to start the activity recognition process by the requester of sensing mission after the self-organization phase. If the sensor node can successfully notify the failed sensor, the activity recognition process will again be started on the later node and the data will again be gathered an processed.

3.6.2 Recreation

However, if the notification is not successful after a few reties, the sensor node will assume that the other sensor node is permanently lost and try to replace it by using another sensor node with the same capabilities. It will – just like in the notification case – again use the satisfied sensing mission which started the activity recognition process, extract the part of the containment hierarchy where the failed node is the root node of and remove the MAC addresses assigned to all the nodes including and below the node which failed to deliver data and in turn create a respective sub sensing mission, which will then be broadcast to the network. Now like in Figure 3.12 a sensor node, in our case the sensor node with MAC address 13, will receive this sub sensing mission and satisfy it. It will respond to the sensor node 6, which will in turn integrate the satisfied sub sensing mission into the global sensing mission and hence replace the failed part with it. Then it will notify the sensor node with MAC address 13, which starts the network setup and again the data will be gathered and processed.



(a) sub-SM (b) local tree on sensor 13 (c) successful overlap (d) successful integration

Figure 3.12: Recreation

3.6.3 Termination

If both strategies, notification and recreation, to recover from an error fail the sensor node, which detected the failure, will report to his ancestors and remaining children to terminate the sensing mission. Those in turn will relay this termination request to their children and ancestors until this notification reaches the leaf nodes on the one hand and the requester of the sensing mission on the other hand. The requester will have to deal with the terminated sensing mission either by re-notification, requesting the sensing mission anew or by choosing another sensing mission which provides the same activity information as the previous one with another set of nodes in the containment hierarchy. E.g. if the requester was running a sensing mission on the left leg which detects locomotion activity and part of the sensors run out of battery running an equivalent sensing mission on the right leg would be a good solution.

Denote that in the examples described, only single node failure was mentioned. However the system can also deal with the failure of multiple nodes at once because for the node that detected the failure only the failure at the root node of that sub tree is visible and hence even if the sensor nodes of the whole sub tree aren't available any more for the node who detected the failure only the failure at the root node is visible the rest is hidden because of the structure of the containment hierarchy.

One case, which has been omitted so far, is the behaviour of children of a node in the containment hierarchy. If the sensor node, which failed, isn't reachable any more they will assume that the data they gather and process is not needed any longer and terminate the activity recognition process and notify their children to stop the activity recognition process as well. This behaviour is also applied if the requester of a sensing

mission fails or disappears. The root node will notify its children and they in turn their children and the data flow will stop.

3.7 Conclusion

This chapter presented **D**arSens , an approach for sensing goal initiated multi-sensor activity recognition, which yields the benefit that the sensing goal can be changed at runtime unlike in traditional systems, as presented in the related work, where this change has to happen at compile time. This goal oriented approach enables us to detach the link between the activity recognition process and the types and amount of sensors used, as multiple sensing missions with varying sensor resources can be used for recognizing the same activities.

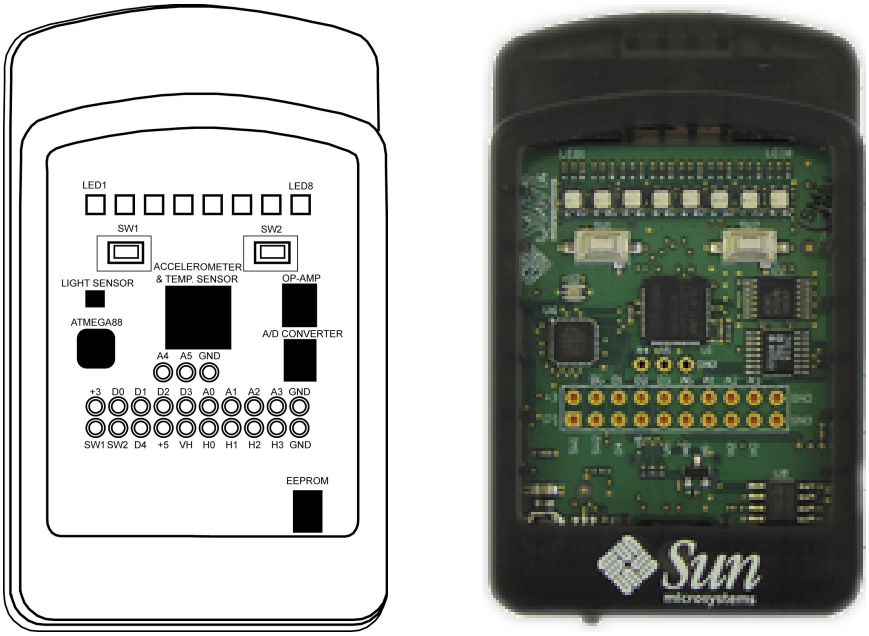
Furthermore through using the sensing mission and their containment hierarchy, the creation of a sensor ensemble is only constrained by the data which needs to be gathered from the different body positions and not by MAC addresses of the sensors involved as seen in traditional multi-sensor systems.

Each sensor node can communicate with its environment, state its capabilities and define what is needed from others to fulfil a sensing mission. This behaviour enables the sensor nodes to self-organize themselves and self-manage the sensor ensemble, open communication channels and deliver data from the sensors to the inquirer of a sensing mission.

As changes, disappearing and joining of sensor nodes, may happen spontaneously in a wireless sensor network **D**arSens reuses the self-organization approach, which created the sensor ensemble in the first place to mitigate this change and self-adapt, e.g. by replacing a disappeared sensor node with a sensor node with the same capabilities, without disturbing the activity recognition.

3.8 Technology

For the implementation of **D**arSens the Sun Microsystems Small Programmable Object Technology (SPOT) platform was chosen. It incorporates a system on a chip design which includes a ARM920T ARM Thumb Processor clocked at a maximum speed of 180 MHz. As for memory the Spot contains a 4 MB NOR Flash memory and 512 KB RAM whereas parts of the flash are occupied by the bootloader, the virtual machine, some libraries and user applications.



(a) Scheme (b) Photo

Figure 3.13: Sun SPOT

For communication with the outside world the Sun Spot features 8 tri-color leds, type B USB mini connectors, a few general purpose input/output (GPIO) pins which can be used to extend the Sun Spot with additional hardware and a IEEE 802.15.4 compliant wireless radio chip.

The Sun Spot also features a temperature, light and acceleration sensor. The later is heavily used in **D**arSens as it is the main source of information for activity recognition. Additionally two buttons are located at the top side of the Sun Spot which can be used for user input.

The Sun SPOT includes an Software Development Kit (SDK) to implement own applications for the Sun SPOT platform. Software for the Sun Spot is written in the Java Programming Language, to be more precise in the Java Mobile Edition which is a subset of the standard Java Language. The complete software is available under an open source license as well as the hardware schematics.

Chapter 4

Architecture

The implementation of **D**arSens features a multi layered architecture, cf. Figure 4.1, which consists of several services. The lowest layer consists of two services, to be specific a sensor and communication service, which are closely coupled to the hardware. As the one access the acceleration sensor of the Sun SPOT and the other accesses the wireless communication facilities. On top of it are the core services of **D**arSens which heavily use those hardware coupled services, namely interaction, messaging, feature and classification service. The behaviour of all services can be adjusted by using a configuration class.

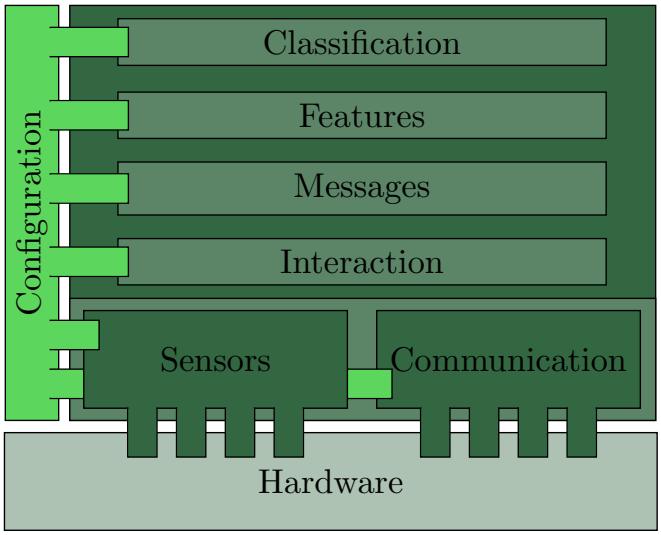


Figure 4.1: Software architecture of DarSens

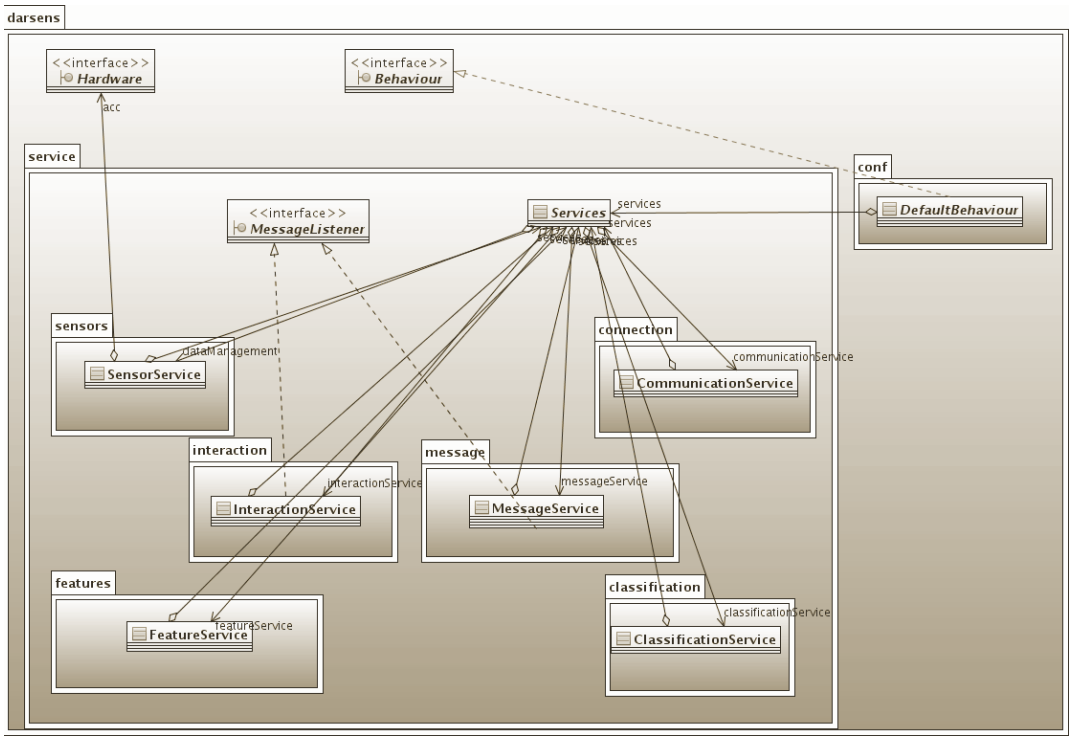


Figure 4.2: Service UML diagramm of DarSens

4.1 Services

At the core of the **D**arSens framework, cf. Figure 4.2 lies the Service class, cf. Figure 4.3. With this class one can access any service running, the MAC address of the spot it is running on, the containment hierarchy which describes where the spot is placed on – aka "bodyPart" it is located on –, the configuration or behaviour used to customize this spot and finally access the LED of the Sun Spot for debugging/information purposes. The service class glues together all layers of the architecture, cf. Figure 4.1 and is used by all services to accesses the other service or facilities of the framework. Denote that this is an abstract class and has to be customized for usage. There is a implementation for Sun Spot base stations and for regular Sun Spots because e.g. the base station doesn't have LEDs, or an accelerometer and it doesn't need to run sensor, interaction, feature, classification or message service.

In the following all services, which are grouped into different packages with their closely related classes, of the framework will be described.

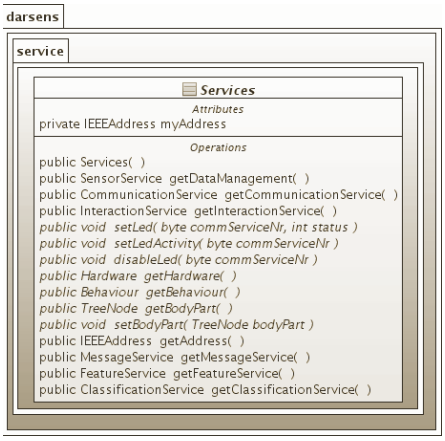


Figure 4.3: UML diagram of the Service class

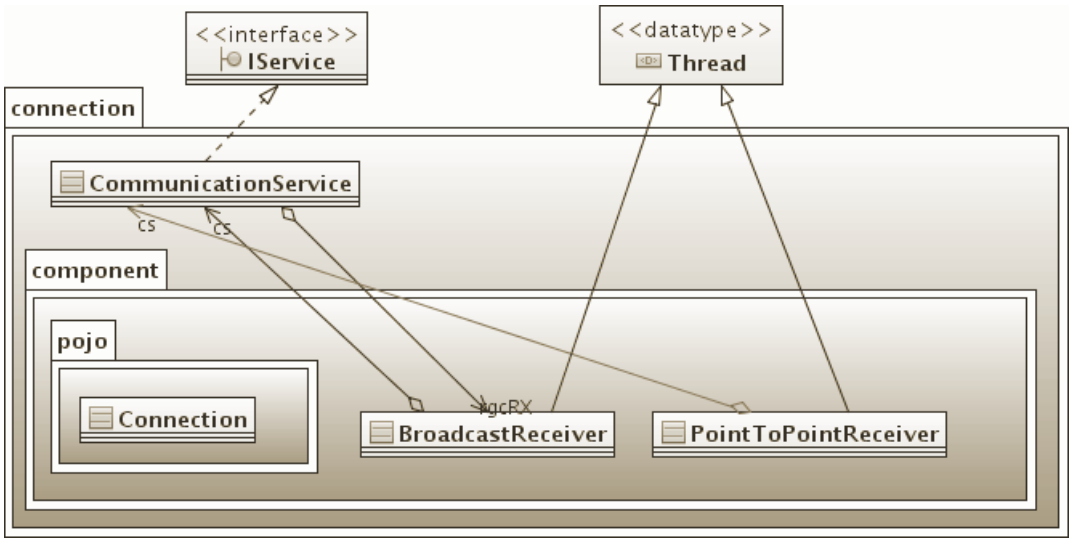


Figure 4.4: UML diagram of CommunicationService and closely related classes

4.1.1 Communication Service

The CommunicationService, cf. Figure 4.4, handles communication. It can and should be used by using the Services class, cf. Figure 4.5 class it also implements the IService Interface from the Sun SPOT API. This can be used for life-cycle management of services, for further details see `http://www.sunspotworld.com/docs/javadoc/com/sun/spot/service/IService.html`. It maintains two receiving channels, one to receive broadcast messages and the other to receive direct messages. Additionally it will open connections to other sensor nodes, if needed by other services, and close and clean up those connection after some time.

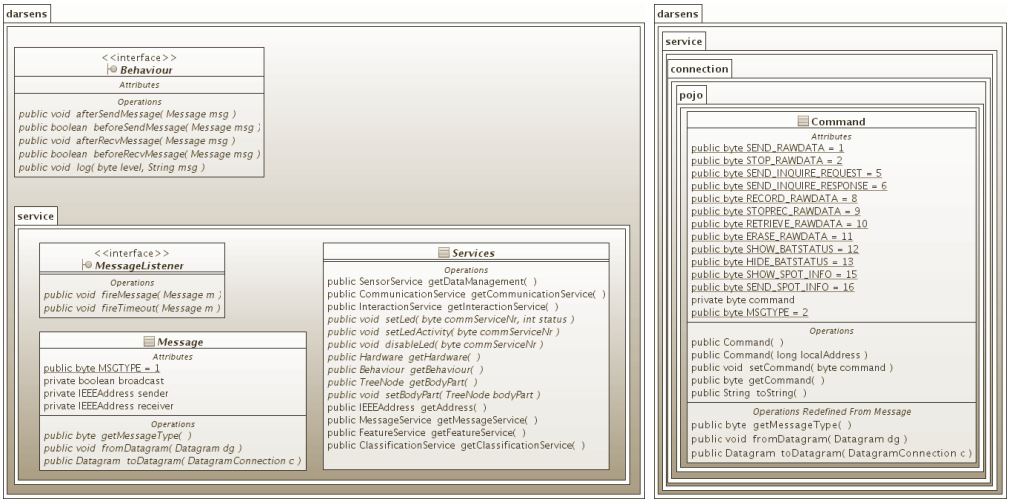
4.1.1.1 Receiving

For incoming connections the CommunicationService uses two different classes. For receiving broadcast messages, the BroadcastReceiver class is used, and for receiving messages directed to the sensor node, the PointToPointReceiver class is used. Both classes implement the Thread class and are hence used as Threads.

Those two classes will receive Datagrams from the physical connection. They are creating inherited instances of the Message class. The Message class bundles information common to every message sent and received: a broadcast flag, the sender address, the receiver address and the message type. Basically this is the header information for the messages used within the framework. Every implementing class of the Message class will have a different message type and hence the BroadcastReceiver and PointToPointReceiver can distinguish those messages. Those classes all implement the fromDatagram method of the Message Class to read the contents of the datagram and store the retrieved data in the corresponding fields. This is basically a payload evaluation.

After the PointToPointReceiver or BroadcastReceiver class read in the content of the datagram and evaluated its content, they will call the receivedMessage method in the CommunicationService. This method will trigger “pre-receive” behaviour, queuing of the message and “post-receive” behaviour. An example of pre-receive behaviour would be to stop messages from being queued, e.g. a sensor node may not participate in self-organization, like a mobile phone and hence will not be interested to process this kind of messages. An example for post-receive behaviour would be logging “message received and queued”.

The method receivedMessage in the CommunicationService will also notify the thread which is responsible for processing the message queue. This thread will in turn try to empty the message queue and notify all MessageListeners registered at the Com-



(a) Services, Message, MessageListener and Behaviour class/interface (b) Command class

Figure 4.5: UML diagrams of Services, Message, MessageListener and Behaviour class/interface and an implementation of the Message class: Command class

muncationService, e.g. the sensor service. The MessageListener interface consists of two methods: fireMessage, which will be used to process the received message and fireTimeout which will be called when the delivery of a message fails. This is necessary for e.g. termination of a running sensing mission when the parent node in the containment hierarchy about to receive data is not reachable any more.

4.1.1.2 Sending

A connection class bundles all necessary information about an outgoing connection. It contains the remote address and port, a dirty flag, a cleanUpAllowed flag and the "physical" connection itself. The dirty flag and the cleanUpAllowed flag are used within the CommunicationService to handle the connection management. For each connection the CommunicationService will evaluate periodically if this connection is still in use by setting the dirty flag of the connection to true. In the next cycle if the connection is still marked dirty it will close the connection. However if this connection is used the dirty flag will be set to false by its user and the connection will not be closed. Alternatively if the cleanUpAllowed flag is set to false for a Connection class it cannot be closed by the communication service.

Additionally there are three types of outgoing connections available. A broadcast connection and two types of direct connections. One direct connection for datagram based communication, which is used for normal messages based on the Message class. The

second type of direct connection is a stream based communication which is used for sending data too large for a single datagram. This is e.g. used for fetching classification model for a sensing mission.

4.1.2 Sensor Service

The sensor service is able to extract data from the acceleration sensors and either relay it to a remote listener – e.g. for visualisation – feed it into the recognition chain by using the local feature service or a remote one or store it into the flash memory. The later method is needed for recording training data, which can later be used to create classification models because its is not possible to transmit the gathered data at high frequency to a remote location, because of the bandwidth restriction of the IEEE 802.15.4 standard.

The Sensor Service is closely coupled to the hardware and can be started by attaching listeners to the data it is able to gather. Basically either by adding a SensorDataListener, like the FeatureService does, or by specifying the MAC Address of a remote receiver. In both cases the SensorData class is used to encapsulate the gathered data, at a given time point, and deliver it to the requester.

Additionally using the Command class, cf. Figure 4.5(b), with the following commands: SEND_RAWDATA, STOP_RAWDATA, RECORD_RAWDATA, STOPREC_RAWDATA, RETRIEVE_RAWDATA, ERASE_RAWDATA, one can invoke several functions of the Sensor Service directly. SEND_RAWDATA and STOP_RAWDATA are used for retrieving and stop retrieving raw data. The raw sensor data retrieved can be visualised using the DarSens EKG Application, cf. Figure A.4. Further more using RECORD_RAWDATA, STOPREC_RAWDATA, RETRIEVE_RAWDATA, ERASE_RAWDATA one can accesses the capabilities of the Sensor Service to access the flash storage of the sensor node. The first two commands work similar as SEND_RAWDATA and STOP_RAWDATA but the data is transferred into the flash memory using the toFlashString() method of the SensorData class, whereas one line in the flash storage represents one sample point in time. Using the RETRIEVE_RAWDATA command the data can be retrieved and it will be printed out to standard output (stdout). For further processing it is recommended to redirect stdout into a file when the sensor node is connected via USB to a computer. After retrieval the data in the flash storage can and should be erased to free up the space. Currently a little less than 10 minutes of data can be recorded using a sampling frequency of $20Hz$ which is sufficient for activity recognition according to [25].

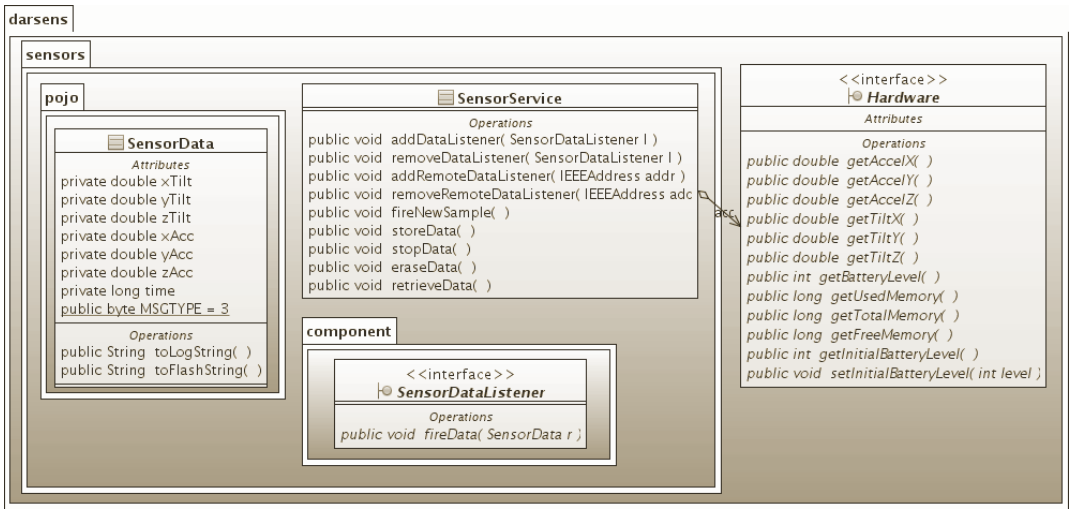


Figure 4.6: UML diagram of SensorService and closely related classes

4.1.3 Interaction Service

The interaction service, cf. Figure 4.7 is used within **D**arSens to implement the self-organization phase. It will receive and evaluate sensing missions and try to satisfy them either solely or by the help of other sensor nodes and respond to the respective requester.

The Interaction Service will add itself as a MessageListener at the communication service. Hence for any message received by the communication service the fireMessage() method is called for the interaction service which in turn will evaluate the message type and contents. The interaction service will only process messages of the following types: InteractionTree and InteractionDiffNumber. Both are used for the self-organization process presented in section 3.4. An incoming InteractionTree message contains the containment hierarchy of the sensing mission, cf. Figure 4.7 bodyPart, an unique identifier and a type identifier, which can be one of the following: REQUEST_CAPABILITIES, RESPONSE_CAPABILITIES SEND_CAPABILITIES, ECHO_CAPABILITIES, ACQUIRE_ACTIVITY, DISPOSE_ACTIVITY, DATA_SOON. Whereas the latter three are used solely within the MessageService.

When broadcasting a sensing mission the InteractionTree message's type identifier is REQUEST_CAPABILITIES. Upon arrival of such a message the InteractionService will check if there is an overlap between the local containment hierarchy, representing the position of the sensor node on the body, which is stored in the Services class, and the incoming containment hierarchy. If both hierarchies don't overlap nothing will happen but if an overlap exits the difference number will be calculated and based on the difference in nodes the InteractionService proceeds differently.

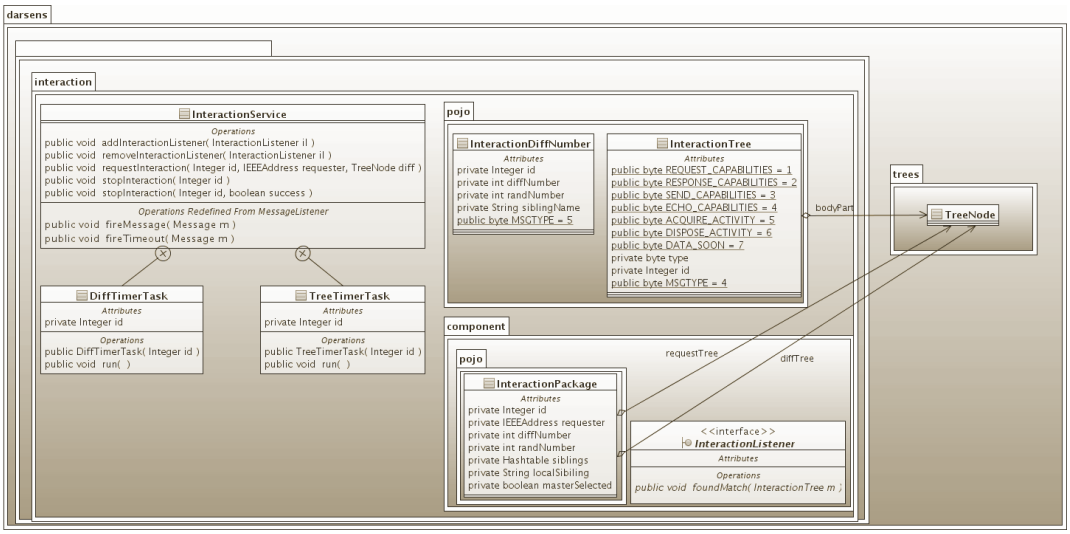
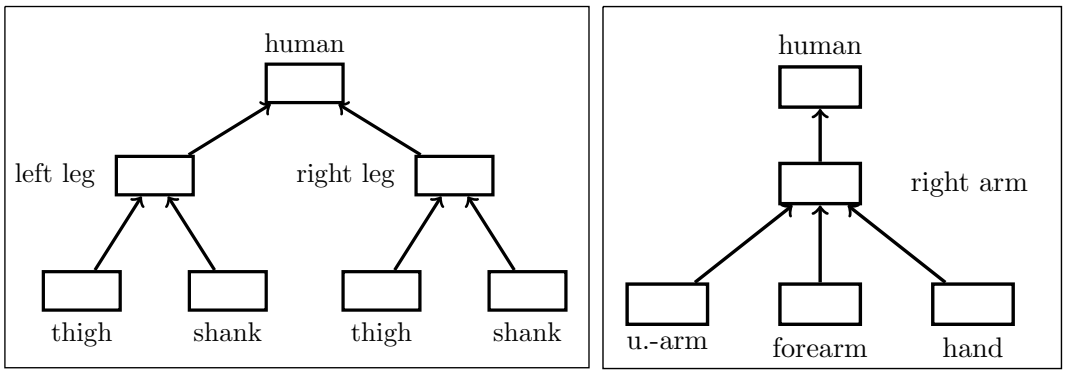


Figure 4.7: UML diagram of InteractionService and closely related classes

difference of zero the interaction service will create an InteractionTree message based upon the received message with type RESPONSE_CAPABILITIES and same containment hierarchy but with his MAC-Address assigned to all nodes. This is the implementation of the concept satisfied sensing mission.

difference of one the interaction service will create an InteractionTree message based upon the difference tree between the local and received containment hierarchy with type identifier REQUEST_CAPABILITIES and a new unique id. The difference tree and the requested tree will be stored in an InteractionPackage's diffTree and requestTree respectively, along with the id and the MAC address of the requester. Furthermore a TreeTimerTask will be created which in turn will stop the self-organization process for this received sensing mission if it can't be satisfied after a certain time by receiving a InteractionTree message containing the difference tree with containing a MAC-Address at every node and the type RESPONSE_CAPABILITIES.

difference greater than one the interaction service will store the same data in an InteractionPackage, like the previous case. However he will start a master selection by sending out an InteractionDiffNumber with the difference number and additional random number and its sibling name. Denote that the siblings names are the node names after the first fork after the node, for which a master is to be elected, cf. Figure 4.8. This InteractionDiffNumber message will be broadcast and the InteractionDiffNumber messages of other nodes will be received and compared to the local information. If one of the other sensor nodes has a lower difference number and optional random number in case of equality the master selection process will stop as it continued on the other sensor node. On the other



(a) sibling names for the master selection at the root node: left leg, right leg

(b) sibling names for the master selection at the root node: u-arm, forearm, hand

Figure 4.8: InteractionDiffNumber - siblings name explained

hand when the sensor node received a InteractionDiffNumber message from all siblings (which are stored in the InteractionPackage's siblings field) and it has definitely the highest number, it will elect itself as master. Alternatively because all InteractionDiffNumber messages are broadcasts it may happen that the sensor node doesn't receive a message from siblings or because the siblings are to far away to have received the original request it will elected itself automatically as master after a certain period this behaviour is implemented through the DiffTimerTask. After the election as master it will REQUEST_CAPABILITIES for the difference tree just like in the previous case.

4.1.4 Message Service

The message service is used to implement the setup of the network, the activity recognition process and the delivery of data within the running sensing mission. Upon receiving a InteractionTree message of type ACQUIRE_ACTIVITY the MessageService will use the information – id, sender mac and containment hierarchy, which equates to the satisfied sensing mission – in the message to setup a MessageBuffer object for this sensing mission. In turn the MessageService will schedule the execution of two periodic tasks, namely startTree and sendData.

The former method will setup the local processing by invoking the recursive function createBuffers. It does the following for each node in the containment hierarchy, with MAC address assigned:

- add a buffer within the MessageBuffer

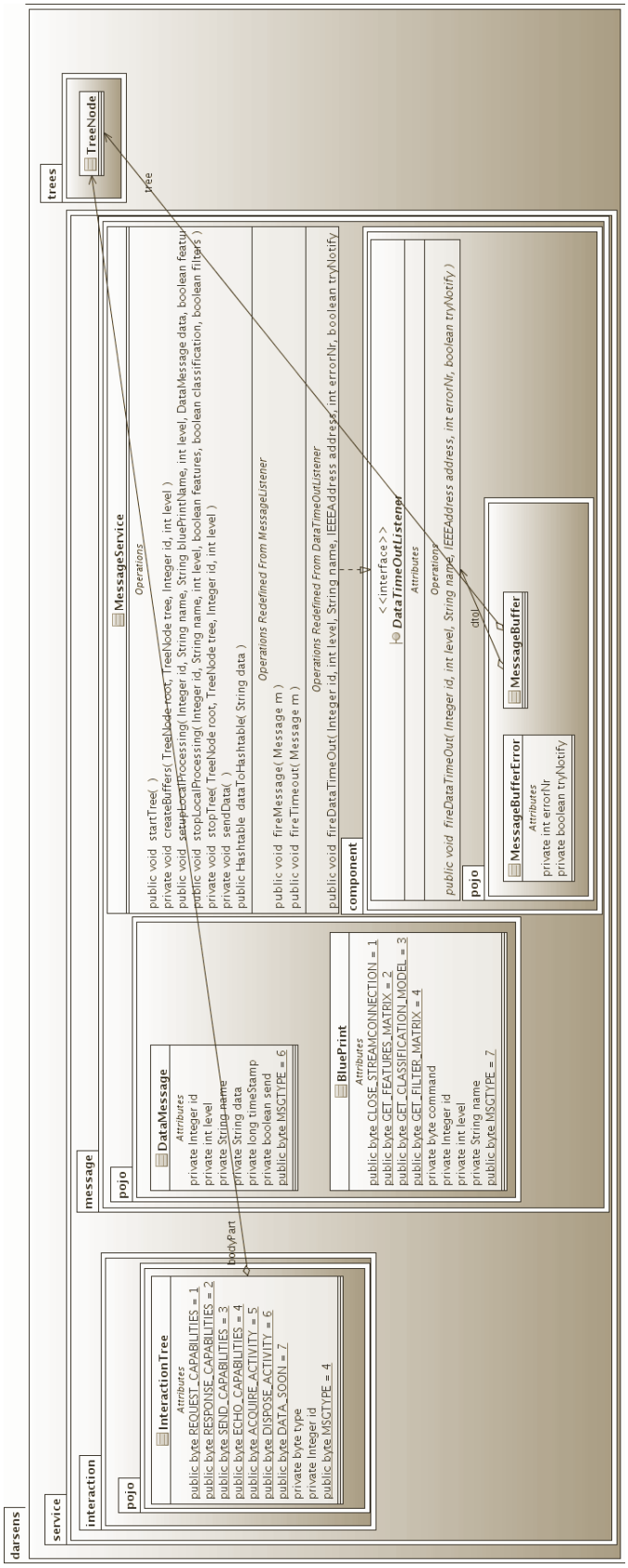


Figure 4.9: UML diagram of Message Service and closely related classes

- create a DataMessage object, its data field will later and periodically be filled by the activity recognition chain running on this node and sent by the MessageService

- invoke setupLocalProcessing(), which fetches the classification model and feature matrix from the requester if needed; Denote through the size of the data to be transmitted a StreamConnection will be opened, the data will be fetched and the connection will be closed. To control and implement this behaviour the Blueprint class is used which exactly specifies what is needed from the remote side.
  Using the classification model and feature matrix obtained their respective service will be invoked to start the processing or creation of data

When the recursive traversal of the createBuffers hits a node, which is not assigned to this nodes MAC address, it will notify the node by sending an InteractionTree message of type ACQUIRE_ACTIVITY and terminate further traversal as this is done on the other node.

The later method, sendData, will simply send all DataMessages created by the startTree method. Furthermore the method will also fetch and remove data from the MessageBuffer prepare the data for the classification service by invoking dataToHashtable function and update the data the classification services operates on.

4.1.4.1 MessageBuffer

The MessageBuffer class holds all information belonging to a deployed sensing mission. For every node in the sensing mission with a MAC address assigned to this node a Buffer exists with a fixed size. If the buffer is filled faster then the data can be processed and removed, the oldest data is removed without processing. This ensures that the buffer will not be filled till the device is out of memory and also keeps the data "fresh". If on the other hand the data is processed and removed faster then the buffer can be filled a MessageBufferError will be created. Denote that the buffer will not remove the last element to insure that the processing can continue. However after a certain amount of continuous errors the MessageBuffer will fire a DataTimeOutListener event and notify the MessageService. This error notification contains all necessary information for further processing of the error, namely the place in the containment hierarchy, of the sensing mission where the error occurred and the MAC Address from the physical sensor node, which is not delivering the information. The MessageService uses this information to start the self-adaptation behaviour.

In case of a notification the MessageService will create a InteractionTree Message with type ACQUIRE_ACTIVITY similar to the one in the createBuffers method. If this

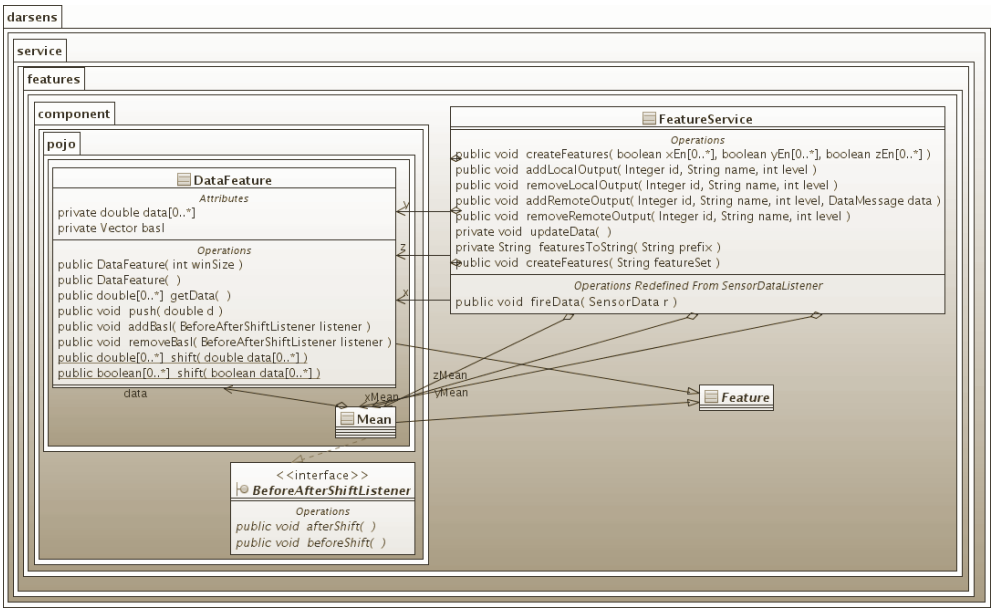


Figure 4.10: UML diagram of FeatureService and closly related classes

message is received by another node it will either start the setup behaviour described earlier – e.g. because the node rebooted because of an system error and has lost all runtime information – or reply with a InteractionTree Message with type DATA_SOON stating the setup is complete and that data will soon be available. This DATA_SOON message will reset the error count in order to stop the self-adaptation behaviour.

If the sensor node is not able to send a message to the sensor node in question because there is no route to the sensor node (network layer) the Sun Spot framework will throw an appropriate exception and the self-adaptation behaviour will stop trying to notify the node and instead start a self-organization process for the failed part of the containment hierarchy. If the self-organization process is successful the MessageBuffer for the failed node will be updated removing the old data provider and instead adding the new data provider – the responder of the self-organization process. If the self-organization process fails continuously – for a few times – then the error can not be recovered at all and the sensor node will send an InteractionTree Message with type DISPOSE_ACTIVITY to the root node of the sensing mission and stop the local processing and notifying the sensor node's own children in the containment hierarchy of the failed sensing mission.

4.1.5 Feature Service

The feature service is able to process raw acceleration data and calculate the following features standard deviation, energy, mean, mean crossing rate, zero crossing rate,

axis	raw data	mean	variance	standard deviation	fluctuation	mean crossing rate	root mean square	zero crossing rate	energy
x	1	0	0	0	0	0	1	0	0
y	1	1	1	1	0	0	0	0	0
z	0	0	0	0	0	0	0	0	0

Table 4.1: String representation of a feature matrix

variance, root mean square and fluctuation for each axis separately. Those features are considered relevant features for activity recognition and were used in previous research [25]. As that there is a dependency between those features so in order to calculate e.g. mean crossing rate also mean has to be calculated, cf. the aggregations in Figure 4.11.

The FeatureService is started using the createFeatures method – invoked by the message service – which takes as an argument the textual representation of the feature matrix, an example can be seen in table 4.1, denote that the legend and white space is not part of the actual representation. The matrix will be translated into three arrays one for each line/axis. A 0 disables the computation of a feature a 1 enables it and the index in the matrix corresponds to the id of the feature. Based on the containment hierarchy the MessageService will either add a local or remote output. When new data is available the SensorService will notify the FeatureService using the fireData function. The FeatureService will in turn update the data and compute the features anew and additionally relay the computed features towards all local and remote targets.

4.1.5.1 Features, DataFeature, ...

Every feature has a unique id, which is used for internal representation, a feature name and stores calculated feature values. The feature interface provides getters for a common access pattern to any feature. Denote that all features currently operate on a sampling window. Therefore they share the same original data set for their computation, implemented within the DataFeature class. The DataFeature class stores a data array of the sampling window size. When data is pushed into a DataFeature it will notify all BeforeAfterShiftListener's using the beforeShift method. Then shift the data array by one index, removing the oldest element and in turn add the new data at the end of the array. This is e.g. useful for features that use a sum computation, removing

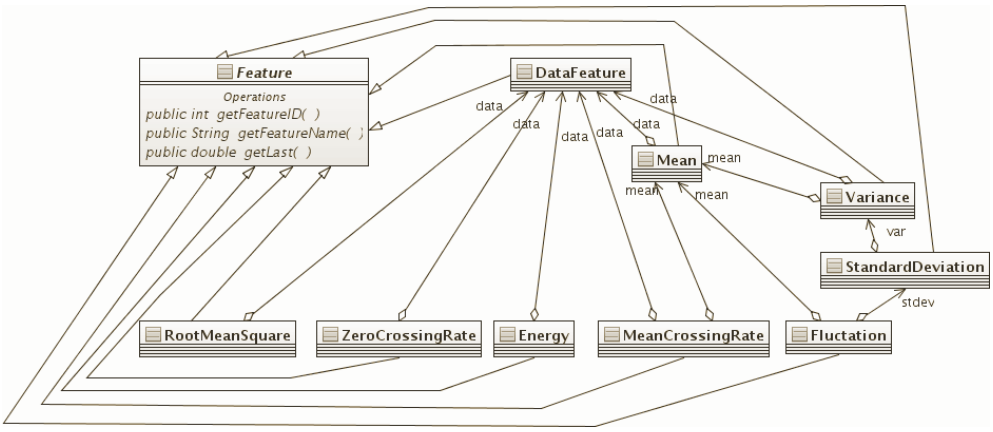


Figure 4.11: UML diagram of Features and closly related classes

the value of the first element before the shift and adding the newest element to the sum after the shift instead of iterating over summing up all values anew at each time when a new value appears.

4.1.6 Classification Service

The classification service is responsible for classification and filtering of data. The classification service's startClassification method will be invoked by the message service. Using the id, level and node name and a classification model. The classification service first parses the model's internal representation and creating a classifier out of it, which in turn will be used to create a classification task. The classification task will use the latest features/data to classify and in turn periodically update the data to be sent by the MessageService, namely the DataMessage's data field. This ensures that the data to be sent will be the newest data available.

Filter and FilterTask work in a similar fashion the only filter currently implemented is a pass through filter, which doesn't filter anything yet. More useful filter are an open issue and could be implemented in further work.

4.1.6.1 J48 Classifier

The only classifier currently implemented is the J48 classifier. J48 has been found one of the most efficient classifiers for activity recognition from accelerometer data according to [30]. Its textual representation can be stored and used to transfer the classification model to a sensor node. The implementation consists of a model class J48 and a tree

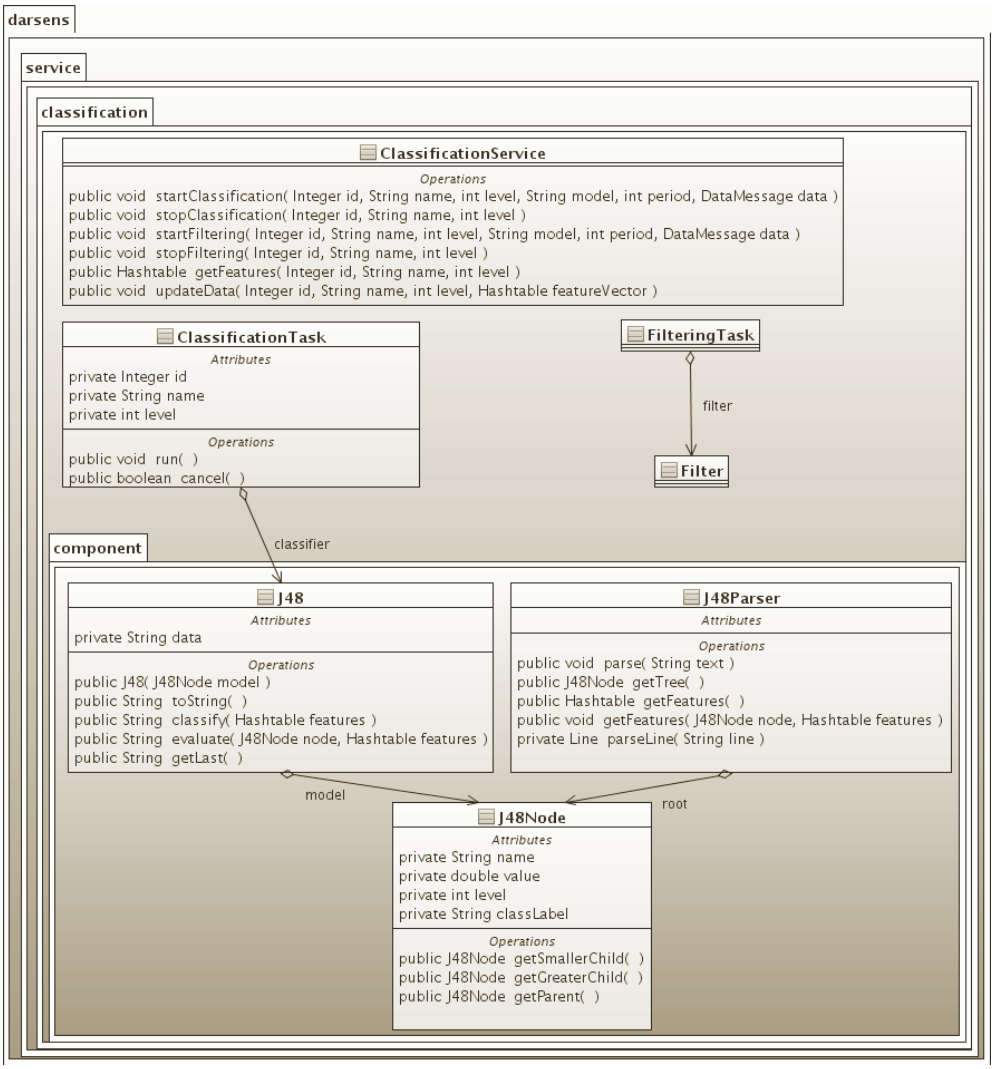


Figure 4.12: UML diagram of ClassificationService and closly related classes

node class J48Node. The model class holds the model based on tree nodes and the classification function which uses a feature set as input.

The J48Node class represents a node in the tree consisting of the following:

name this is the name of the feature

value this is the decision boundary of the feature

smallerChild a J48Node to expand if the current feature value is smaller than the value stored in this node

greaterChild a J48Node to expand if the current feature value is greater or equal than the value stored in this node

If the J48Node has neither a smallerChild or a greaterChild its is a leaf node and has a class label assigned.

The J48 classifier will classify, a new sample of features, by evaluating its model by invoking the recursive function evaluate with the models root node and the new sample. The recursive function will examine the features name at the current node and get the appropriate value from the feature set. If the value is smaller than the value stored in the J48Node it will expand and evaluate using the node's smallerChild. If the value is greater or equal than the node's value the classifier will expand and evaluate using the node's greaterChild. This process is continued until the evaluation function finds a leaf node and returns the class label attached to this node.

4.2 Framework

In Figure 4.13 a more detailed picture of the framework is presented than in 4.2. It shows how everything fits together.

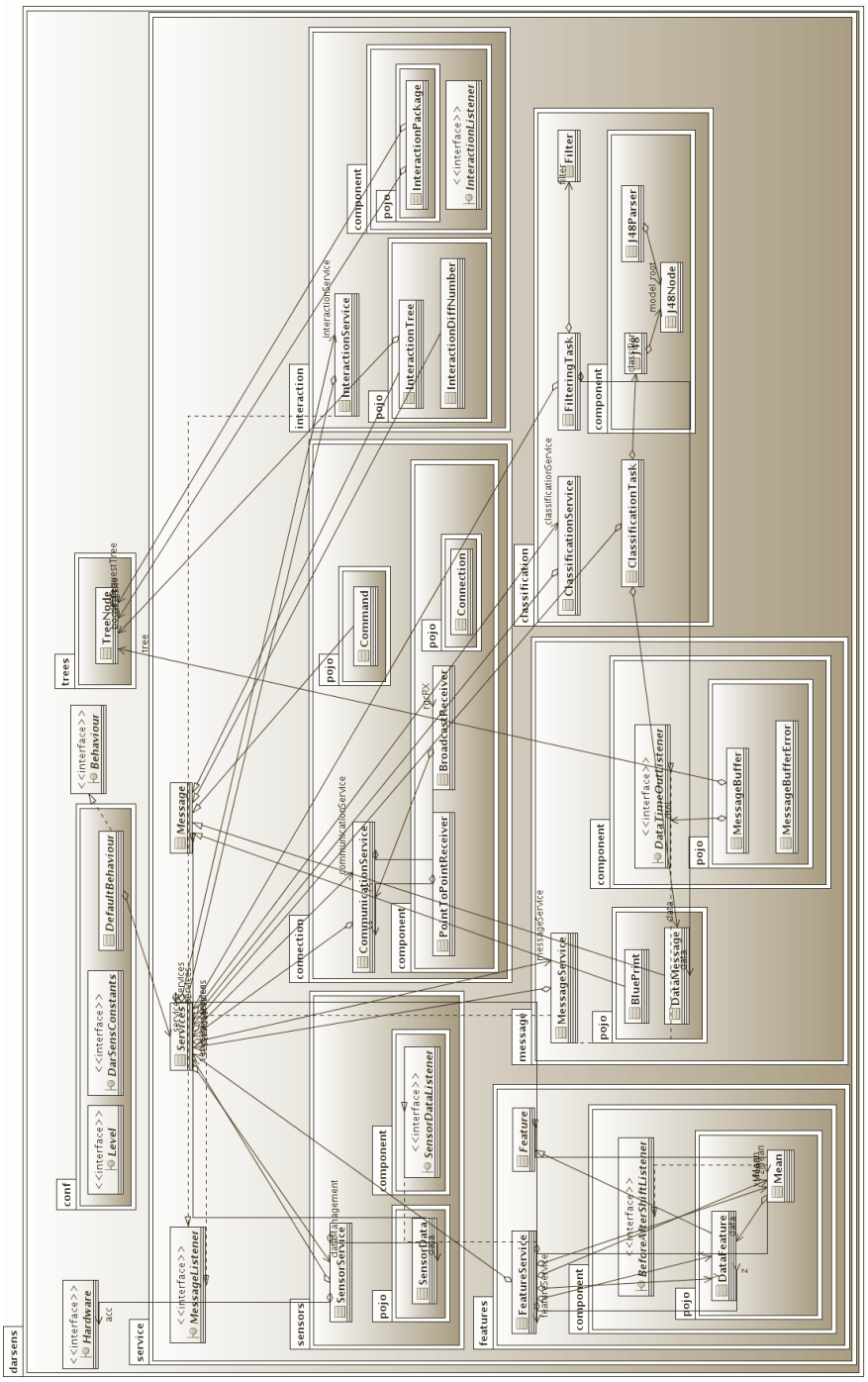


Figure 4.13: UML diagram of DarSens

Chapter 5

Evaluation

In order to evaluate the approach of **D**arSens three experiments have been conducted. The first experiment examines the self-organization phase and how long it takes the sensor nodes within a network to satisfy a sensing mission. The second experiment looks at different system properties when using **D**arSens in an activity recognition scenario. The third experiment shows that the system can self-adapt to changes in the sensor network. These experiments also provide proof for the hypotheses stated in the introduction.

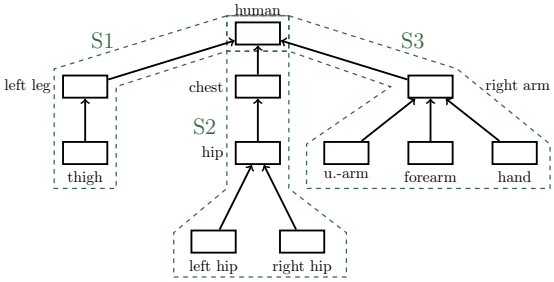


Figure 5.1: Sensing mission(s) used for self-organization

5.1 Self-organization

In order to start a recognition process sensors need to interact, more precisely self-organize. A "requester" will broadcast a sensing mission and the sensors will try to satisfy it either solo or together. The following parts of the hypotheses are to be proven with this experiment:

a) The asserted benefit of using a sensing goal initiated multi-sensor activity recognition approach is that such a system is capable of recognizing different sets of activities at runtime by only changing the recognition goal which corresponds to changing the used sensing mission. Therefore its necessary to proof that a sensor network can self-organize itself for different sensing missions without recompiling.

b) To avouch that a sensor is able to communicate and interact with its environment and specify what 'duties' are needed from others, and what it can do itself, we specify sensing missions which are unable to be satisfied by a single sensor node, but only by cooperation of sensor nodes.

5.1.1 Experiment setup

A total of six Sun Spots where randomly placed on a $30cm^2$ area. Every sensor had a distinguished position on the body assigned to it, cf. the leaf nodes in Figure 5.1.1. A base station was attached to a computer running a version of the **D**arSens EKG, cf Appendix A, which was used to measure the time needed for self-organization. The measurement was started when the sensing mission was broadcasted to the network and the time difference was calculated for each returned satisfied sensing mission.

Different types of sensing missions were used to test the time needed for self-organization. The sensing missions are based on three different types of sensing mis-

	S1	S2	S3	S1S2	S1S3	S2S3	S1S2S3
leaf nodes/number of sensor nodes	1	2	3	3	4	5	6
total number of nodes	3	5	5	7	7	9	11
tree height	3	4	3	4	3	4	4
master selection	no	no	yes	yes	yes	yes	yes

Table 5.1: Self-organization sensing missions properties

sions: S1, S2, S3, cf. Figure 5.1.1 dashed line and composition of them, summing up to a total of seven different sensing missions. These sensing missions differ in the total number of sensor nodes used, the height of the tree, the total number of nodes and if a master selection will take place during the self-organization phase, cf. Table 5.1 for an overview of the properties.

5.1.2 Experiment implementation and conclusion

With the start of the broadcast of the respective sensing mission, the time measurement started and, upon receiving respective responses of the sensor network, the time difference was calculated. Denote, that for almost every broadcast of a sensing mission multiple valid responses were received, if the sensing mission contained more than one leaf node. However only the first response were used for our plots. The experiment was repeated ten times for every sensing mission.

The resulting time for the completion of the self-organization process for the different sensing missions are presented in Figure 5.2(a) and 5.2(b). Figure 5.2(c) shows the time needed for the self-organization process regarding to the number of leaf nodes, which is also the number of physical sensor node participating in the sensing mission. Figure 5.2(d) on the other hand shows the time for self-organization with respect to the total number of nodes in the sensing mission. Denote that in both figures the dashed line represents the average time to complete the self-organization process.

It is clearly visible that if more sensor have to participate to satisfy a sensing mission, the self-organization process takes longer to complete. This can be related to two facts. First one is that the sensing mission is transmitted in textual form, hence "bigger" sensing missions take longer to transmit and of course the parsing of the sensing missions takes more time. Among other things by shortening the textual representation of the sensing mission the results from the framework in [14] have been improved. The second fact is that through the use of broadcasts, the error, introduced by them, has to be mitigated. Unlike when using direct communication between two sensor nodes, when using broadcast there is no reception guarantee. Hence messages broadcast can and will be lost – which has been observed through the development of **D**arSens – and

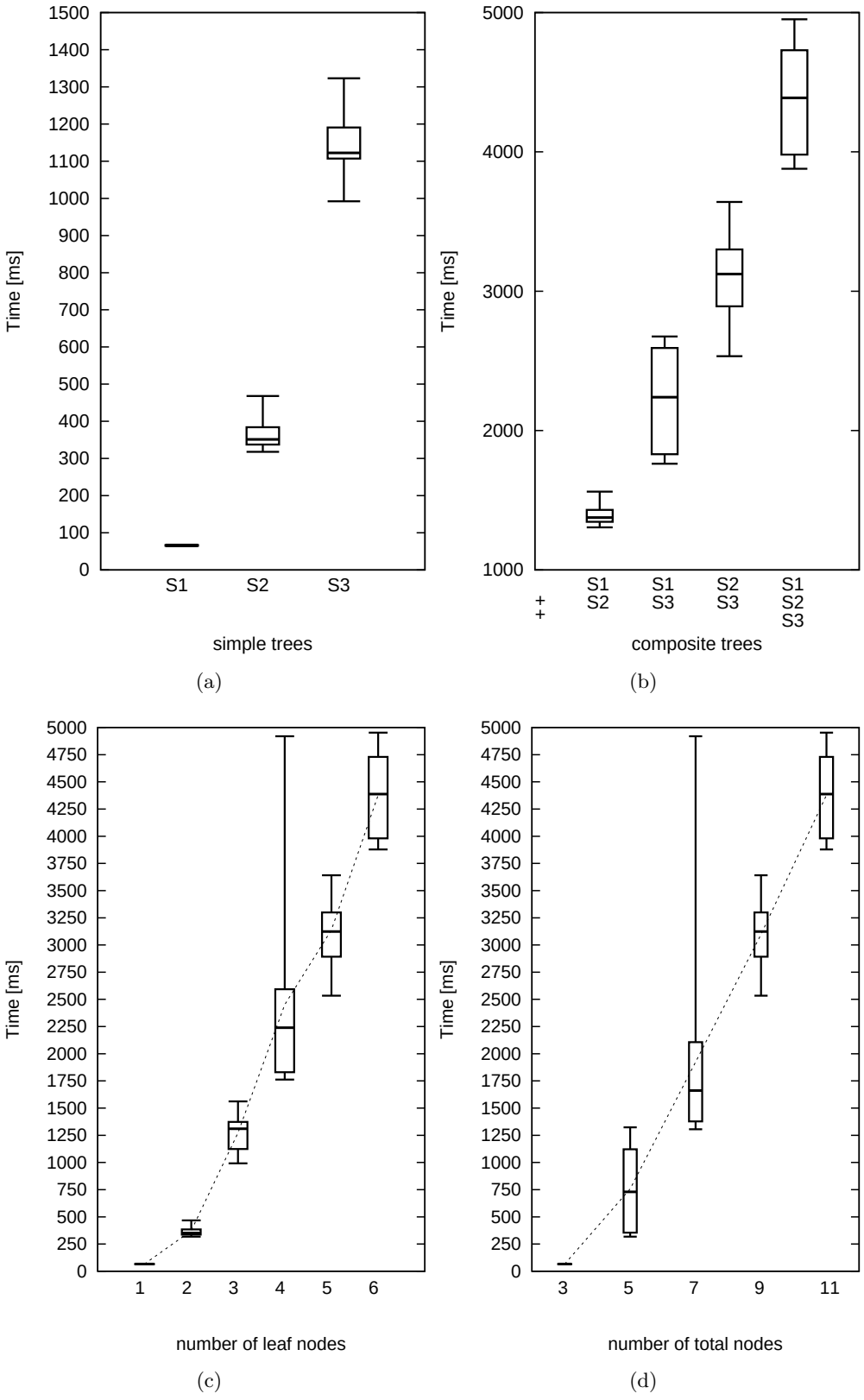


Figure 5.2: Self-organization experiment plots

not be retransmitted unless the application developer takes care of it. As for the interaction service, it will retransmitted certain requests if it doesn't get a response in time. However the self-organization process relies on broadcasts heavily, for requests of sensing missions and sub-sensing missions, and especially for the master selection.

Furthermore as seen in Figure 5.2(a) and 5.2(c) there is big time increase between S2 and S3/2 or 3 leaf nodes. The reason for this behaviour is that in case of three nodes or S3 a master selection takes place while in case of two nodes or S2 only one sub request is created to satisfy the sensing mission. This is because there is a time-out within the master selection process for receiving replies, which is between 300-400 ms.

Concluded one could argue that it would be more feasible to use a discovery approach which would lead to an routing table so that its not necessary to rely on broadcasts. However this would lead to a conflict with the requirement enforced by the idea of opportunistic activity recognition namely that you will deal with a highly dynamic environment. Hence sensor nodes will leave and join the network on a regular basis and keeping routing tables updated would be a constant effort which will drain the battery of the sensor nodes. So this dynamic approach with broadcasts may seem slower however energy wise it will be more efficient as network organization will take place only on demand and furthermore an application developer/framework user can use previously satisfied sensing missions and use them to skip the self-organization phase entirely. This even works if some of the sensors are not available any more because they will be replaced using the self-adaptation technique.

As for the parts of the hypotheses to be proven: ad a) Has been affirmed successfully because all different sensing missions were broadcasted and satisfied using the same software version. ad b) Has been avouched successfully because we used sensing missions with more than one sensor node involved e.g. S3.

5.2 Activity recognition

After self-organization is complete the activity recognition process can start. When considering a typical activity recognition scenario there are trade-offs between different requirements, which are mutually exclusive to some extend. Derived from previous activity recognition projects in research, cf. Chapter 2 the following three requirements are considered essential:

accuracy to which percentage is the system able to recognize a certain set of activities correctly

speed in term of set-up time

energy efficiency how much energy is consumed when performing an activity recognition task

The following parts of the hypotheses are to be proven with this experiment:

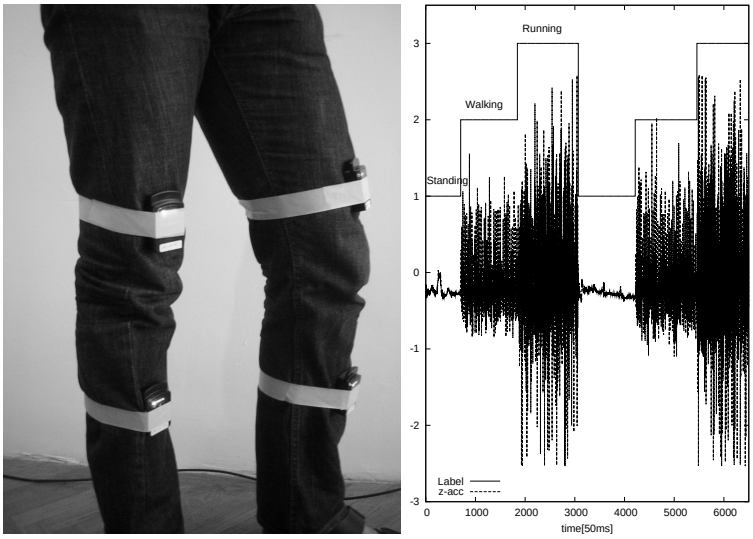
Using the self-organization principles the sensor ensemble can perform self-management as it can satisfy a recognition goal and the corresponding sensing missions. Furthermore the sensor ensemble will open communication channels and deliver data, process it and send the recognized activity to the requesting peer. Hence it is necessary to use sensing missions, which include more than one sensor node and where the data is processed on more than one device.

5.2.1 Experiment setup

To show the feasibility of our approach, an artificial activity recognition scenario was created with the following three types of activities to distinguish: standing, walking and running and took a supervised learning approach.

A dataset of six minutes was recorded on four body positions, namely thigh and shank on both legs, cf. Figure 5.3(a). Each activity was performed for one third of the duration, as outlined in Figure 5.3(b). The data was labelled with the activity performed at that moment. Afterwards the following features were extracted from the dataset: standard deviation, energy, mean, mean crossing rate, zero crossing rate, variance, root mean square and fluctuation. This features have been also used in previous research [25]. For each axis the raw data and the 8 computed features sum up to a total of 27 values per sample in the recording. In order to decrease the computational load and the bandwidth needed for transmission only two of three axis were used. The axis measuring forward and upward acceleration were found to be the most useful ones by [9]. This leaves a total of 18 values per sample, which was further diminished by using a "best-first-search" feature selection algorithm provided by the WEKA machine learning toolkit [12]. Up to six features for both axis together were then used for training a classifier. In this case, the J48 classifier, which has been found one of the most efficient classifier for activity recognition from accelerometer data according to [30].

Every sensing mission may feature multiple classifiers. Classifier were trained for each of the four body positions. In order to train meta classifiers, a meta dataset by combining the output of the classifiers and the true label was created. This meta dataset was then used to train another J48 classifier. E.g. for training the classifier of the leaf nodes in



(a) deployment (b) labelled data

Figure 5.3: Activity recognition experiment sensor setup and recorded data

SSM1, cf. Figure 5.4(b), the recorded data set was used; for the classifier in left leg the output of its corresponding child nodes were used – the leaf nodes: thigh and shank – to create a meta data set and train the classifier.

5.2.2 Experiment implementation and conclusion

In order to show the usefulness of our approach in an activity recognition approach, three sensing missions, cf. Figure 5.4 were choosen with different layouts. They do not only differ in the layout but can also be intuitively be assigned to one of the typical requirements of an activity recognition system. SSM3 would be the fastest and most energy efficient. SSM2 would be the most accurate and SSM1 the good average. The performance of the sensor network was measured while performing the same procedure used for gathering the training data, namely walking, running and standing for two minutes each. Denote that in all three sensing missions the features are calculated on the leaf nodes and used as input vector for the classifier located on the same node. This saves a lot of bandwidth because only one value, the classification output, has to be transmitted instead of the up to 27 values produced by a feature service. As already explained earlier the meta-classifiers of SSM1 and SSM2 use the classification output of their respective child nodes as input and classify correspondingly. Denote that in Figure 5.4 exemplary satisfied sensing missions are shown. The numbers symbolize the MAC-Address of a certain sensor node and show that the tasks of multiple nodes are executed on the same physical device. Which in turn concludes that the data flow bottom up will not always be transmitted wireless, cf. Figure 5.4 dotted line, but be

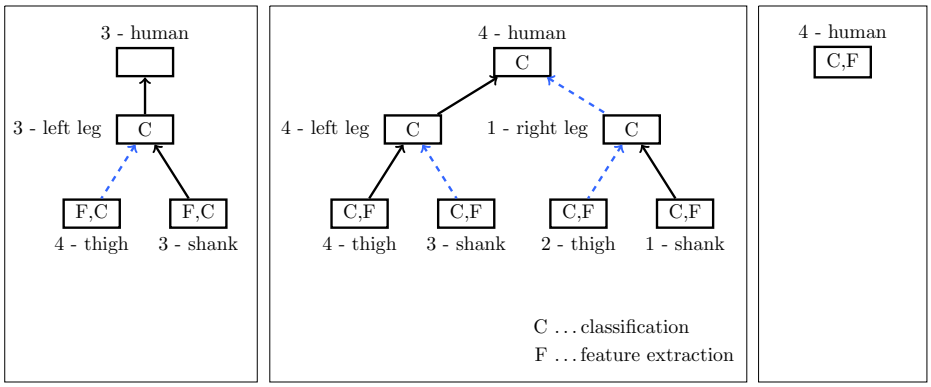
transmitted locally, e.g. in SSM2 from the node "left leg" to "human" node. The size of the local transmitted data will not be added to the communication effort in table 5.2 as it does not consume transmission power to transfer it. Furthermore any type of data produced in the activity recognition process (raw- , feature extraction- , classification-data) which contains the same information as the previously produced one will not be transmitted at all in order to save bandwidth and power needed for the transmission. In order to avoid the triggering of self-adaptation behaviour on the ancestor node, the same data will be transmitted after some time. Denote that **D**arSens not only samples with a frequency of 20 Hz but also transmits data with this frequency, which is a good value as shown by research conducted by[25]. [25] identified that the sampling frequency from acceleration data of a human subject stabilizes between 15 and 20 Hz, meaning a higher sampling frequency will not yield more significance in relation to the activity recognition task.

The results of the experiments are shown in Table 5.4. Denote that the number of nodes in the sensing mission is always the same as the number of wireless connections needed. As accuracy's concerned the table presents both the accuracy calculated by the WEKA toolkit on the training set and the accuracy of the sensor system when doing the experiment.

It can be seen that there is a significant drop in accuracy between those two across all sensing missions. Possible causes are amongst others an accuracy drop due to displacement – previous research [21] showed that the drop can be as high as 72% and the training data set may be too small and hence the classifiers are overfitting. The later problem could of course be mitigated by obtaining more training data, however this would improve only the accuracy – and not change the systems behaviour – and is not the focus of this experiment.

When a sensing mission is deployed, it takes some time for the first activity information to reach the requester, due to processing delay and the time needed to set-up the tasks for the nodes in the containment hierarchy in the sensing mission. E.g. there is a two seconds delay before a sensor node will fetch a classification model or feature matrix from the requester. The time difference is referred to as time to data in Table 5.4. Denote that the fetching a classification model and a feature matrix will be performed in parallel within a level in the hierarchy and sequentially per level. Therefore the delay increases only slightly when the hierarchy is deeper or more complex, cf. time to data from SSM1 and SSM3.

Interestingly the total amount of data to be communicated wireless through the network scales very well with the amount of sensor nodes used. E.g. SSM1 has 35 KB having only one node and one wireless link and SSM2 has the double amount of wireless links



(a) SSM1 (b) SSM2 (c) SSM3

Figure 5.4: Exemplary satisfied sensing missions with marked wireless connections

and physical nodes and amount of data is doubled as well. When looking at SSM2, it would be expected that the amount of data to be transmitted be not only doubled but is thrice as high as SSM1 because the containment hierarchy is a mirror image of SSM1 and there are two more inter-system connection links. However through sending only the classification output and not sending the same data all the time the communication effort can be greatly reduced.

As far as energy consumption is concerned the results are not very accurate, because the Sun Spot API only delivers a very raw estimation of the real value. However the layout of the containment hierarchy of the sensing missions number doesn't significantly influence the power consumption when looking at a single node, of course the overall consumption throughout all sensor nodes is higher. This can be traced back to scalability of the approach in terms of communication effort. Because the main power draw of such systems is the wireless communication facility and reducing the communication effort also reduced, in turn, the power consumption.

In total, the experiment has confirmed the intuition previously described: SSM3 is the fastest in terms of set-up time and self-organization time. SSM2 the most accurate and SSM1 the good average. While exploring which types of sensing mission to use, also a sensing mission similar to the layout of SSM1was used but with no local classification on the leaf nodes and combining the feature values at the left leg node classifier. However the amount of data to be transmitted was to high. The network could not handle the load and the output buffer of the sensor node filled and rendered the sensor node useless and in turn the whole system broke down.

As for the part of the hypotheses, it has been proven successful as the sensor network is able to ensemble itself into three different sensing ensembles (SSM1, SSM2 and SSM3),

tree	SSM1	SSM2	SSM3
number of nodes	2	4	1
wireless connections	2	4	1
LIVE accuracy (%)	79	85	73
WEKA accuracy (%)	98.5	99.3	97.8
self-org. time (ms)	518	2395	75
time to data (ms)	2422	3132	2162
memory usage (KB) up to	350	363	309
comm (KB)	75	150	35
energy consumption (%)	$1-2$	$1-2$	1

Table 5.2: Comparison of different sensing missions

which can collect data and send them to the other sensor nodes for further processing and recognize the specified activities.

5.3 Self-Adaptation

When an activity recognition process is running in the wireless sensor network, it can be disturbed because single sensor nodes may disappear because of faults, e.g. battery ran out. Therefore it is necessary, that the sensor ensemble can deal with such problems.

The following parts of the hypotheses are to be proven:

a) Run-time operation of goal driven sensing ensembles can be protected using the self-organization technique against spontaneous and occasional sensor faults. Therefore we let a sensor ensemble perform a recognition task and simulate the disappearance and joining of sensor nodes by turning them off and on again, expecting that the activity recognition will continue and the sensor node will be reintegrated.

b) To affirm that the configuration of the sensor ensemble is derived by the sensing mission and only constrained by the body positions and not by a MAC address, a sensing mission's containment hierarchy only specifies which body parts data needs to be gathered from. When a sensor ensemble is performing a recognition task we replace a sensor node at a given body position with another sensor node with the same body position assigned to it, expecting again that the activity recognition will be continued and the new sensor node be integrated in the sensor ensemble.

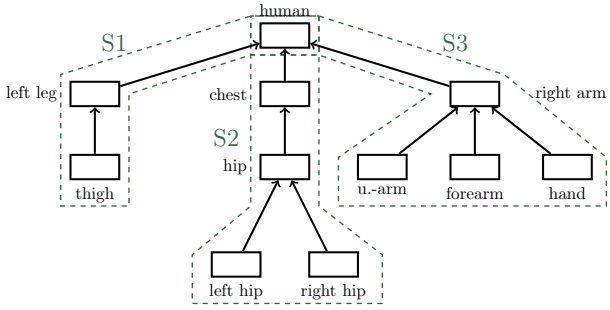


Figure 5.5: sensing mission

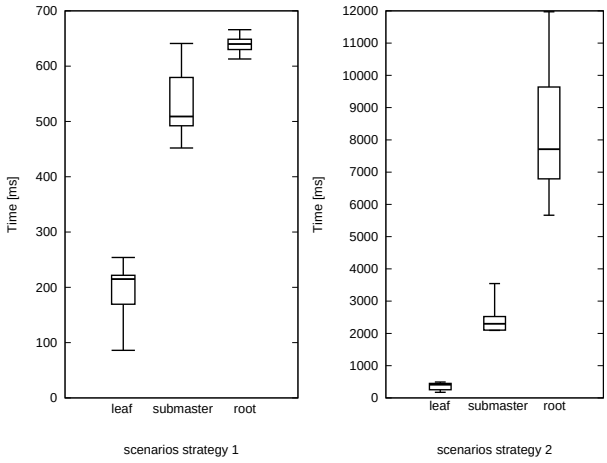
5.3.1 Experiment setup

The sensing mission, cf. Figure 5.5 was used to self-organize the sensor network and setup the activity recognition process. Denote that the sensor network consisted of 7 sensor nodes whereas two had the same body position assigned to them, to be precise the sensor node for the hand position had a backup sensor node.

5.3.2 Experiment implementation and conclusion

In order to show that the self-adaptation worked when running the activity recognition process for the strategy notification, the sensor nodes "hand" was restarted loosing all run-time information, including its knowledge of being part of the sensor ensemble running in the network. The sensor node assigned to the containment hierarchy node right arm detected the missing of information from hand and notified the sensor node accordingly using its MAC address. Upon receiving the notification the sensor node hand joined the sensor ensemble once more and delivered data again. The timespan from sending the notification until receiving the first data packet was measured 10 times and can be seen in, cf. Figure 5.6(a). The same procedure was applied for the sensor node assigned to "right arm", meaning the whole sub-tree S3 had to be notified to continue delivering information. This was also performed for the sensor node assigned to the root node of the containment hierarchy.

The self-adaptation strategy recreation was validated in a similar way, but unlike in the notification case, we did not restart the sensor nodes but replaced them by sensor nodes with the same body positions assigned to them. Therefore in the first case of the failed leaf node "hand", the node couldn't be notified and the sensor node assigned to the containment hierarchy's node right hand started a self-organization process for the failed network part. As we had a backup node in place, this sensor node satisfied the sensing mission for the failed part of the network and joined the sensor ensemble



(a) Notification (b) Recreation

Figure 5.6: Timings for self-adaptation

in turn. The time measured in figure 5.6(b) is the time it took for the self-organization process to complete and the first data from the replaced part to be received. The same procedure was repeated while the sensor node was also assigned to the right arm node (submaster) and the root node where in those cases the self-adaptation had to replace the S3 tree of the containment hierarchy and the whole sensing mission in the root case.

As for the parts of the hypotheses to be proven: ad a) It was proven that using self-organization techniques, a goal-driven opportunistic sensing ensemble can be protected against temporary sensor faults (restarted case) and permanent (replaced) case. ad b) The self-adaptation strategy recreation proved that the sensor ensemble only needs to be constrained by the body position specified within the sensing mission as a sensor node can be replaced by another sensor node with the same body position assigned to it.

5.4 Summary

With the three experiments performed, all hypotheses have been confirmed:

- The system is able to change its sensing goal at runtime and therefore the system can choose the sensors to perform an activity recognition task at runtime, based on the available sensors.

- The sensor system is only constrained by the sensing missions definition from which body parts data needs to be gathered, unlike, in traditional sensor systems, from which sensor.

- Each sensor node is able to communicate and interact with the environment by specifying its needs and capabilities. This enables self-organization and self-management because upon receiving a sensing mission the sensor node will try to satisfy either by itself or with the help of other sensor nodes. Furthermore communication channels will be opened and closed automatically based on the necessity of the data to be delivered bottom up from the sensors to the receiver of the recognition chain.

- A sensor ensemble is able to cope with the disappearance of sensor nodes, because of sensor faults, without disturbing the activity recognition by either reintegrating the same sensor if it reappeared or by replacing it entirely by a sensor with the same capabilities.

Chapter 6

Conclusion

In this thesis, **D**arSens, a concept, framework and the implementation for a distributed activity recognition system/algorithm has been presented. The system is able to access the capabilities of multiple sensor nodes, be it for data processing purpose or sensor data acquisition and use it to perform activity recognition.

DarSens is able to make best use of available sensor nodes and orchestrates them into an cooperative ensemble. The orchestration process takes into account the location and spatial relation between sensor nodes which in turn leads to the benefit that data to be transmitted will be sent to neighbours in close proximity first – which then will lead to sub-activity-recognition – until it finally reaches a central coordination point, to finalize the activity recognition, which in turn will deliver the outcome of such a multi-layered context recognition process to a final destination, which could be any application interested in context recognition.

In order to evaluate **D**arSens three experiment have been performed. The first experiment assess the frameworks capabilities of self-organization from a independent sensor network towards a coordinated cooperative activity recognition ensemble. The second experiment appraises how the specification for coordination of such an activity recognition process influences the systems performance in a typical activity recognition scenario. The third experiment shows how the system deals with sensor faults by self-adaptation without disturbing the activity recognition process.

6.1 Future Work

Currently there are two limitation with **D**arSens. The framework relies upon the sensor node knowing its position on the body it is attached to and induce the appropriate con-

tainment hierarchy for this location. However this conclusion is not made dynamically but is predefined and based on the MAC address of a sensor node. Although it can be changed at runtime using the **D**arSens EKG, cf. Appendix A. To fix this problem the best solution would be to integrate the approach to locate a sensor node on a human body from [19].

The second limitation is that currently the system will only work in a single person environment, where all sensor nodes are attached to the same person. Otherwise the self-organization process may create a sensor ensembles that use sensor nodes from different person and the recognition process would provide an application with wrong data. In order to overcome this issue the approach taken by [23] could be used.

To further improve the feasibility of the framework for application developers some of the task required to create a sensing mission should be automated.

- online-labeling facility while recording the data
- automatic retrieval of the recorded data of all sensor nodes wireless
- automatic feature extraction and feature selection
- automatic creation of multiple hierarchy linked classifier using the WEKA Toolkit or components of it
- automatic creation of the sensing missions data files (classification model, feature matricies, ...)

Furthermore in order to improve the activity recognition process it would be interesting to investigate the possibility of integrating user profiling. This would mean that the wearer of the sensor system would have the ability of updating and adjusting the classifier models while using it and in turn update the sensing mission for later usage. This would be an interesting feature for application developers as they can create very general models for the sensing missions that will work good in average over a lot of people and an application user can in turn adept and perfect the sensing mission for his own use.

A lot of activity recognition systems especially for long term monitoring work with the activity in relation time, as the human is not only a creature of habit and will perform the same activity either for a longer period in time but also e.g. at the same time on different days. Furthermore our physiology constrains us in so far as we are not able to change from one activity, we are performing, to another at will. There are

transition phases which can be used to further improve the activity recognition. E.g. it is unlikely for us to change from running to sitting without standing in between. This likelihood of a transition between activity can be modelled with a Hidden Markov Model (HMM). Therefore enhancing the system with such temporal models at any node in the containment hierarchy would be an effective way to improve the activity recognition performance.

Another possibility to further improve the system would be porting it to another hardware platform and operating system. Emphasized by its event driven architecture, TinyOS [10] would be a good alternative to the Sun Spot Platform used now. With this switch it would also be interesting to investigate if its possible to create a real-time system.

Another possibility would be to focus on a proper power management. Due to the layout of the containment hierarchy, it is clear that only two layers have to be powered on at a given time point in order to send and receive data. E.g. when the lowest level (leaf nodes) powers on their antenna to sending data to the next level, also the next layer will power on their radio for receiving this data. After reception the next layer powers on their antenna to receive the data from the second layer and so forth. Obviously this a very sophisticated synchronisation problem. However it would greatly lower the power consumption and make the system able to use for long-term activity recognition.

Bibliography

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a Better Understanding of Context and Context-Awareness. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, London, UK, 1999. Springer-Verlag.

[2] Ling Bao. Physical Activity Recognition from Acceleration Data under Semi-Naturalistic Conditions. Master's thesis, Massachusetts Institute of Technology, 2003.

[3] Ling Bao and Stephen S. Intille. Activity Recognition from User-Annotated Acceleration Data. In Alois Ferscha and Friedemann Mattern, editors, *Pervasive*, volume 3001 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2004.

[4] G. M. Bertolotti, A. Cristiani, R. Gandolfi, and R. Lombardi. A Portable System for Measuring Human Body Movement. *Digital Systems Design, Euromicro Symposium on*, 0:569–576, 2006.

[5] Driss Choujaa and Naranker Dulay. TRAcME: Temporal Activity Recognition Using Mobile Phone Data. *Embedded and Ubiquitous Computing, IEEE/IFIP International Conference on*, 1:119–126, 2008.

[6] Travis C. Collier and Charles Taylor. Self-organization in sensor networks. *J. Parallel Distrib. Comput.*, 64(7):866–873, 2004.

[7] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000.

[8] Elisabetta Farella, Augusto Pieracci, Luca Benini, and Andrea Acquaviva. A Wireless Body Area Sensor Network for Posture Detection. *Computers and Communications, IEEE Symposium on*, 0:454–459, 2006.

[9] Jonny Farringdon, Andrew J. Moore, Nancy Tilbury, James Church, and Pieter D. Biemond. Wearable Sensor Badge and Sensor Jacket for Context Awareness. *Wearable Computers, IEEE International Symposium*, 0:107, 1999.

[10] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC language: A holistic approach to networked embedded systems. *SIGPLAN Notes*, 38(5):1–11, 2003.

[11] Rahul Gupta, Sumeet Talwar, and Dharma P. Agrawal. Jini Home Networking: A Step toward Pervasive Computing. *IEEE Computer*, 35(8):34–40, 2002.

[12] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. *The WEKA Data Mining Software: An Update*, volume 11. ACM, 2009.

[13] Holger Harms, Oliver Amft, Gerhard Tröster, and Daniel Roggen. SMASH: a distributed sensing and processing garment for the classification of upper body postures. In *BodyNets '08: Proceedings of the ICST 3rd international conference on Body area networks*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[14] Clemens Holzmann and Michael Haslgrübler. A Self-Organizing Approach to Activity Recognition with Wireless Sensors. In *Proceedings of the 4th International Workshop on Self-Organizing Systems (IWSOS 2009)*, ETH Zurich, Switzerland, December 2009. Springer LNCS.

[15] Yu-Jin Hong, Ig-Jae Kim, Sang Chul Ahn, and Hyoung-Gon Kim. Activity Recognition Using Wearable Sensors for Elder Care. *Future Generation Communication and Networking*, 2:302–305, 2008.

[16] Do-Un Jeong, Se-Jin Kim, and Wan-Young Chung. Classification of Posture and Movement Using a 3-axis Accelerometer. *Convergence Information Technology, International Conference on*, 0:837–844, 2007.

[17] Holger Junker, Paul Lukowicz, and Gerhard Tr?ster. PadNET: Wearable Physical Activity Detection Network. *Wearable Computers, IEEE International Symposium*, 0:244, 2003.

[18] James F. Knight, Huw W. Bristow, Stamatina Anastopoulou, Chris Baber, Anthony Schwirtz, and Theodoros N. Arvanitis. Uses of accelerometer data collected

from a wearable system. *Personal Ubiquitous Comput.*, 11(2):117–132, 2007.

[19] K. Kunze, P. Lukowicz, H. Junker, and G. Tröster. Where am i: Recognizing on-body positions of wearable sensors. pages 264–275, 2005.

[20] Kai Kunze and Paul Lukowicz. Dealing with sensor displacement in motion-based onbody activity recognition systems. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pages 20–29, New York, NY, USA, 2008. ACM.

[21] Kai Kunze and Paul Lukowicz. Dealing with sensor displacement in motion-based onbody activity recognition systems. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pages 20–29, New York, NY, USA, 2008. ACM.

[22] Kristof Van Laerhoven and Hans-Werner Gellersen. Spine versus Porcupine: A Study in Distributed Wearable Activity Recognition. *Wearable Computers, IEEE International Symposium*, 0:142–149, 2004.

[23] J. Lester, B. Hannaford, and G. Borriello. "Are You with Me?"-Using Accelerometers to Determine If Two Devices Are Carried by the Same Person. *Pervasive Computing*, pages 33–50, 2004.

[24] Paul Lukowicz, Jamie A. Ward, Holger Junker, Mathias Stäger, Gerhard Tröster, Amin Atrash, and Thad Starner. Recognizing Workshop Activity Using Body Worn Microphones and Accelerometers. In *Pervasive Computing: Proceedings of the 2nd International Conference*, pages 18–22. Springer-Verlag Heidelberg: Lecture Notes in Computer Science, apr 2004.

[25] Uwe Maurer, Asim Smailagic, Daniel P. Siewiorek, and Michael Deisher. Activity Recognition and Monitoring Using Multiple Sensors on Different Body Positions. *Wearable and Implantable Body Sensor Networks, International Workshop on*, 0:113–116, 2006.

[26] Kevin L. Mills. A brief survey of self-organization in wireless sensor networks. *Wireless Communications and Mobile Computing*, 7(7):823–834, 2007.

[27] David Mizell. Using Gravity to Estimate Accelerometer Orientation. *Wearable Computers, IEEE International Symposium*, 0:252, 2003.

[28] Venet Osmani, Sasitharan Balasubramaniam, and Dmitri Botvich. Self-organising object networks using context zones for distributed activity recognition. In *BodyNets '07: Proceedings of the ICST 2nd international conference on Body area networks*, pages 1–9, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[29] Muhannad Quwaider and Subir Biswas. Body posture identification using hidden Markov model with a wearable sensor network. In *BodyNets '08: Proceedings of the ICST 3rd international conference on Body area networks*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[30] Nishkam Ravi, Nikhil D, Preetham Mysore, and Michael L. Littman. Activity recognition from accelerometer data. In *In Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence(IAAI*, pages 1541–1546. AAAI Press, 2005.

[31] T. Stiefmeier, D. Roggen, G. Troster, G. Ogris, and P. Lukowicz. Wearable Activity tracking in car manufacturing. *IEEE Pervasive Computing*, 7(2):42, 2008.

[32] Thomas Watteyne, Isabelle Augé-Blum, Mischa Dohler, and Dominique Barthel. AnyBody: a self-organization protocol for body area networks. In *BodyNets '07: Proceedings of the ICST 2nd international conference on Body area networks*, pages 1–7, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[33] Fen Zhu, Matt W. Mutka, and Lionel M. Ni. Service Discovery in Pervasive Computing Environments. *IEEE Pervasive Computing*, 4:81–90, 2005.

Appendix A

DarSens EKG

The **D**arSens EKG is mainly a demo application, which acts as a testbed for testing the different functions of the framework. It should be used to get a feeling for the concepts of the framework and its implementation and how to create an application.

A.1 Command

Using the command menu one can issue a lot of commands. e.g. SEND_INQUIRE_REQUEST (this is a broadcast to inquire which sensor nodes are out there, the list of sensor nodes will be shown in left side with their respective MAC address), ECHO_CAPABILITES (sends a direct message to a sensor node to obain his containtment hierachy; its necessary to select a MAC address from the list of sensor nodes beforehand)

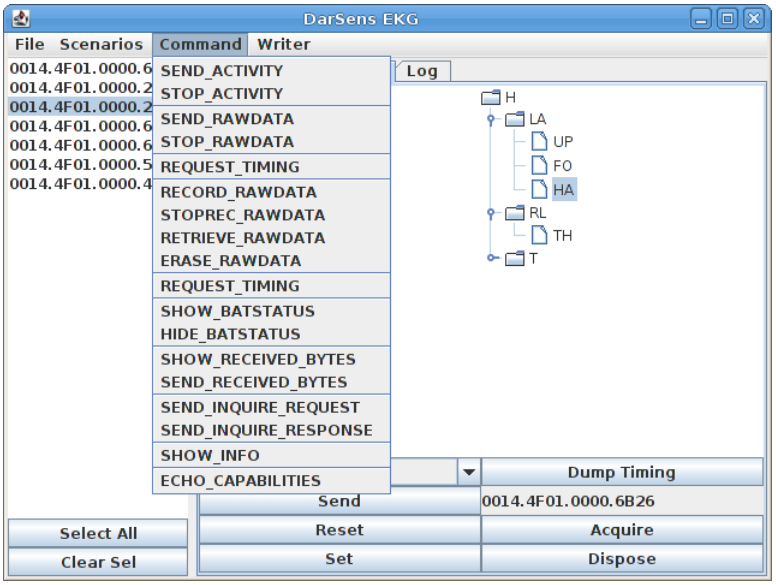


Figure A.1: Commands menu

A.2 Interaction

The interaction menu can be used for testing self-organization, network setup and activity recognition. E.g. to test self-organization one can use the selection box to choose from predefined sensing missions, which upon selection are shown as a tree (the left one). They can be broadcast using the send button. The response is then displayed in the right tree. Denote that selecting a node there, will reveal the MAC address assigned through the self-organization process.

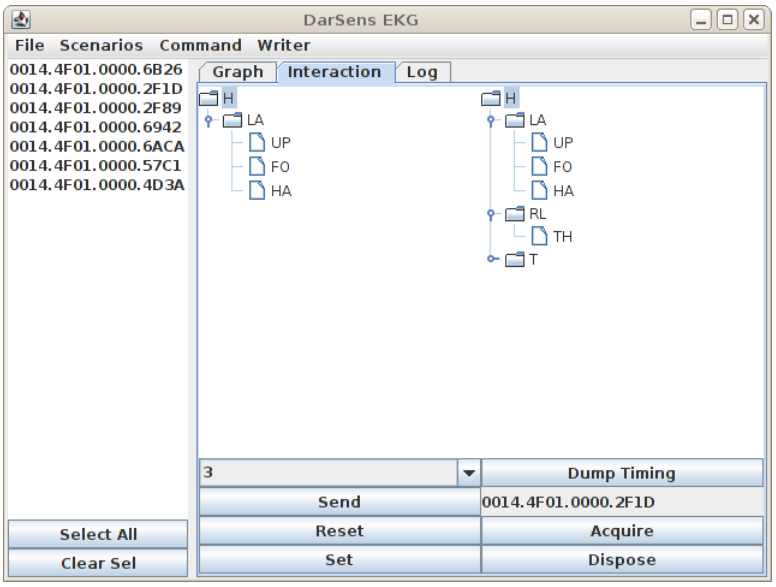


Figure A.2: Interaction/Self-Organization menu

A.3 Scenario

With the scenario menu one can reuse predefined satisfied sensing mission to start the activity recognition process.

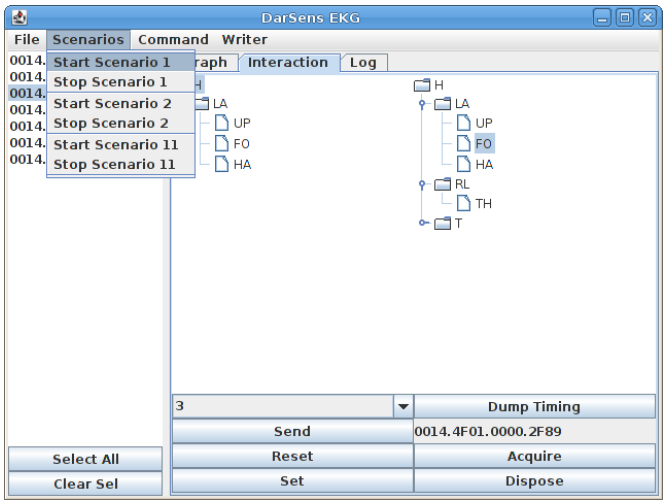


Figure A.3: Scenario menu

A.4 Visualisation

Using the SEND_RAWDATA command to a sensor node one can display the raw sensor data gathered by the accelerometer sensor. The gathered data is then displayed upon the graph panel.

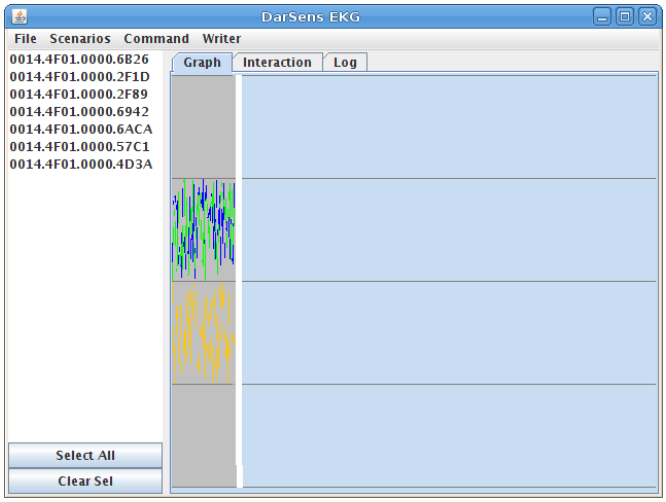


Figure A.4: Visualisation of raw sensor data