Medical Agents

Using Artificial Intelligence to Support Immediate Medical Care

Date

15th March 2016

Author

Brian van der Bijl 1540263

Examiners

Ing. Henk van Nimwegen (first examiner)

Ir. Joop Kaldeway (second examiner)

Company Supervisor

Dr. Leo van Moergestel

Utrecht University of Applied Science

Faculty of Nature and Technology

Dedication

I dedicate this to my mother Thecla van der Bijl–van Dillen, who unfortunately did not live to see this research completed, and to my uncle Wim de Bakker, who left this world due to a heart attack. It was this tragic event that inspired me to search for a way to apply my knowledge of computer science and AI to help make losses such as his avoidable.

Preface

This thesis marks the result of my study Technical Information Science (now part of HBO-ICT) at the University of Applied Sciences Utrecht. It is the final test for my Bachelor of Science degree.

Over the years I spent at this institute I have met a lot of people who have been an inspiration to me in one way or another. I would like to thank these people for everything they have taught me. I would especially like to thank Dr. Van Moergestel for his faith in my abilities to realise this thesis, even when I had my doubts myself, and for his guidance in plotting out my future career. I would also like to thank Ir. Kaldeway, my second examiner, for helping me keep on track when I tended towards getting lost in details and perfectionism. Outside the institute, I would like to thank Ruben Seggers, BSc. for introducing me to the subject of Machine Learning, proofreading the various documents this research project produced, and probably preventing me from turning them into a unreadable, horridly convoluted mess.

Finally, I would like to thank my friends and family for all the support I have been given during the execution of this research project, and for their understanding when I was unable to pay them the attention they deserve.

Executive Summary

Cause

The Hogeschool Utrecht Centre of Technology and Innovation is currently researching various applications of multi-agent technology. This research considers the posibilities of developing an agent-based system capable of monitoring vital functions and making predictions on a patient's health to facilitate early detection of imminent heart failure enabling faster response-times and preventing loss of quality adjusted life years.

Suggestions

Based on the experiences acquired in the development of a proof-of-concept medical multi-agent system, this research suggests such a solution is feasible; though a number of questions remain to be answered, further research by a larger, multidisciplinary team could continue the prototype's development into an extendable, adaptable product.

Motivation

As the prototype demonstrates, the modularity and extendability offered by a multi-agent solution present a promising base for the design of medical monitoring systems. By dividing the stated problem into three responsibilities (input, processing, output), an intelligent medical agent supported by a set of input (sensor) agent and output (communication) agents can use standard machine learning algorithms to make assessments on available data in a feature-agnostic way, greatly increasing reusability.

Perceived consequences

Further development of this project will require knowledge outside the field of computer science in addition to continued involvement of programmers, technicians and data scientists.

Glossary

ACL Agent Communication Language, a standardised language for agent communication developed by FIPA. 6, 31

BGFS Broyden-Fletcher-Goldfarb-Shanno algorithm: a optimisation algorithm used in machine learning to find the minimum of a cost function. 26

cardiac arrest sudden stop in blood circulation due to the failure of the heart to contract effectively or at all. 1

cost function A function mapping a possible value for θ to a numerical representation of its performance. 24–26

CVA cerebrovascular accident (brain attack): poor blood flow in the brain resulting in cell death. 1

decision boundary the hyper-surface that partitions the data-points into two sets. Points on one side belong to one category, points on the other side belong to the other. Points on the decision-boundary are ambiguous: both possible categories have a 0.5 probability. 14, 15, 24, 27, 45

Delaunay triangulation A triangulation algorithm where a field of points in a diagram is connected in such a way that no created simplex' circumcircle contains other points. 15

feature A single factor considered by the logistic regression algorithm in classifying an example, *e.g.* heart rate or diastolic blood pressure. 12, 14, 21–24, 27, 28, 30–32, 38, 44

feature mapping The process of generating additional features from existing ones, used to turn linear values into polynomials, thus allowing a more complex decision boundary. 21, 23, 27–29, 31

feature scaling The process of mapping a real value between a set minimum and maximum to a value in $(-1, 1)$ and vice versa. Used to ensure all

features are of the same weight when considered by logistic regression. 21, 38

FIPA Foundation for Intelligent Physical Agents: a foundation focused on the developing standards for agent systems. 6

hypothesis function A function combining the vector θ, found using the training algorithm, with a vector x, containing a set of observed features, to calculate a prediction. 23, 24, 27, 45, 47

JADE JAVA Agent DEvelopment Framework: a Java based development framework for FIPA based agents developed by Telecom Italia in 1998. 6, 12, 29, 37, 41, 42

locus For two vertices, the set of points equidistant from either vertex. 17

logistic regression A classification algorithm used to divide observations either inside or outside of a category. Outputs a value in $(0, 1)$ representing the probability that the observation falls into the category. 21, 23, 24

MAS multi-agent system: a system composed of multiple intelligent agents interacting within an environment. 2, 4–7, 11, 29, 31, 37, 39–44, 47

MI myocardial infarction (heart attack): cessation of blood flow resulting in damage to the heart muscle. 1

overfitting An undesirable situation where a hypothesis function becomes over-specific to the training data and fails to generalise to new observations. 24, 27, 47

QALY quality-adjusted life years. 1, 2, 42

sigmoid function A function used in logistic regression to convert a real number into a value in $(0, 1)$. Defined as $g(z) = \frac{1}{1+e^{-z}}$. 23, 24

simpex The n-dimensional generalisation of a triangle, tetrahedron, etc. The most basic n-dimensional object defined by $n+1$ points in n-dimensional space. 14–18, 45

training example A set of features collectively representing a single observation used for training the logistic regression algorithm. 23, 24, 27

underfitting An undesirable situation where a hypothesis function fails to develop a good fit to the data. 27

vertex A point serving as a corner for a n-dimensional shape. 14, 15, 17, 19, 20, 45

XML eXtensible Markup Language, a document format developed by W3C designed to be readable to both humans and machines. 6, 31, 34, 38, 44

Contents

1 Introduction

The Hogeschool Utrecht Centre of Technology and Innovation is currently researching various applications of multi-agent technology. My research suggests a new application of this interesting field of technology, applying lessons learned to medical support. Multi-agent technology studies the utilisation of groups of intelligent agents to autonomously perform complex tasks. The modularity and expandability offered by this approach could benefit medical support of patients, with limited impact on their daily lives as agents living inside a compact device could monitor vital statistics and respond when necessary, thereby reducing the need for conventional medical surveillance. This study explores these possibilities and marks the start of a larger project towards realising better ways of ensuring people's health.

1.1 Problem to be addressed

As medical science progresses, many debilitating and potentially fatal conditions are becoming easily preventable in the presence of first-world health care. The success of treatment largely depends on quick action being taken: as more time passes before adequate care is provided, the chance of recovery quickly decreases and the loss of quality-adjusted life years (QALY) increases as lack of oxygen causes brain cells to permanently die. Acute cardiovascular diseases[i] are among the most prevalent diseases in the western world: In the Netherlands alone, approximately 15 000 people suffer cardiac arrest outside of hospitals each year (Nederlandse Hartstichting, n.d.-b). In 2012, 29 000 people suffered a Myocardial Infarction (MI) and 44 000 people were affected by a Cerebrovascular accident (CVA) (Nederlandse Hartstichting, n.d.-a). All of these show a direct correlation between the time to onset of treatment and QALY in victims.

[i] Brain- and heart attacks and related afflictions.

1.2 Proposed solution (hypothesis)

This study aims to explore the possibilities of making monitoring vital functions in "high-risk" individuals[ii] more accessible and affordable by utilising agent technology to create cheap and replaceable dedicated monitors. By proposing a "medical agent", this study aims to decrease response-time leading to increased QALY following an attack. Furthermore, a "medical agent" could potentially allow people to live independently when without it they would be forced to rely on a retirement home to provide constant attention. This would reduce both the costs on society as well as leading to better psychological health (World Health Organisation, 2013) in patients.

To solve this problem, this research proposes a medical agent or medical Multi-Agent System (MAS) programmed to monitor relevant vital functions and use its knowledge of the patients situation to anticipate emergencies, communicating them to the patient, caregivers and/or medical authority. This research focuses on the above-described cardiovascular diseases, but should in no means be limited to this kind of ailment. The proposed model could be extended to watch for a variety of problems, *e.g.* pulmonary embolism, acute dyspnea, (chronic) obstructive pulmonary disease, asthma exacerbation, epilepsy and diabetesmellitus. For the sake of manageability, this research will address a single area. Should the results be as expected, further research could be initiated to expand the scope. A medical agent should be, by design, extendable: additional sensors and knowledge could be incorporated into the framework using a common architecture to address different threats using a single core design.

1.3 Research questions

The main question this research will aim to answer can be summarised as "**How could agent technology contribute to increase the number of quality-adjusted life years (QALY) in acute cardiovascular disease?**". To answer this question, several sub-questions will need to be asked:

RQ1 Which biological factors are indicative of imminent acute cardiovascular disease?

[ii] Those with an elevated risk of acute cardiovascular diseases, *e.g.* elderly and those with a history of such attacks.

RQ2 How could these factors be discreetly monitored?

RQ3 Considering the results of RQ1 and RQ2, what would be the most appropriate design for a medical agent?

RQ4 How can a proposed medical agent be trained to recognise and respond to alarming measurements?

RQ5 What would be the requirements in hardware and software of such an agent?

RQ6 What is the relation between the product agent suggested by Van Moergestel et al. and the medical agent suggested in this research?

RQ7 How is important patient information stored and communicated in current medical care?

RQ8 How could a "medical agent" be securely linked to / incorporated into existing systems in medical care?

RQ9 How could this agent be tested to ensure sufficient reliability?

I have endeavoured to answer these questions by means of a literature study and development of a prototype, as documented in the following chapters. Table 1.1 presents an overview of the chapters in relation to the research questions. The order of the questions does not conform to the order of the chapters. I chose to present my work in this order because I believe it should be easier for my readers to follow. For example, Chapter 2 deals with the overall design of the system, introducing separation of responsibilities. It makes sense to discuss this aspect before introducing specific agents starting in Chapter 3.

Chapter	RQ1	RQ2	RQ3	RQ4	RQ5	RQ6	RQ7	RQ8	RQ9
1									
2			✓						
3	✓	✓							
4				✓					
5							✓	✓	
6					✓				
7									✓
8						✓			
9	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1.1: Relation between research questions and chapters

2 Design

As the medical agent should be designed to be as modular as possible, implementation as a *distributed problem solving MAS* (Wooldridge, Jennings & Kinny, 2000) seems a logical choice — even though this does make the term "medical agent" somewhat imprecise. Implementing the medical monitor as a MAS allows for the division of the product's responsibilities between specialised agents, creating a central agent, responsible for the interpretation of observations, and separate, less complex agents for tasks related to input and output. In a medical MAS, three sets of responsibilities could be distinguished, each corresponding to a distinct category of agents:

- Sensor monitoring,
- decision making and
- outside communication.

Each MAS could contain any number of agents from the first and third categories (collectively known as utility agents), as well as one central agent capable of mapping observations from sensor agents to actions performed by communication agents. This design allows for agents to be added and removed dynamically while keeping core functionality intact. An example medical MAS is shown in Figure 2.1, including a number of potential utility agents.

2.1 Sensor-monitoring agents

Sensor-monitoring agents are the simplest and most diverse agents in the medical MAS. These agents exist primarily to support modularity: As sensors do not necessarily produce compatible signals to communicate their results, a small, dedicated agent could be programmed to read the sensor-data and communicate it to the decision-making agent in standard format. This way, the decision-making agent need not know the way measurements are performed. A sensor could easily be exchanged for a different kind of sensor, together with its associated agent, without necessitating mayor changes to the decision-making

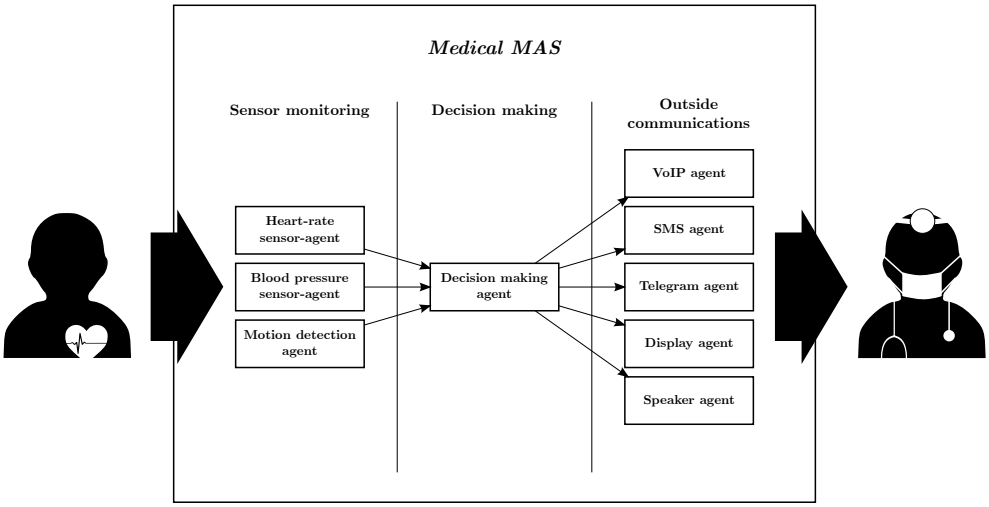


Figure 2.1: The three types of agents.

agent. Similarly, new sensors could be added to facilitate patients developing new diseases. Though this would require new data to be added to the decision-making agent, the process of acquiring the measurements could easily be added by inserting a new sensor-monitoring agent to the MAS. This part of the system will be discussed in Chapter 3.

2.2 Decision-making agent

The decision-making agent operates in the middle layer of the medical MAS architecture. This is where most of the complex mathematics take place. This part of the MAS will be discussed in detail in Chapter 4.

2.3 Communications agent

Just as there are various ways to acquire data to facilitate the decision-making agent, there are also myriad methods to communicate its results. These include telephony, instant messaging, patient-information logs and on-device IO[i]. The preferred methods of communication may be subject to change over time as the patient's situation changes, as doctors come and go and as new forms of communication are developed, become widely adopted and are eventually deprecated. Therefore, MAS communication to the outside world should be

[i] displays, alarms, etc.

modular. Just like with sensor-reading agents described in section 2.1, methods of communications could be implemented by small, trivial single-purpose agents. By abstracting the way information is delivered, the decision-making agent can communicate its results in a predefined manner indicating the conclusions to be sent and the perceived level of panic. Communications agents can pick up these messages and assume responsibility of relaying the information to the appropriate recipients. These agents relate to the seventh and eighth research question and will be described in Chapter 5.

2.4 Internal communication

The above described categories of agents will operate together in a single MAS. Though it would be possible to have agents running outside this MAS and still be able to communicate with the agent within, this aspect falls outside of the scope of this research and will be briefly discussed as future work in Chapter 9.4.

The gold standard for agent development is the JAVA Agent DEvelopment Framework (JADE) developed by Telecom Italia. It provides a set of tools and an extendable framework for creating agents and MAS' using the Foundation for Intelligent Physical Agents (FIPA) standard. Within JADE, agents exchange information using the Agent Communication Language (ACL) protocol (Foundation for Intelligent Physical Agents, 2002) developed by FIPA. The prototype medical MAS has been developed in JADE and as such uses the ACL protocol for communicating information. The ACL protocol allows for a number of languages to encapsulate data, including eXtensible Markup Language (XML); as the XML format provides a means to represent data in a semantic way that is readable to both humans and computers it appears to be a good choice for inter-agent communication. A typical exchange of messages is shown in Figure 2.2. Two sensor-agents send a stream of measurements to the decision agent, which periodically calculates its assessment of the situation. When the assessment reaches a certain defined threshold, it starts to send messages to the communication agents. The communication agents can respond with a confirmation or a message indicating either failure to relay the communication or failure to understand the message. If the decision-agent receives a message indicating failure, it tries to alert an operator using the designated fallback.

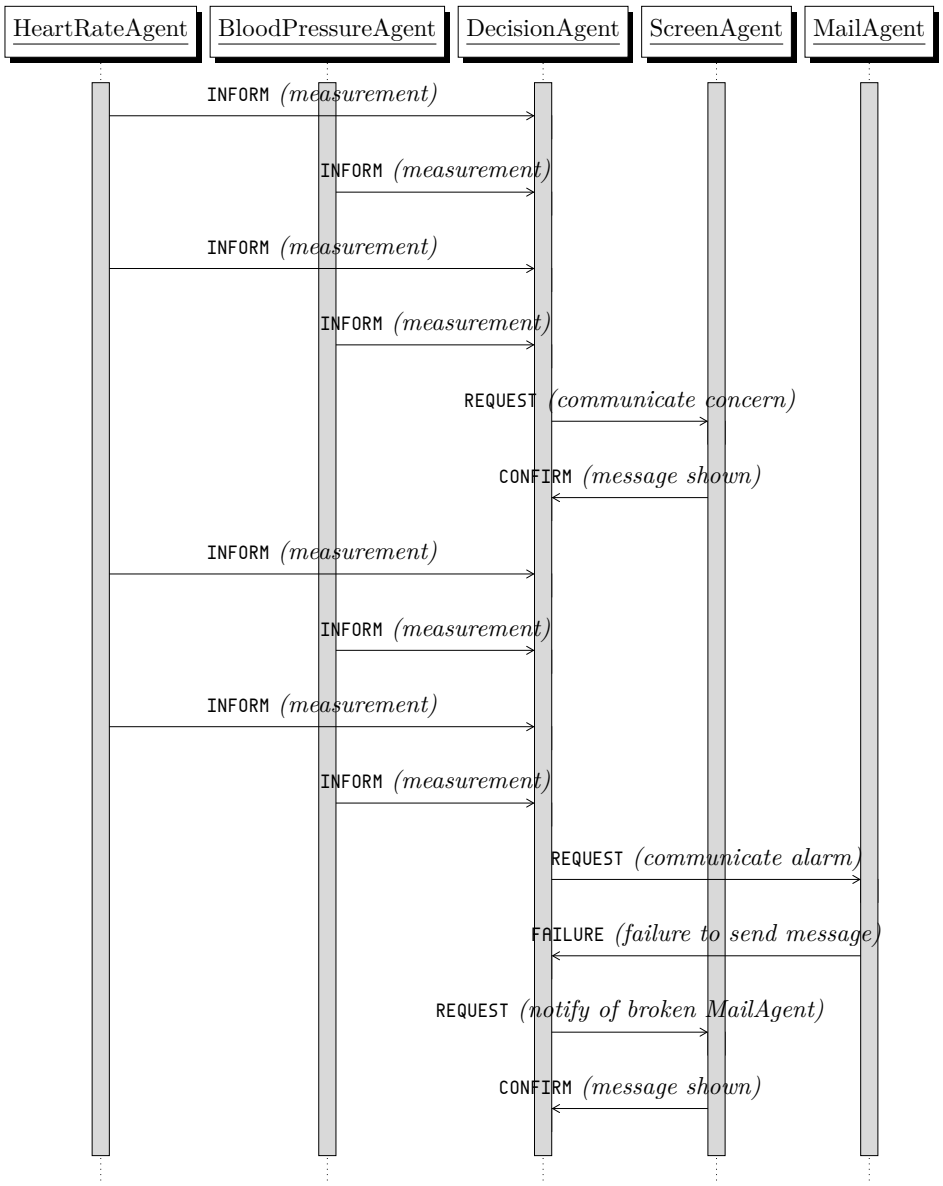


Figure 2.2: Message exchange within the medical MAS.

3 Biological factors and monitoring

In order for any system to be able to assess a patients health, it needs to know about a number of biological factors. For heart failure and related problems — the focus of this research, the patients heart rate and blood pressure seem to be obvious choices for factors to be monitored. For other diagnoses another set of variables might contain more meaningful information. However, as a computer scientist focused on artificial intelligence (AI), I am unqualified to provide a decisive list of factors to be considered. As such, after consultation with my company supervisor, I chose instead to design my prototype to be as factor-agnostic as possible. Each factor to be considered is added during initialisation as a feature to a n-feature algorithm, and an appropriate sensor is added to the system to collect the necessary data. By scaling each feature to a value in the range $(-1, 1)$, as described in Section 4.1.8, the specifics of each feature are abstracted away from the reasoning process: the product does not need to know what a given value represents, only how it affects the output of its prediction-function.

Thus, for the system to work with a given set of variables, three things are needed to add the desired behaviour to the existing product:

- A sensor capable of measuring the new feature,
- a mapping between sensor-output and a value in the range $(-1, 1)$ and
- a dataset for the prediction algorithm featuring the new factor[i].

To facilitate the development of sensor-agents, an abstract class has been written to minimise the required amount of boilerplate code. All sensor-agents have a similar structure: a single, repeating behaviour and a standardised method of communicating measurements to the decision-making agent. The

[i] the prototype developed as part of this research includes a way to accomplish this; see Chapter 4.

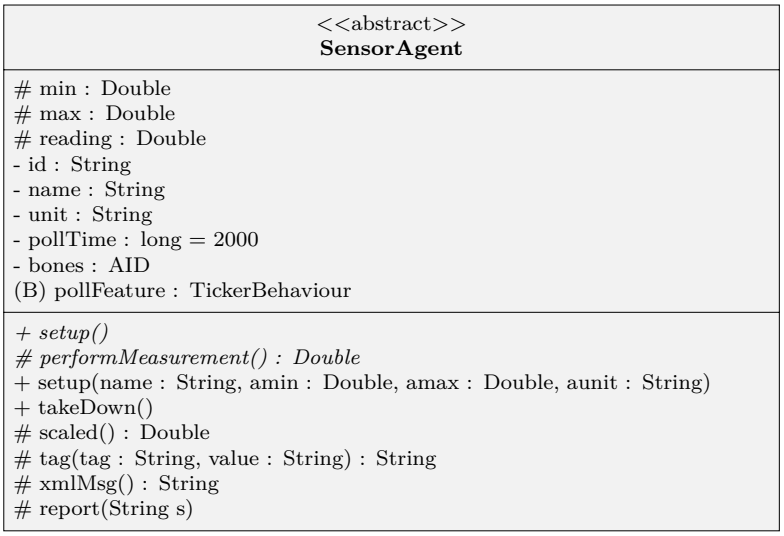


Figure 3.1: The SensorAgent abstract class.

`SensorAgent` class, as shown in Figure 3.1, provides an abstraction of these similarities and only needs a `setup()` method and a `performMeasurement()` method to be implemented to create a concrete class. The `setup()` method should set any relevant variables (of which the variables required by the `SensorAgent` class can be set by calling the `setup(String, Double, Double, String)` methods of the superclass). The `performMeasurement()` method should contain any code needed to perform a measurement, and return its results as a `Double`. The default behaviour of the `SensorAgent` is to execute the measurement-method every 2[ii] seconds and to send the measurement to the decision-making agent using an XML message as shown in Listing 3.1. In this example, a subclass called `HeartRateAgent` sends a message containing both the raw values of the measurement (a real number within the range supported by the sensor) and the same measurement scaled to the range of $(-1, 1)$.

```
<measurement>
  <feature>HeartRate</feature>
  <raw>68.950161</raw>
  <scaled>-0.080665</scaled>
</measurement>
```

Listing 3.1: A typical message sent from `HeartRateAgent`

[ii] This frequency can be configured in the variable `pollTime` which is used by the `pollFeature` behaviour.

The `SensorAgent` itself contains a number of private and protected attributes to store its specifics such as the name it uses for communication, the specifics of the associated sensor such as minimum / maximum / current values, and the Agent ID of the decision agent[iii]. A `SensorAgent` contains a single behaviour, represent here as (B). As UML was not designed for representing agents and various "Agent UML's" are still in their early stages, I chose this representation as an extension to UML.

As described above, the `SensorAgent` contains two abstract functions needed to implement a subclass. In addition, the class contains a few helper-methods to limit code duplication. `scaled()` uses the minimum and maximum values to apply feature scaling (described in more detail in Section 4.1.8), `tag(String, String)` and `xmlMsg()` provide a more readable way to generate the XML and `report(String)` is a wrapper around `System.out.println(String)` prepending the output with the agent name to make it easier to distinguish messages when various agents are reporting at the same time.

3.1 HeartRateAgent

As an example, Listing 3.2 provides a template for how a `HeartRateAgent` would look as a subclass of `SensorAgent`. The hardware-specific code can be filled in when a sensor has been selected to produce a functional sensor-agent.

```
1  public class HeartRateAgent extends SensorAgent
2  { public void setup()
3    { super.setup("HeartRate", 0.0, 150.0, "bpm"); }
4
5    public Double performMeasurement()
6    { try
7      { // read sensor
8        // calculate heart rate from reading
9        return reading; } // feature scaling happens when a message is sent.
10     catch (Exception e)
11     { // handle exception
12       return reading; } } } // return the old reading
```

Listing 3.2: A sample SensorAgent subclass

[iii] The name "bones", as seen in Figure 3.1, is short for sawbones, signifying its status a as crude prototype of an actual medical expert.

3.2 Monitoring features

As there is at present no conclusive answer as to which features to monitor, the process of monitoring these specific factors is unknown as well. However, as recent advances in this area allow even laymen to measure an increasing number of statistics regarding their health using only a smartphone, it is unlikely that a lack of options for feature-monitoring will present a serious challenge in the development of a usable medical MAS. As accessible eHealth applications become more and more ubiquitous (Satchwell, n.d.), it can safely be concluded a medical MAS would benefit from theses advances as well. For now, the `SensorAgent`-class contains the boilerplate on which any future sensor monitoring agent can be based.

The `SensorAgent`-class also forms the basis for a number of "mock agents" designed to simulate measurements using small random increments and decrements to an average value. These mock agents obviously do not represent true sensors, but facilitate testing of various subsystems in the decision agent. For the purpose of easily testing the decision agent's behaviour in case of emergencies, a special mock `HeartFailureAgent` has also been created simulating a rapidly decaying heart rate.

4 Agent Reasoning

This chapter is divided into three parts. The first section focuses on the process of gathering data regarding the chosen features by querying an expert about the expected predictions for a set of measurement-combinations. It also introduces an intelligent way to suggest which data points to query based on existing knowledge. The second part describes the learning algorithm used to formulate a mathematical representation of the knowledge gathered in the first section. Finally, in the third section a decision-making agent will be explored to use the gathered knowledge to make predictions. The first and second sections together describe the methods used to get to train the agent; this process has been implemented in a prototype using R, a modern language designed for numerical analysis well suited for this kind of task. As the result is mathematical vector, it can easily be used in the Java / JADE based agent as described in the last section. When choosing R for the learning process I also took into account the existence of Shiny, a library used to provide a web-interface for R-based applications. Though this library is not used in the prototype, it should be an asset in facilitating a more user-friendly interface in a future version.

4.1 Data Entry

In order to interpret the measurements acquired from the sensors and predict whether the current patient situation constitutes a cause of alarm, the decision-making agent needs a way to classify potentially high-dimensional[i] data. As this information is not guaranteed to be available for various combinations of biological features, it makes sense to explore a way for medical personnel to easily enter such data into the system. Not only does this guarantee the required data can be generated, if not available, it also allows for far greater personalisation, providing the agent with a data-set tailored to its

[i] Each biological factor considered in the model represents an additional dimension for data points.

patient. Manual entry, or at least confirmation, also allows an expert intimate knowledge of the agents decision-making process, potentially increasing trust by removing the "black box" aspect of machine learning.

Teaching the system to recognise alarming measurements and differentiate between various levels of threat requires large amounts of information provided by medical personnel, preferably tailored to the patient as thresholds might not be the same for every person. Entering this data can be challenging: as potentially multiple factors need to be taken into account together, it becomes progressively harder for humans to visualise and communicate relevant thresholds. A better way might be to input a set of data-points, together with appropriate assessments of the situation associated with each data-point. These data-points could be used, alone or in conjunction with more general datasets, to train a classification algorithm.

In order to train an agent to make accurate predictions, training data will need to be entered into the system by a medical expert. This should be as easy as possible: the focus should be to quickly train an agent without expending significant time accommodating the system. Unfortunately, entering possibly poly-dimensional data graphically is a difficult task. For one or two dimensional data, clicking points in a scatter plot, as pictured in Figure 4.1, can be a quick way to enter points; for three dimensional data this becomes

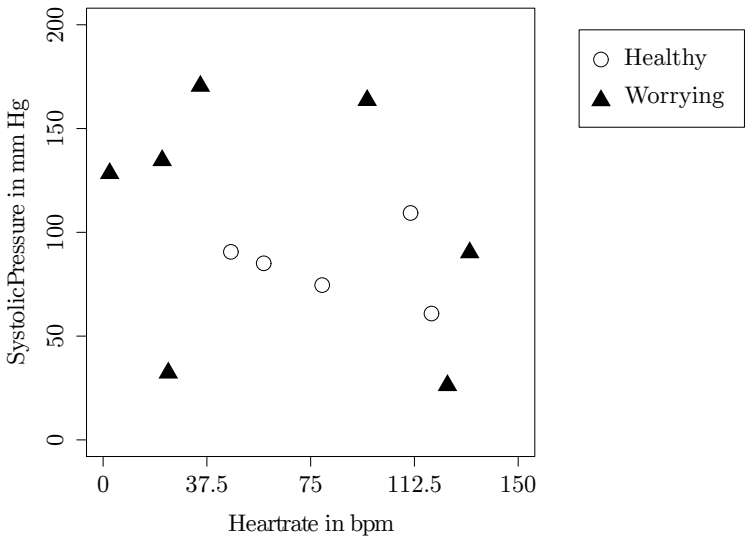


Figure 4.1: Scatter plot in two dimensions of a small random dataset.

harder: a scatter-plot is still possible for data-visualisation, but entry becomes impossible as a mouse or trackpad and a computer screen are both essentially two-dimensional. For even more simultaneous features, only a subset of the features can be plotted at the same time.

An alternative approach would be to require the expert to manually enter all features, as well as the results that the system should predict. Not only is this rather work-intensive, but also prone to omissions: as it is hard for the human mind to visualise all features simultaneously large gaps are a significant risk.

A better solution would be for the system to dynamically suggest data-points based on the largest knowledge gaps. This section considers an approach to accomplish this. Each problem will be examined in two dimensions first, as this makes it easier to visualise and demonstrate the applied methods. After the solution has been sufficiently exposed a generalisation will be made in n-dimensions. Triangles and their higher-dimensional analogues (the tetrahedron in three dimensions, the 5-cell in four, etc.) are collectively referred to as n-simplices or just simplices (singular: simpex). As a triangle (2-simpex) is defined by three vertices of the form (x, y) and and a tetrahedron (3-simpex) is defined by four vertices of the form (x, y, z), a n-simpex is the most basic n-dimensional object defined by $n + 1$ vertices in n-dimensional space.

4.1.1 Finding the most valuable points for data-querying

When entering data-points to train an agent, some points are more valuable than others. For example, potential locations completely surrounded by existing data-points all belonging to the same class are unlikely to add any new information to the system. Similarly, points in sparse areas are potentially more valuable, as are points closer to the centre of the point cloud. Figure 4.2 shows the same scatter plot as Figure 4.1, but adds a decision boundary [ii] and three possible locations for new data points marked by numbers. Location 1 does not appear to be a good addition, as it is very close to existing points and is therefore unlikely to add a great deal of information. Location 2 is not a good suggestion either, as it is very far from the decision boundary — it will likely have the same category as the points surrounding it, especially if a large amount of data has been entered. Location 3 is a better spot for a new data point: it is not a near duplicate of another point, and it lies close

[ii] The line or multi-dimensional equivalent separating the set of points into two categories.

to the decision boundary. Depending on the category this point will be assigned to it may significantly change the decision boundary in either direction.

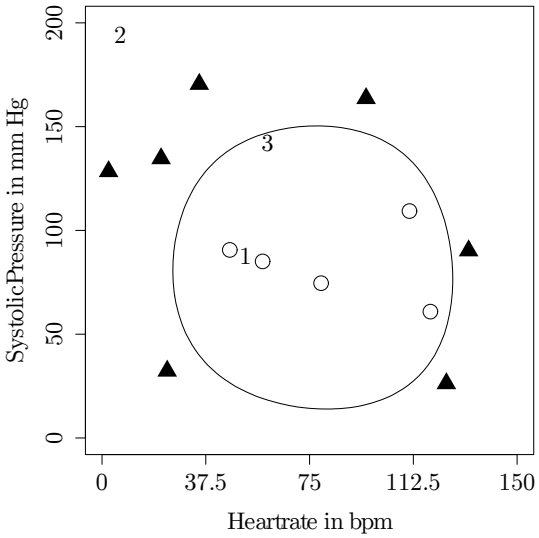


Figure 4.2: Three possible locations for a new data-point.

4.1.2 Data-point-distribution

To find sparsely populated areas to add new data-points, we first create a triangulation containing all data points. For each of these triangles, the circumcentre is calculated, and the collection is ordered based on the area of the triangles. These points can now be evaluated in order to find points close to the current decision-boundary.

4.1.3 Triangulating n-dimensional space in simplices

To triangulate a set of points we utilise the Delaunay triangulation (Eric W. Weisstein, 2002b). Most mathematical libraries include a function to quickly get the Delaunay Triangulation of a set of points in n dimensions. Triangulating the example data from Figure 4.1 yields the triangulation as shown in Figure 4.3.

4.1.4 Calculating the size of each n-simplex

To find the largest simpex we use the determinant of the matrix constructed by adding each vector representing a vertex as a single column, and adding a final row of ones (Stein, 1966). For a triangle, the absolute value of the result is

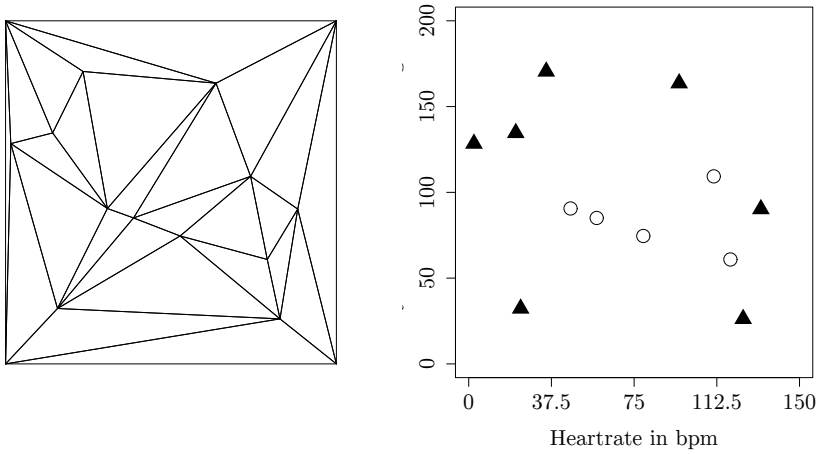


Figure 4.3: Triangulation and scatter plot in two dimensions

equal to two factorial times the triangle's area. For a tetrahedron, the absolute value equals three factorial times the volume. For higher-dimensional shapes, this method continues to yield a scalar multiple of the n-hypervolume of the simpex. As the simpex' size is only used for sorting, the scalar multiplication does not influence the ordering and can safely be ignored. As an example, the size of a triangle described by $a = (0, 0)$, $b = (0, 4)$ and $c = (3, 0)$ is given by

$$\begin{aligned}
\operatorname{abs}\left(\begin{vmatrix} 0 & 0 & 3 \\ 0 & 4 & 0 \\ 1 & 1 & 1 \end{vmatrix}\right) &= \operatorname{abs}\left(0\begin{vmatrix} 4 & 0 \\ 1 & 1 \end{vmatrix} - 0\begin{vmatrix} 0 & 0 \\ 1 & 1 \end{vmatrix} + 3\begin{vmatrix} 0 & 4 \\ 1 & 1 \end{vmatrix}\right) \\
&= \operatorname{abs}(3 \cdot 0 \cdot 1 - 3 \cdot 4 \cdot 1) \\
&= \operatorname{abs}(-12) \\
&= 12,
\end{aligned}$$

which is twice the area of the triangle. Listing 4.1 shows an implementation of the area function in R.

```
area ← function (X)
{ X ← cbind(t(X), 1)
  abs(det(X)) }
```

Listing 4.1: Area function implemented in R.

4.1.5 Calculating the circumcentre of each n-simplex

Once the largest data-gap has been found, we want to find its centre to suggest as a new data point. A simpex has multiple definitions of its centre; for this purpose the circumcentre, the point equidistant from all its vertices (Eric W. Weisstein, 2002a), seems a logical choice. Given a n-simpex defined by vertices $\boldsymbol{v^{(1)}}, \boldsymbol{v^{(2)}}, \dots \boldsymbol{v^{(n+1)}}$ with a circumcentre c, we know that the distance between any vertex and c must, by definition, be equal. For any two vertices $\boldsymbol{v^{(a)}}$ and $\boldsymbol{v^{(b)}}$, this means:

$$\|\boldsymbol{v^{(a)}} - \boldsymbol{c}\| = \|\boldsymbol{v^{(b)}} - \boldsymbol{c}\|$$
$$\|\boldsymbol{v^{(a)}} - \boldsymbol{c}\|^2 = \|\boldsymbol{v^{(b)}} - \boldsymbol{c}\|^2$$
$$(\boldsymbol{v^{(a)}} - \boldsymbol{c}) \cdot (\boldsymbol{v^{(a)}} - \boldsymbol{c}) = (\boldsymbol{v^{(b)}} - \boldsymbol{c}) \cdot (\boldsymbol{v^{(b)}} - \boldsymbol{c})$$

We translate each vector by $-\boldsymbol{v^{(1)}}$ so that $\boldsymbol{v^{(1)}}$ becomes the origin (denoted $\boldsymbol{o}$) and equate the distance to $\boldsymbol{c}$ of each remaining vector with the distance of $\boldsymbol{c}$ to $\boldsymbol{o}$, yielding the locus for each translated vertex $\boldsymbol{v}$ and the origin $\boldsymbol{o}$:

$$(\boldsymbol{o} - \boldsymbol{c}) \cdot (\boldsymbol{o} - \boldsymbol{c}) = (\boldsymbol{v} - \boldsymbol{c}) \cdot (\boldsymbol{v} - \boldsymbol{c})$$
$$\boldsymbol{c}^2 = \boldsymbol{v}^2 - 2\boldsymbol{v} \cdot \boldsymbol{c} + \boldsymbol{c}^2$$
$$2\boldsymbol{v} \cdot \boldsymbol{c} = \boldsymbol{v}^2$$
$$\boldsymbol{v} \cdot \boldsymbol{c} = 0.5\boldsymbol{v}^2$$
$$\boldsymbol{v}_1\boldsymbol{c}_1 + \boldsymbol{v}_2\boldsymbol{c}_2 + \dots + \boldsymbol{v}_n\boldsymbol{c}_n = 0.5\|v\|^2$$

Doing this for every vertex $\boldsymbol{v^{(2)}}$ to $\boldsymbol{v^{(n+1)}}$ gives us n equations, allowing us to find the n-dimensional vector $\boldsymbol{c}$. We can write these equations in matrix form and solve all equations simultaneously:

Writing

$$S = \begin{pmatrix} v_1^{(2)} - v_1^{(1)} & v_2^{(2)} - v_2^{(1)} & \dots & v_n^{(2)} - v_n^{(1)} \\ v_1^{(3)} - v_1^{(1)} & v_2^{(3)} - v_2^{(1)} & \dots & v_n^{(3)} - v_n^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ v_1^{(n+1)} - v_1^{(1)} & v_2^{(n+1)} - v_2^{(1)} & \dots & v_n^{(n+1)} - v_n^{(1)} \end{pmatrix}$$

$$\boldsymbol{c} = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} \quad \boldsymbol{r} = 0.5 \begin{pmatrix} \|v^{(2)} - v^{(1)}\|^2 \\ \|v^{(3)} - v^{(1)}\|^2 \\ \vdots \\ \|v^{(n+1)} - v^{(1)}\|^2 \end{pmatrix},$$

we have

$$S\boldsymbol{c} = \boldsymbol{r} \ .$$

Given this, we can multiply both sides by S^{-1} to get

$$\boldsymbol{c} = S^{-1}\boldsymbol{r} \ .$$

As c was translated by $-v^{(1)}$, all that remains is adding $v^{(1)}$ to find the triangle's circumcentre. The `circumcentre(X)` function is implemented in R as shown in Listing 4.2.

```
circumcentre ← function (X)
{ origin ← X[,1]
  vectors ← X[,2:ncol(X)] − origin
  rhs ← diag(0.5 * t(vectors) %*% vectors)
  c ← ginv(t(vectors)) %*% rhs
  origin+c }
```

Listing 4.2: Circumcentre function implemented in R.

4.1.6 Avoiding suggesting out-of-bounds points

As shown in Figure 4.3, Delaunay triangulations are prone to yielding obtuse simplices, in particular around the edges. This can be a problem because an obtuse simpex has a circumcentre outside itself. On the edges, this will result in the algorithm suggesting points outside the sensor's bounds. As these points are meaningless and only serve to distract the user, we would like to avoid generating obtuse simplices.

We solve this problem by introducing a border of false data-points around the edge. These data-points are only used to determine the Delaunay triangulation, and are not present in the actual training-data being generated. The number of data-points is determined by a variable $\beta \in \mathbb{N}_1$: For $\beta = 1$, only the corners of the graph are added. For larger values of β, each axis is subdivided into β parts. As β becomes larger, out-of-bounds points become increasingly unlikely, and suggestions start to gravitate towards existing data-points.

As Figures 4.4 and 4.5 show, too large a value for β makes the algorithm increasingly unlikely to suggest points around the edges. Though more central points are preferred, limiting data-points to a central cluster might not be the way to go. A solution for this could be to gradually decrease beta over time.

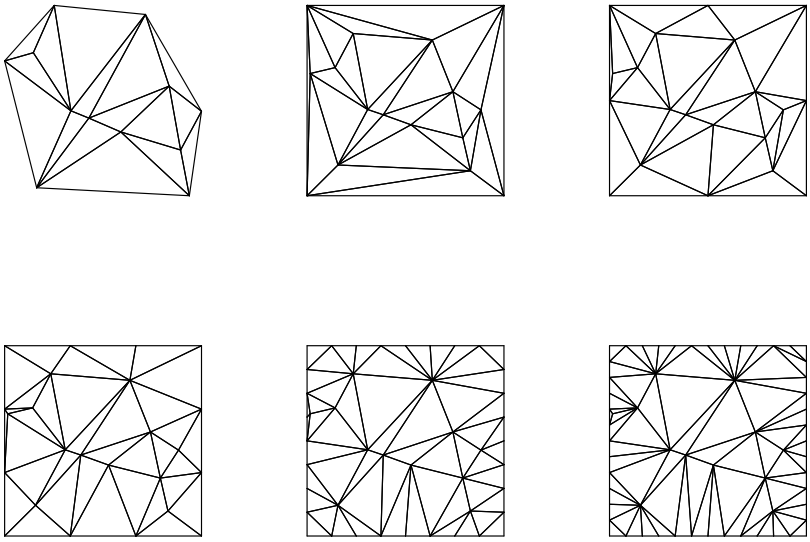


Figure 4.4: Triangulation for $\beta \in \{1, 2, 3, 8, 12\}$ alongside original triangulation.

4.1.7 Generating the borders

The set of points to be used as a border constitutes of the following:

- a point for each vertex of the n-cube describing the range of data-points
- $(\beta - 1)$ points on each edge (1-face)
- $(\beta - 1)^2$ points on each face (2-face)
- $(\beta - 1)^3$ points on each cell (3-face)
- ...
- $(\beta - 1)^{n-1}$ points on each $(n - 1)$-face

The number of points needed given a dimensionality n and a border-saturation β can therefore be calculated by

$$\#P(n, \beta) = \sum_{i=0}^{n-1} F(n, i)(\beta - 1)^i$$

where $F(n, i)$ is the number of i-faces on a n-cube (McCann, 2010):

$$F(n, i) = 2^{n-i} \binom{n}{i}$$

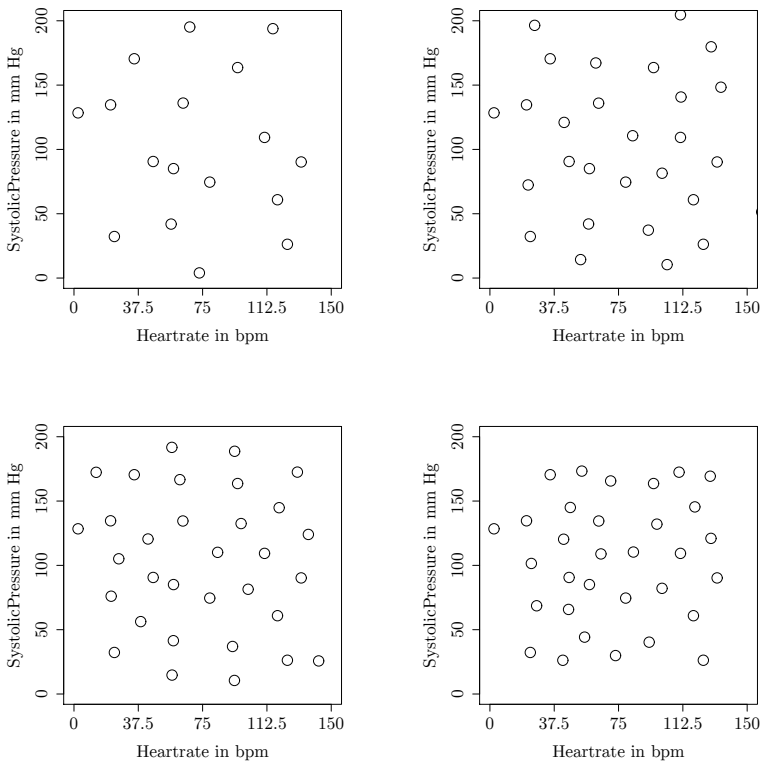


Figure 4.5: Scatter plot of the first twenty suggestions for $\beta \in \{1, 2, 4, 8\}$. Note that out-of-bounds points are not plotted, implying that plots with fewer points have more points located outside the boundaries. Only the last two show all 32 points.

The actual value of $P(n, \beta)$ can intuitively be seen as the Cartesian product of n instances of interval(β), also known as its Cartesian Power, of which only those points for which at least one of its members is equal to -1 or 1 are kept. In other words, for which the infinity norm $\|x\|_\infty$ equals 1.

$$P(n, \beta) = \{x \mid x \in \text{interval}(\beta)^n \wedge \|x\|_\infty = 1\}$$

$$\|x\|_\infty = \max_i |x_i|$$

The `border(dim, beta)` function is shown in Listing 4.3. `cart_power` is set to the appropriate Cartesian power of $(-1, 1)$ and subscripted by a list of booleans to filter out values: any vertex which, after application of the anonymous function, evaluates to `FALSE` is removed from the list.

```
border ← function (dim, beta)
{ if (beta %% 1 > 0 || beta ≤ 0)
  { stop("beta must be a positive integer") }
  else if (dim %% 1 > 0 || dim ≤ 0)
  { stop("dim must be a positive integer") }
  else
  { interval ← seq(from = −1, to = 1, by = 2/beta)
    cart_power ← expand.grid(rep(list(interval),dim))
    cart_power[apply(abs(cart_power) == 1, 1,
                     function (r)
                     { Reduce(function (x,y)
                              { x || y }, r) }) ,]}}
```

Listing 4.3: Border function implemented in R.

4.1.8 Feature Scaling

The interval-function creates an interval between -1 and 1 in β steps. This is because all features are scaled to lie between -1 and 1, even though the actual measurements might range from 0 to some arbitrary maximum. This feature scaling is applied to make sure that all features are of the same importance when applying logistic regression later on. As shown in Listing 4.4, the actual features can easily be converted to scaled features, and vice versa, using the sensors' associated minimum and maximum values. All functions are implemented with feature scaling in mind and expect values in the range $(-1, 1)$.

```
funscale ← function(X, sensors)
{ scale ← (sensors$max − sensors$min) / 2
  return(as.vector(X) * scale + scale) }

fscale ← function (X, sensors)
{ scale ← (sensors$max − sensors$min) / 2
  return((as.vector(X) − scale) / scale) }
```

Listing 4.4: Feature scaling implemented in R.

4.1.9 Avoiding symmetry

The algorithm presented above tends to favour generating a symmetrical dataset: As the range of values is a perfect n-cube, the first point suggested will be the centre, followed by a group of points equidistant from the first. This is undesirable, as symmetrical data points features will introduce redundant features when multiplied during the feature mapping process, which will not help in generating a better hypothesis but will slow down the learning algorithm.

To prevent generating such a duplicate set of data, we will move each suggestion by a small random amount, controlled by a variable δ which represents the maximal displacement for each point in each dimension. In order to ensure that this displacement will not place points outside the feature boundaries, this displacement will be opposite to the sign of the original location. This results in the data point being moved slightly towards the centre, which generally is the most interesting area to collect data on. We achieve this by replacing each vector element c_i by the weighted mean of $r \cdot 0$ and $(1 - r)c_i$, where $r \sim U([0, \delta])$ is a random variable uniformly distributed on $[0, \delta]$. Listing 4.5 shows the implementation of this function in R.

```
displace ← function (cc, delta)
{ for (i in 1:length(cc))
  { disp ← runif(1, (1 − delta), 1)
    cc[i] ← cc[i] * disp }
  return(cc) }
```

Listing 4.5: Displacement function implemented in R.

4.1.10 Putting it together

The `suggest` function will be responsible for the suggestion of data points, using the methods described above. Its source is shown in Listing 4.6

```
suggest ← function (X, beta = 8, delta = 0.05)
{ b ← border(ncol(X), beta)
  colnames(b) ← colnames(X)
  X2 ← rbind(b, X)
  del ← delaunayn(X2)
  tris ← split(del, c(row(del)))
  tris ← lapply(tris, function(tri) t(X2[tri,]))
  ccs ← lapply(tris, circumcentre)
  ccs ← lapply(ccs, function(cc) displace(cc, delta))
  areas ← lapply(tris,area)
  ordering ← order(unlist(areas))[length(areas)]
  ccs[ordering] }
```

Listing 4.6: Suggest function implemented in R.

4.2 Learning

Once enough data points have been entered, a classification algorithm can be run on the final X, containing the input (features[iii]), and $\boldsymbol{y}$, containing the trained output (associated predictions), to create a hypothesis function. As this is a relatively simple dataset, we will use logistic regression with feature mapping and regularisation. If multi-class classification is desired, a one-vs-many approach may be used to distinguish between a given set of classes. For now, we will limit the output to two classes (healthy and alarming); as the output represents the certainty of a set of measurements belonging to either category, we can still differentiate between readings that *might be* cause for alarm, and readings that certainly are. As we are using logistic regression, the hypothesis function will look as follows:

$$h_\theta(\boldsymbol{x}) = g(\boldsymbol{\theta}^T \boldsymbol{x}),$$

where g is the sigmoid function

$$g(z) = \frac{1}{1 + e^{-z}} \ .$$

After logistic regression has converged on the training data, the result will be the parameter $\boldsymbol{\theta}$ to which the hypothesis function can be applied to create a function $h(\boldsymbol{x})$ which the agent can use to interpret the data collected by the sensors. As $\boldsymbol{x}$ is constructed for each measurement by the agent and $\boldsymbol{\theta}$ is a variable determined by the training algorithm and sent to the agent at creation, The hypothesis function $h_\theta(x)$ can be hard-coded into the agent. By separating $h_\theta(x)$, which is always the same, and $\boldsymbol{\theta}$, which is not, the agent can be updated with a new hypothesis function (*e.g.* when a new sensor is added) by sending a new $\boldsymbol{\theta}$ variable, eliminating the need to create and install an entirely new decision agent.

4.2.1 Training

The $\boldsymbol{\theta}$ parameter is determined from the training data by iterating logistic regression, repeatedly evaluating the current fit for each training-example and adjusting $\boldsymbol{\theta}$ until a good fit has been found.

[iii] The matrix X contains a row for every training example; the vector $\boldsymbol{x}$ contains a single set of measured values fed into the finished hypothesis function used to generate a prediction.

First, an empty $\boldsymbol{\theta}$ is initialised (by convention as a vector of zeros), and a cost function $J(\boldsymbol{\theta})$ determines the aggregate error for the hypothesis function on the training data:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\Big(\log(g(X\boldsymbol{\theta}))^T \boldsymbol{y} + \log(\mathbf{1} - g(X\boldsymbol{\theta}))^T (\mathbf{1} - \boldsymbol{y})\Big) + \frac{\lambda}{2m}\boldsymbol{\theta'}^T\boldsymbol{\theta'}$$

where

$$\boldsymbol{\theta'} = \boldsymbol{\theta}_{1.....n} \ .$$

Here, the matrix X is multiplied by the vector $\boldsymbol{\theta}$ and the sigmoid function is applied to the resulting vector, yielding the predictions for every training example. Applying the log function to every element yields a large negative number for each example where the prediction is negative, approaching $-\infty$ as the confidence in the prediction increases. Subsequently, this vector is multiplied by $\boldsymbol{y}$, in which every positive example is marked by a 1 and every negative example is marked by a 0. This way, the penalty for a negative prediction is kept for every example which should have been positive, whereas the correct predictions have their penalty multiplied by zero. The same operations are applied to the predictions and correct answers a second time after subtracting each from a vector of ones (denoted here as $\mathbf{1}$) of size m[iv]. This penalises every false positive while ignoring every true positive. The resulting cost is divided by the number of training examples m to get the average error. As this method yields a negative cost, the result is multiplied with -1.

The second part of the cost function uses the regularisation parameter λ to penalise large values in $\boldsymbol{\theta}$. By encouraging the algorithm to keep $\boldsymbol{\theta}$ small, we prevent overfitting. Without it, logistic regression can end up finding a good fit to the training data which is worthless for unseen examples. Figure 4.6 shows an example of an overfitted hypothesis function: the generated decision boundary is very specific to this training set and would likely misclassify new measurements. Section 4.2.4 discusses regularisation in further detail.

$\boldsymbol{\theta}_0$ is, by convention, not regularised by λ, as it represents the bias term, a constant in the hypothesis function. Each element of the vector $\boldsymbol{\theta}$ is a used to weigh one of the features of the input $\boldsymbol{x}$. Listing 4.7 shows the implementation of `sigmoid(z)`, `h_theta(theta, X)` and `J(thetha, X, y, lambda)`, together with the initialisation of `theta`.

[iv] m represents the number of training examples.

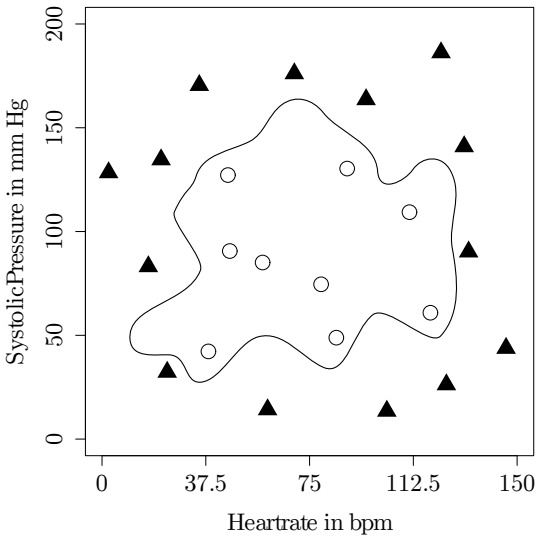


Figure 4.6: Overfitting.

```
theta ← matrix(0, ncol=1, nrow=2)

sigmoid ← function (z) { 1.0 / (1.0 + exp(−z)) }

h_theta ← function (theta, X)
{ sigmoid (X %*% theta) }

J ← function (theta, X, y, lambda)
{ hx ← h_theta(theta, X)
  m ← nrow(y)
  cost ← (−1 / m) * ( t(log(hx))   %*% y
                    + t(log(1−hx)) %*% (1−y) )
       +  lambda / (2*m) * theta[−1] %*% t(theta[−1])
  colnames(cost) ← c("cost")
  return(cost) }
```

Listing 4.7: Sigmoid, hypothesis and cost function implemented in R.

4.2.2 Finding a good value for $\boldsymbol{\theta}$

After the cost for the initial $\boldsymbol{\theta}$ has been determined, it has to be altered $\boldsymbol{\theta}$ in such as way that it will decrease the cost function, thus finding a better fit. This is achieved by the calculation of the partial derivative of $J(\boldsymbol{\theta})$ for each element in the vector. As the derivative represents the slope of the function $J(\boldsymbol{\theta})$ at point $\boldsymbol{\theta}$, it can be used to find a better value for $\boldsymbol{\theta}$. This process will be repeated until the derivative approaches zero, signifying a minimum for the cost function and thus an optimal fit to the training data.

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_0} = \frac{1}{m} X^T (h_\theta(\boldsymbol{x}) - \boldsymbol{y})$$

gives us the derivative for $J(\boldsymbol{\theta})$ for $\boldsymbol{\theta}_0$, whereas

$$\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_j} = \frac{1}{m} X^T (h_\theta(\boldsymbol{x}) - \boldsymbol{y}) + \frac{\lambda}{m} \boldsymbol{\theta}_j$$

is the derivative for $J(\boldsymbol{\theta})$ for $\boldsymbol{\theta}_1 \dots \boldsymbol{\theta}_n$. Both are combined into a single R function as shown in Listing 4.8.

```
gradient ← function (theta, X, y, lambda)
{ hx ← h_theta(theta, X)
  m ← nrow(y)
  grad1 = (1/m) * t(X[,1]) %*% (hx—y)
  gradn = (1/m) * t(X[,—1]) %*% (hx—y)
        + (lambda / m) * theta[—1,]
  grad = rbind(grad1, gradn) return(grad) }
```

Listing 4.8: Gradient function implemented in R.

4.2.3 Hill climbing

Using the cost function, its partial derivative and the variables X, $\boldsymbol{y}$, λ and an initial variable $\boldsymbol{\theta}$ we can then utilise a hill-climbing optimisation algorithm to iteratively calculate which direction we need to move $\boldsymbol{\theta}$ in to lower our cost function. For this, many algorithms exist. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm is an efficient algorithm (Daumé III, 2004) available in R (R Foundation, n.d.). Applying this to the mentioned functions and variables, we can get a good fit for our variable $\boldsymbol{\theta}$. The full `logit(X, y, fd, lambda)` function is shown in Listing 4.9. The `fd` parameter is explained in Section 4.2.5.

```
logit ← function (X, y, fd = 1, lambda = 0.01)
{ X ← mapfeature(X,fd)
  th_init ← rep(0,ncol(X))
  result ← optim(th_init, J, gradient, X, y, lambda, method="BFGS")
  theta ← matrix(result$par, ncol=1)
  rownames(theta) ← colnames(X)
  return(theta) }
```

Listing 4.9: Logistic regression using BFGS in R.

4.2.4 Regularisation

As mentioned when discussing the learning algorithm, care should be taken not to overfit the training-examples and produce a hypothesis function that is specific to a small dataset but fails to generalise. The opposite, however, also exists: when λ is set too large, we risk underfitting: generating a hypothesis function which is too general and fails to classify measurements appropriately. Underfitting can often be solved by using more training examples, and as the data points are entered manually and a plot of the current decision boundary is shown during entry, underfitting is more easily avoided than overfitting. There are, however, other means to ascertain a correct λ parameter has been chosen for the dataset. A powerful method used in machine learning is cross validation: A part of the training set is not used during training, but instead applied to a generated hypothesis function to verify its performance, providing insight into chosen parameters. Though this method is hard to implement with limited data available and would require hiding entered data points from the set for later usage (which may be confusing to the user as data apparently disappears), this approach does show promise for future inclusion as usage statistics and old data (both from training and usage) become available.

4.2.5 Feature Mapping

As the current model includes each variable only once, the resulting classification is limited to a straight divisor between data points. As most biological functions will have both a minimal and a maximal healthy level, this will not suffice for most features. To counter this, we will use a technique known as feature mapping. A set of features $\boldsymbol{x}$ will have features added as such:

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \text{fmap}(\boldsymbol{x}, d) = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_2^{d+1} \end{pmatrix}$$

This is implemented by first calculating all the combinations with repetition of the columns, mapping each index to the actual value of the column and calculating the product. This function is shown in Listing 4.10.

```
mapfeature ← function (X,deg, unique = T)
{ combs ← combinations(1+ncol(X), 1+deg, c(0:ncol(X)), repeats.allowed = T)
  newX ← matrix(0, ncol=0, nrow=nrow(X))
  for (j in 1:nrow(combs))
  { newCol ← matrix(1, ncol=1, nrow=nrow(X))
    for (k in 1:ncol(combs))
    { if (combs[j,k] > 0) { newCol ← newCol * X[,combs[j,k]] } }
    colnames(newCol) ← c(paste(colnames(X)[combs[j,]], collapse = " X "))
    newX ← cbind(newX, newCol) }
  colnames(newX)[1] ← c("Bias")
  return(newX) }
```

Listing 4.10: Feature mapping implemented in R.

4.3 The Decision Agent

Now that we have a variable $\boldsymbol{\theta}$, it can be used to program a decision-making agent. In order to make the agent as generalised as possible it will be given a set of general behaviours dependent on $\boldsymbol{\theta}$, so that its behaviour can be changed by sending new parameters instead of changing the code. The agent will need more information than just the $\boldsymbol{\theta}$ variable: it will need a list of "plans" telling it how to react to predictions, the order of the variables within X, the level of feature-mapping, and which agents to contact.

A UML overview of the decision-agent's structure is show in figure 4.7. As shown, the decision agent contains a great amount of variables and operations. The agent contains matrices (or more precisely, mathematical vectors) for storing $\boldsymbol{\theta}$ and $\boldsymbol{x}$ (the latter both scaled an as raw values). A `Map` is used to associate the names of features with their position in `x`. In addition, it contains a variable `interval` controlling how often a prediction is made, an integer telling the feature mapping function how many polynomials it should generate, a set of `MessageTemplates` allowing it to distinguish various kinds of messages, and a `boolean` indicating whether the agent is operational (*i.e.*, has received a valid set of instructions). The list of `Plans` and the fallback communication method are described below in Section 4.3.4.

The boolean `MDB` and the method `mdb(String)` are short for Math DeBug, and are used to print all matrices used in calculating the prediction. As this yields a large amount of multi-line output it has been given its own quasi-log-level.

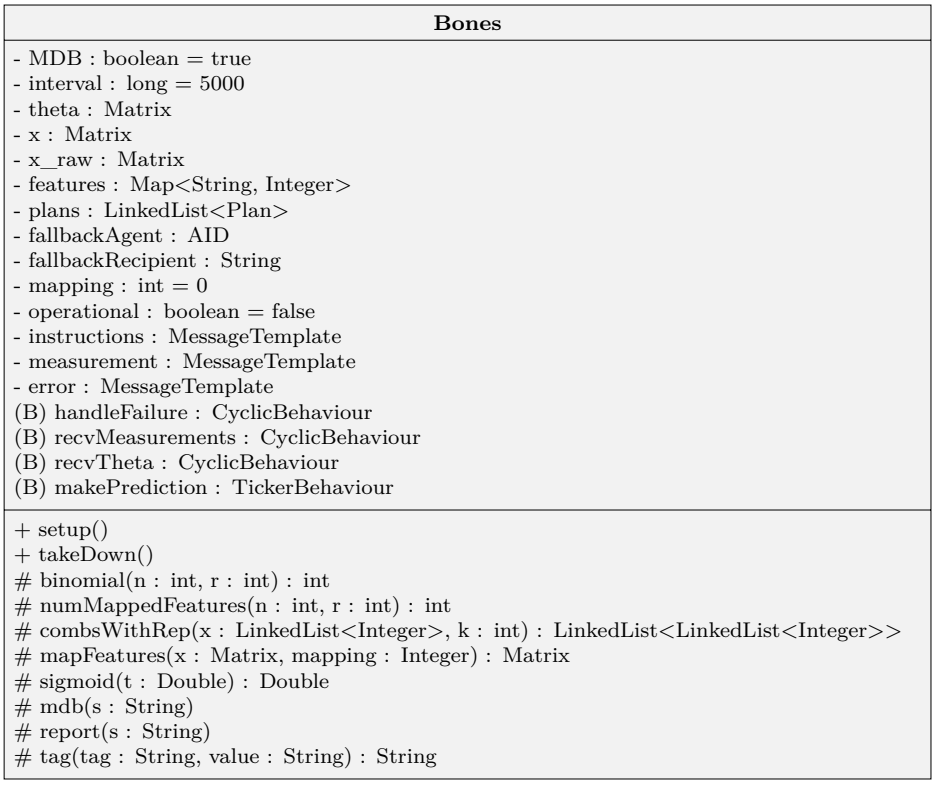


Figure 4.7: The decision-agent class.

4.3.1 Methods

In addition to the `setup()` and `takedown()` methods required by JADE as pseudo-constructors / destructors, the agent contains a number of utility-functions serving to make the code (mostly contained in its behaviours) more readable and to improve maintainability. `binomial(..)`, `combsWithRep(..)`, `numMappedFeatures(..)` and `mapFeatures(..)` are used to perform and verify the feature mapping and follow the same principles as described in Section 4.2.5.

4.3.2 Behaviours

As the decision-agent is the central part of the medical MAS, it contains a large set of behaviours: three cyclic behaviours, which are constantly active, and a ticker behaviour operating on an interval depending on the `interval` variable:

- `handleFailure` listens for messages indicating failure in any of the communication agents. In such an event, it will use a designated fallback-agent to alert an operator that the system might be unable to communicate.

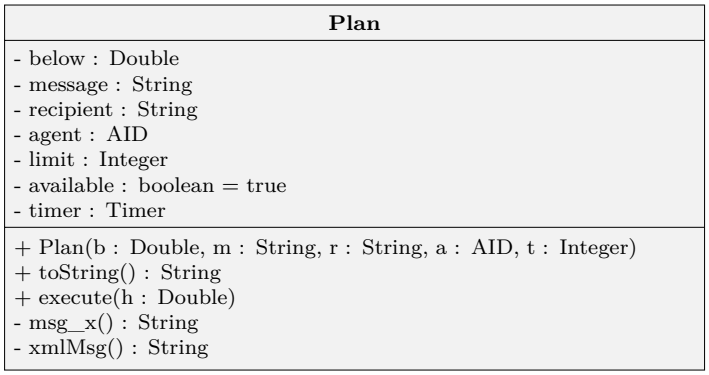


Figure 4.8: The plan inner class

- `recvMeasurements` listens for messages from the sensor-agents and saves them in `x` and `x_raw` for use in predictions.
- `recvTheta` listens for messages containing instruction sets. This aspect is explored in Section 4.3.5.
- `makePrediction` is responsible for periodically multiplying `theta` and `x` to make a prediction regarding the patient's health. This process is described in Section 4.3.3.

4.3.3 Assessing the situation

Every `interval` milliseconds, the agent uses its then-current knowledge of the features, represented in `x`, to construct a feature-mapped column-vector `x_mapped`. The inner product of `x_mapped` and the row-vector `theta` is passed through the `sigmoid(double)`-method yielding a `double` between zero and one, representing the probability that the patient is still healthy. As this number decreases, the probability of something being wrong increases. After a prediction has been calculated, the value is compared to the thresholds defined for each plan; if the calculated result is below the threshold for a given plan, the agent will attempt to execute it by messaging a communication agent.

4.3.4 Executing plans

Plans are represented by a special `Plan` class, shown in Figure 4.8. Each time a prediction is made, the agent attempts to invoke the `execute(Double)` method for each plan, passing the predicted probability. Each plan contains a threshold `below`, which is compared to the prediction when `execute(Double)` is called. Execute will send its message to its specified recipient via its specified

agent, provided two conditions are met: The prediction passed as an argument to `execute(Double)` is lower than or equal to the threshold for the plan and the `boolean available` is set to `true`.

The value of `available` is initialised as `true`, but is set to false when the plan is first executed. At the same time, a timer is started for `limit` seconds, after which `available` is reset. This prevents the MAS from flooding its recipients with messages as a new prediction is calculated, by default, every five seconds; though it may be meaningful to provide an occasional update, a realistic poll frequency for the agent to make predictions is likely always higher than a realistic notification frequency. By using a plan specific interval all frequencies can be chosen separately.

4.3.5 Receiving instructions

All of the necessary information can be delivered to the agent within a single ACL message; Listing 4.11 shows a sample XML fragment containing instructions for two features, second-degree feature mapping and two plans.

```
<instructions>
  <features>
    <feature id="SystolicBloodPressure">
      <label>Systolic Blood Pressure</label>
      <min>0</min>
      <max>200</max>
      <unit>mm Hg</unit>
    </feature>
    <feature id="HeartRate">
      <label>Heart Rate</label>
      <min>0</min>
      <max>200</max>
      <unit>bpm</unit>
    </feature>
  </features>
  <mapping>2</mapping>
  <theta>
    <value>2.402548</value>
    <value>2.769392</value>
    <value>3.467782</value>
    <value>−7.500590</value>
    <value>−2.189613</value>
    <value>−11.995721</value>
    <value>−2.301167</value>
    <value>2.064028</value>
    <value>−2.568114</value>
    <value>−2.736256</value>
  </theta>
```

```
  <plans>
    <plan>
      <below>0.6</below>
      <message>Watch out!</message>
      <via>ScreenAgent</via>
      <to></to>
      <limit>30</limit>
    </plan>
    <plan>
      <below>0.4</below>
      <message>Panic!</message>
      <via>MailAgent</via>
      <to>dokter.bernard@example.com</to>
      <limit>3600</limit>
    </plan>
  </plans>
  <fallback>
    <via>ScreenAgent</via>
    <to></to>
  </fallback>
</instructions>
```

Listing 4.11: An initialisation message as sent to the decision-agent.

The initialisation consists of five parts:

- A `features` node containing information about each feature. The order the features are presented in determines the position of values within $\boldsymbol{x}$, and must be identical to the order used during the learning process.
- A `mapping` node containing a single integer value determining the amount of feature-mapping.
- A `theta` node containing a series of decimal values representing $\boldsymbol{\theta}$. As with the features, the order is important here.
- A `plans` node containing a set of plans used to respond to predictions. Each plan includes a threshold, below which the plan will be executed, a message to be sent, the name of the agent responsible for relaying the message, an optional recipient (whether this is needed depends on the agent: a mail or SMS agent would require a recipient, whereas a screen agent would not), and a limit in seconds determining how often a plan can be executed in order to avoid flooding messages.
- A `fallback` node containing an agent and a recipient to alert when a communications agent is not functioning properly and cannot be trusted to relay important messages.

5 Communicating results to the outside world

The decision-agent as described above relies on other agents to communicate the results to a medical expert and/or the patients themselves. This choice is deliberate, as it allow new methods of communication to be added "on the fly", without changing the decision-agent's behaviour. Each communication agent added to the system represent a new option to communicate the patient's health and relay concern. As with the sensor-agents, an abstract class has been provided to facilitate the development of additional agents. This class, `CommunicationsAgent`, shown as UML in Figure 5.1.

Each communication agent has a `String` variable to hold its name and an `AID` representing the decision agent. A variable `testInterval` controls how often the agent performs a self-diagnostic. Two behaviours are present: one to continually listen for requests for communication, and another to perform the self-tests on an interval dictated by `testInterval`. The class provides the methods for setup, agent destruction and reporting to `stdout`.

Three abstract methods need to be implemented to create a `CommunicationsAgent` subclass:

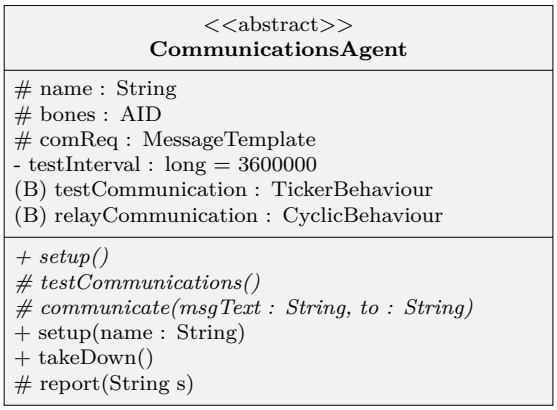


Figure 5.1: The CommunicationsAgent abstract class.

- `setup()` should set any relevant variables, at the very least including the agent's name.
- `testCommunications()` should include the code needed to run a self-test, if applicable, and throw an exception if it fails to complete the test. This exception is caught by the `testCommunication` behaviour after which a `FAILURE` message is sent to the decision-agent indicating the communication agent has become unreliable.
- `communicate()` should include all code needed to send a message, such as setting up the necessary objects for IO in Java (provided this needs to be done each time a message is sent; if the method of communication features a persistent object that can be trusted to remain operable, it can be setup in the `setup()` method) and actually sending the message. It will send a `CONFIRM`-message back to the decision agent if the sending process did not encounter any errors; in case of failure it can send either a `NOT_UNDERSTOOD` message to indicate the XML received was illegible, or a `FAILURE` indicating some sort of IO error encountered in trying to relay the message to its recipient.

5.1 Communication Requests

Requests for communication from the decision agent are packaged in a small snippet of XML, as shown in Listing 5.1. The `message`-node contains the body of the email, including the most recent value for each feature.

```
<request>
  <to>dokter.bernard@example.com</to>
  <message>Patient health in serious condition!
          — HeartRate = 54.93483905942176
          — SystolicBloodPressure = 86.20808412990199
  </message>
</request>
```

Listing 5.1: A typical message sent to `MailAgent`

5.2 MailAgent

As a sample, a simple agent relaying messages via e-mail has been provided. Its source-code it listed in Listing 5.2. It implements the `setup()` method to

set a name and to instantiate a `Session` object. As the actual authentication needs to be done every time the agent connects to the SMTP server, the code responsible was put in the `communicate` method.

```
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;

import java.io.IOException;

public class MailAgent extends CommunicationsAgent
{ String from = "<withheld>";
  String pass = "<withheld>";
  Session session;

  public void setup()
  { super.setup("MailAgent");

    Properties properties = new Properties();
    properties.put("mail.smtp.auth", "true");
    properties.put("mail.smtp.starttls.enable", "true");
    properties.put("mail.smtp.host", "smtp.gmail.com");
    properties.put("mail.smtp.port", "587");

    SmtpAuthenticator authentication = new SmtpAuthenticator();
    session = Session.getDefaultInstance(properties, authentication); }

  public void testCommunications() throws IOException
  { try
    { Transport gmail = session.getTransport("smtps");
      gmail.connect("smtp.gmail.com", from, pass);

      MimeMessage message = new MimeMessage(session); }
    catch (Exception e)
    { e.printStackTrace();
      throw new IOException(); } }


  public void communicate(String msgText, String recipient) throws IOException
  { report("Mailing! \"" + msgText + "\" to " + recipient);
    try
    { Transport gmail = session.getTransport("smtps");
      gmail.connect("smtp.gmail.com", from, pass);

      MimeMessage message = new MimeMessage(session);

      message.addRecipient(Message.RecipientType.TO, new InternetAddress(recipient));
      message.setFrom(new InternetAddress(from));
      message.setSubject("Medical MAS Alert!");
      message.setText(msgText);

      message.saveChanges();
```

```
49
50      gmail.send(message, message.getAllRecipients());
51      report("Sent message successfully....");  }
52    catch (Exception e)
53    { e.printStackTrace();
54      throw new IOException();  }  }
55
56  public class SmtpAuthenticator extends Authenticator
57  { public SmtpAuthenticator()
58    { super();  }
59
60    @Override
61    public PasswordAuthentication getPasswordAuthentication()
62    { return new PasswordAuthentication(from, pass);  }  }  }
```

Listing 5.2: A sample CommunicationsAgent subclass

Using this abstract class, any method of communication can be added to link the system up to existing medical care, and an existing system can easily be extended to include new ways of communicating. Depending on the availability of usable Java libraries this may be done in relatively small, simple agents.

6 Requirements

For the prototype, JADE was chosen as the development platform. It was selected because of its stability, the abundance of Java-libraries facilitating methods of communication and to conform to the existing product-agent environment. As the prototype MAS was developed in JADE, it should be possible to operate the system on any small device capable of running Java such as the Raspberry Pi (Upton, 2013). My initial aim for this research was to get the prototype working on such a portable system, but due to time-constraints this aspect has been reclassified as future work. As the mathematics involved on the agent side are limited to calculating the inner product of two manageable vectors and a few simple operations on integers and doubles, it is unlikely that the hardware requirements will exceed the capability of small modern single-board computers.

Though the JADE-platform was selected for the prototype, this does not preclude development of a medical MAS in another language. The concepts explored here can be implemented in any language, though support for a solid agent-development framework would be a serious asset. Nevertheless, if better performance is needed, the same principles could be implemented in a lower-level language such as C, reducing much of the overhead at the cost of lower maintainability.

7 Reliability

At the moment, the prototype constructed in this research is still limited to a simulation running on a computer. This allows the logic and principles to be tested, but as actual sensors are yet to be implemented testing these will need to be done once compatible sensors have been developed.
A number of safeguards have already been included in the prototype; Chapter 9.4 includes a few more suggestions.

- The decision-agent checks incoming sets of instructions for incongruity and refuses to accept new instructions if various parts of the XML appear to contradict each other: When the decision-agent receives a set of instructions, it will confirm whether the length of $\boldsymbol{\theta}$ matches the length of $\boldsymbol{x}$ after feature scaling, throwing an error if the two are incompatible. As each level of feature-mapping adds a number of features equal to $\binom{n+d-1}{d}$, the following equality must hold:

$$n_\theta = \sum_{i=0}^{d} \binom{n_x + i - 1}{i} .$$

- The communication-agents regularly perform self-checks and report any failure back to the decision-agent, both when an agent fails to relay a message and when it detects a fault during a self-test. Communication agents also send confirmations when a message has been relayed successfully; keeping track of undelivered messages, however, is described in Chapter 9.4 as future work.

- Each measurement is sent both as a "raw" measurement and scaled to the $(-1, 1)$ interval. At present, the decision-agent just uses the scaled value, but as the sensors' minimal and maximal values are presented in the instructions XML, the agent could verify the two measurements match as an extra verification.

8 Relation to product-agent

During development of the prototype it became clear that I overestimated the level of overlap between the product-agent (van Moergestel, 2014) and the medical agent. Though the product-agent, in particular the "use-phase", was the inspiration for utilising multi-agent technology to monitor the vital statistics of a patient, this phase has not yet been researched to the same level as the product-agent's production phase. At this time, several ideas are being developed but a clear standard has not yet emerged. The approach used by the medical agent, *i.e.* using a classification algorithm to provide meaning to numerical data regarding a patients well-being, might be adapted to suit the purpose of the product agent in interpreting statistics collected during a product's use-cycle. In doing so, it can provide an early warning of components' failure, given sufficient data can be collected for a classifcation algorithm to find correlation between those statistics and the condition of the product's components.

The groundwork laid in the development of the product agent could, however provide a serious asset in realising the medical agent system. As the medical MAS has been developed with modularity in mind, there is no such thing as a "standard medical MAS". Each medical MAS can be tailored to the individual patient and personnel, incorporating a varying subset of agents to provide the desired functionality. Many of these agents require a certain level of hardware-support: sensor-agents can not function without the specific sensor for which they were programmed. Various communications agents require various hardware interfaces to relay information: for most forms of communication this means a mobile internet connection, for others a cellular connection would be more appropriate. Communication to the patients and their surroundings could be done using a visual or auditory interface, each of which requires specific hardware to function. As such, the production of medical agent systems would benefit from agile production technologies as described by Van Moergestel to collect and assemble the necessary pieces of equipment.

9 Results

The result of this research as presented in this thesis is the basis of a long term project. At the conclusion of this research period, the following results have been achieved:

- The concept of a medical MAS consisting of three types of agents working together to monitor the patient and communicate the result.
- A method of collecting data from medical experts and utilising this knowledge to teach an agent to evaluate readings provided by sensors.
- The beginnings of a generalised framework upon which to build agents for inclusion in a medical MAS.

9.1 Answers to research questions

RQ1 Which biological factors are indicative of imminent acute cardiovascular disease? Since acquiring sufficient knowledge of medical science proved to be out of the scope of this research, this question remains unanswered; as my supervisor suggested, I instead restructured my resarch to focus on other questions. The prototype developed as part of this research was therefore designed to be agnostic of the actual meaning of the features it considers; a follow-up research with the means to acquire the relevant medical expertise should be considered to provide information usable in the development of sensor agents.

RQ2 How could these factors be discreetly monitored? This question depends on RQ1 and thus remains open as well. As eHealth applications on smartphones and consumer electronics become more prevalent and accessible even to laymen, it can however safely be postulated that the continued development of a medical MAS is unlikely to suffer from lack of possibilities on the subject of discreet monitoring. The rise of ubiquitous eHealth appliances

marks a promising development and a field for further investigation. As answers to RQ1 become available, it will become possible to explore this question further.

RQ3 Considering the results of RQ1 and RQ2, what would be the most appropriate design for a medical agent? The design of the medical MAS has been the subject of Chapter 2. As the specifics of the features monitored were left out of the picture, an adaptable, modular design has been realised capable of supporting relevant features as they are found. The design presented is adaptable to features unrelated to heart-disease as well, and could easily be adapted into a more generalised intelligent eHealth monitor provided relevant medical knowledge is available.

RQ4 How can a proposed medical agent be trained to recognise and respond to alarming measurements? Using machine learning algorithms, the decision-making agent of the medical MAS can be successfully trained to monitor a diversity of features and recognise alarming situations. Using a set of plans, appropriate responses to these situations can be defined allowing the agent to perform its designated task.

RQ5 What would be the requirements in hardware and software of such an agent? The selection of the JADE platform has made it possible to run a medical MAS on any device capable of running a JVM. Mathematics involved in assessing a patients health have been kept as simple as possible, keeping the system requirements limited. Performance testing on small devices is still required and will be a focus of future research.

RQ6 What is the relation between the product agent suggested by Van Moergestel et al. and the medical agent suggested in this research? Despite my preconceptions, the substantive similarities between the two concepts are limited. The product agent does, however, provide a promising method of production for a medical MAS: the latter's focus on modularity and adaptability can greatly benefit from the agile principles the product agent promises to provide.

RQ7 How is important patient information stored and communicated in current medical care? Like RQ1, this question is left unanswered

and the subject of further research, preferably by a researcher with a background in medical science.

RQ8 How could a "medical agent" be securely linked to / incorporated into existing systems in medical care? Though the previous question failed to yield an answer about existing systems in medical care specifically, the prototype medical MAS was designed to be able to support a large diversity of communication methods. As new methods can easily be provided by the creation of a new agent, it should not be a problem linking a medical MAS with an existing system. Chapter 9.4 provides a suggestion for increasing the security of communications by using asymmetric encryption.

RQ9 How could this agent be tested to ensure sufficient reliability? In chapter 7, some methods of error detection and prevention are discussed. Chapter 9.4, on future research, contains a number of additional suggestions for error detection and handling. Further testing is required once the prototype incorporates actual sensors and reaches the maturity required to warrant a test-phase outside of laboratory-conditions.

The main question or problem statement was: "How could agent technology contribute to increase the number of quality-adjusted life years (QALY) in acute cardiovascular disease?" This research has provided a basic prototype upon which to base further research in realising an agent-based monitoring system capable of detecting and reacting to acute cardiovascular disease and potentially other acute risks to a patients health.

9.2 Prototype (proof of concept)

During this project, one of my primary research methods has been the development of a prototype medical MAS. In addition to the answers presented above, this research has yielded an R-based application capable of querying an expert user for information, collecting the necessary data to train a learning algorithm and provide parameters to a medical agent. In addition, a number of JADE-based agents have been developed: The aforementioned decision-agent forms the central part of the medical MAS and is supported by a number of sensor-agents (providing data to the decision agent) and communication agents (providing a means to communicate the decision agent's results). A number of "mock" sensor-agents have been developed for testing purposes,

simulating measurements collected by a real sensor-agent. Additionally, two communication agents have been provided, the first for printing to a console, the latter capable of sending an e-mail. Collectively, these agents form a prototype medical MAS which demonstrates all the relevant principles detailed in this thesis, and can form the foundation for further development. Figure 9.1 provides a schematic overview of the prototype's subdivisions and flow of information.

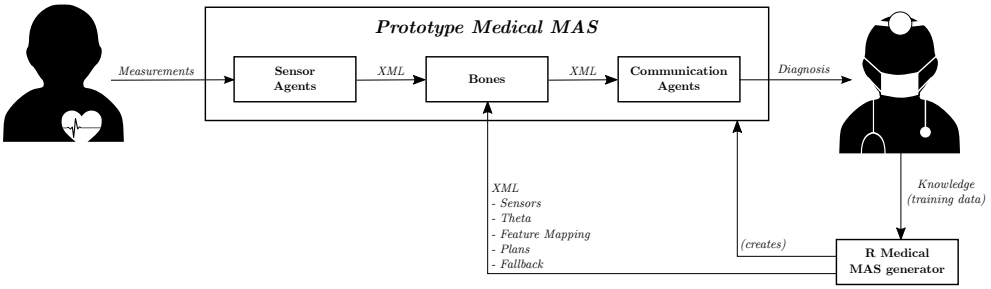


Figure 9.1: Prototype Overview

9.3 Conclusion

Though the prototype produced during this research can not yet be considered "ready-to-market", the foundation laid here provides a promising base for future development. The solution presented in this thesis provides the following advantages:

- Modularity. By separating the mathematical foundation from the input and output and providing a standardised way of connecting the parts into a working MAS, a medical system can be assembled tailored to a patients needs.
- Adaptability. By utilising machine learning to create a feature-agnostic way of assessing dangerous situations, the medical MAS can be adapted to other medical purposes.
- Testability. Because the functionality is separated into small, logical units, each part can be tested in an isolated environment.
- Robustness. The modularity of the product allows for duplicate agents, sensors or other hardware, providing a fallback for broken units.

9.4 Future Work and Recommendations

A proof of concept has been realised, but further development is required before it can be considered "ready to market". This section lists a number of improvements and features recommended for the continued development of the prototype into a finalised product.

9.4.1 Actual sensors

For testing the prototype, a small set of "mock sensor agents" has been programmed to provide random measurements for the decision making agent to consider. The first step towards turning the prototype into a usable product is the addition of actual sensors and their respective measurement functions. More medical knowledge is required to determine valuable features, after which sensors will need to be acquired for these features; in order to use these with the prototype a sensor-agent needs to be developed for the sensor capable of reading the measurement and sending the scaled results to the decision making agent.

9.4.2 Initialisation interface

The prototype uses a very basic command line interface to collect data points and assemble the hypothesis parameter θ. For production use, a web-interface or GUI would be preferable.

9.4.3 Partial / cumulative instruction sets

Right now, the agent expects its instructions in a single XML message containing θ, a list of features, a set of plans to execute on prediction thresholds and a fallback communication. A new XML message would overwrite all existing instructions. A better approach would be to replace parts of the instruction-set only when a new XML message actually contains a replacement for that specific part.
For sets of parameters (such as plans or sensors), another option would be to define instructions as adding / removing items from the set.

9.4.4 Performance testing on smaller systems

The current prototype runs on a PC and serves as a proof-of-concept of the mathematical foundations and design choices regarding a medical MAS. Future research is still needed as to its performance running on smaller devices.

9.4.5 Agents running outside of the platform

- Sensor-agents running on their own hardware, communicating via FIPA messages provided a TCP/IP connection exists.
- Communications-agents running on a cellphone, using SMS to communicate errors.
- Communications-agents running on a server in the hospital, logging or graphing data.
- A doctor communicating instructions to the medical MAS to alter its behaviour.

9.4.6 Trust

Messages to and from the medical agent could be encrypted and signed using public-key encryption to verify the identity of a medical authority before accepting new instructions and protect the users privacy by communicating encrypted data only.

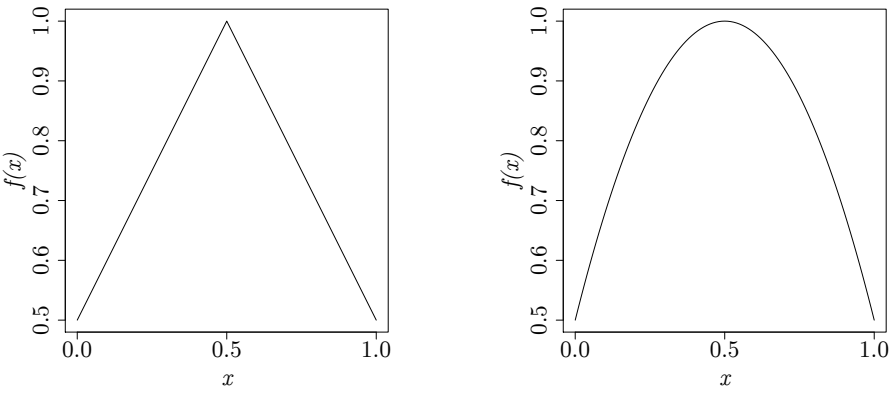
9.4.7 Improved plans

The current prototype only supports plans executed when a set threshold is reached: If the medical agent determines the current set of measurements warrants alert, it will communicate those measurements to medical authority. Future versions of the agent could support communicating when an upper limit is reached (good news) and timed plans like sending an overview of measured data every 24 hours. Timed plans could also be used to implement a "dead man's switch", where the agent communicates its continued operation on a specified interval. Failure to do so could be interpreted as a sign the agent has become non-operational and should be checked immediately.

9.4.8 Improved point suggestion

Presently, new points are suggested based on perceived holes in the information continuum. A future version could take into account the closeness of new points to the decision boundary, by multiplying the determinant of a simpex with a value based on how close the associated suggested vertex is to the boundary. As the hypothesis function for the vertex will output a value in $(0, 1)$ and values are more interesting as they get closer to 0.5, a translation

function will be needed. Functions of the form $f(x) = 1 - |x - 0.5|$ or $f(x) = 1 - 2(x - 0.5)^2$ appear to be good candidates for this purpose.



$f(x) = 1 - |x - 0.5|$ $\qquad$ $f(x) = 1 - 2(x - 0.5)^2$

Figure 9.2: Two candidate functions

9.4.9 Detecting malfunctioning sensors

At present, the decision agent accepts measurements from any sensor-agents it knows, but it does not notice an absence of new measurements coming in. A future version should include a timer for each sensor-agent which counts down from the last message received. If no new measurements are received before the timer runs out, the decision-agent should report the sensor as likely inoperable using its fallback communications method.

9.4.10 Verifying sent messages

Though communication agents send a confirmation when a message has been sent, the decision-agent at present does not verify if said confirmation has been received. The agent should include a data-structure listing sent-but-not-confirmed messages and periodically verify no message remains in this structure for too long.

Methods of communication capable of letting a sender know whether his message has been received and/or read (*e.g.* WhatsApp, Telegram) could warrant multiple levels of confirmation; the communication agent could send a se-

quence of labelled `CONFIRM`-message when a message has been sent, reported as received by the server, reported as received by the remote client, and reported as being read by the remote client. The decision-agent could keep track of the status of any message sent, and use the designated fallback communication agent in case a message does not advance to a new confirmation-level within a specified amount of time.

9.4.11 Configurable λ parameter

The learning process includes a variable λ used to prevent overfitting, as described in Section 4.2.4. Making this parameter configurable during data-point entry would allow the user more control over the "smoothness" of the generated hypothesis function.

9.4.12 Smarter learning

As described in Section 4.2.4, cross-validation could be used to allow the computer to determine whether the λ parameter used results in a good fit. To accomplish this, past training-sets and usage statistics (including measurements, made predictions and false positives / negatives) should be logged and saved in persistent storage for usage in cross-validation. This principle could then be applied to the data itself as well, cross-referencing added data points with existing knowledge. As the medical MAS gets tested and used, more data becomes available allowing the agent to learn not only from its own training-set, but from agents past as well.

10 Evaluation

Looking back at this project, I can safely assert that a lot did not go as I expected, both for better and for worse. I learned a lot about new subjects, especially concerning the more mathematical part of this research: before starting this research, I did not have any prior knowledge regarding the subject of Machine Learning, and though I had successfully finished my math classes I did not regard myself as very knowledgeable on the subject of Linear Algebra. If someone had told me then that these subjects would form an integral part of my project, I probably would not have believed them. In fact, someone did. Following a quick pitch during a road trip, I started out to investigate this new subject and now, six months later, I finished a course in Machine Learning, taught myself two new programming languages and used my newly gathered knowledge in my bachelor thesis project.

An aspect I am less happy about, however, is planning. Though this was never my forte, I set out resolving to try and keep a schedule. For the first few months this worked, although I had a habit of getting ahead of schedule researching while getting behind on the required documents. After these were out of the way, things went rather smoothly until I was confronted by tragic family circumstances and I was forced to delay my thesis from January-February to March-April. After I picked myself back up and resumed my research, most of my planning had gone out of the window and I found it increasingly hard to get back on track. I was making progress, but focusing on perfecting details instead of getting the rest of the work done. During the last sprint my teacher, Ir. Kaldeway, gave me some stern advice on this subject and suggested I drop perfecting the prototype until after finishing my thesis, getting me back on track.

Bibliography

Daumé III, H. (2004, August). Notes on CG and LM-BFGS optimization of logistic regression. doi:10.1.1.142.4746

Eric W. Weisstein. (2002a). Circumcenter from Wolfram MathWorld. Retrieved from http://mathworld.wolfram.com/Circumcenter.html

Eric W. Weisstein. (2002b). Delaunay triangulation from Wolfram MathWorld. Retrieved from http://mathworld.wolfram.com/DelaunayTriangulation.html

Foundation for Intelligent Physical Agents. (2002, December). FIPA ACL Message Structure Specification. Retrieved from http://www.fipa.org/specs/fipa00061/SC00061G.html

McCann, R. J. (2010). Cube face. Retrieved from http://www.math.toronto.edu/mccann/assignments/199S/cubeface.pdf

Nederlandse Hartstichting. (nodate-a). Hart- en vaatziekten in Nederland 2013. Retrieved June 4, 2014, from https://www.hartstichting.nl/downloads/cijferboek-2013

Nederlandse Hartstichting. (nodate-b). Hartstilstand. Retrieved June 4, 2014, from https://www.hartstichting.nl/hartziekten/hartstilstand

R Foundation. (nodate). R Documentation - General-purpose Optimization. Retrieved from https://stat.ethz.ch/R-manual/R-devel/library/stats/html/optim.html

Satchwell, B. (nodate). *The e-health revolution in personal health management*. Retrieved from http://cordis.europa.eu/pub/ist/docs/99helsinki/bsatchwell.doc

Stein, P. (1966). A note on the volume of a simplex. *The American Mathematical Monthly*, *73*(3), 299–301. Retrieved from http://www.jstor.org/stable/2315353

Upton, E. (2013, September). Oracle Java on Raspberry Pi. Retrieved from https://www.raspberrypi.org/blog/oracle-java-on-raspberry-pi/

van Moergestel, L. (2014). *Agent Technology in Agile Multiparallel Manufacturing and Product Support*. Retrieved from http://dspace.library.uu.nl/handle/1874/298812

Wooldridge, M., Jennings, N. R. & Kinny, D. (2000, September). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, *3*(3), 285–312. doi:10.1023/A:1010071910869

World Health Organisation. (2013, September). Mental health and older adults (Fact sheet N°381). Retrieved June 4, 2014, from http://www.who.int/mediacentre/factsheets/fs381/en/

Medical Agents

Using Artificial Intelligence to support immediate medical care

« Project Plan »

Date

9th October 2015

Author

Brian van der Bijl 1540263

Teachers

Dr. Leo van Moergestel

Ir. Joop Kaldeway

Utrecht University of Applied Science

Faculty of Nature and Technology

Contents

1 Motivation and context

This document details the steps necessary for the successful completion of my bachelor thesis. The study it entails is sponsored by the Hogeschool Utrecht Centre of Technology and Innovation (HU-CTI) and based on my own research proposal. Dr. Leo van Moergestel will represent the HU-CTI as "mandator". Ir. Joop Kaldeway will be my supervisor for this project, and together with Henk van Nimwegen will be responsible for my examination as well.

Dr. Leo van Moergestel
Nijenoord 1
3552 AS Utrecht
leo.vanmoergestel@hu.nl
+31 88 481 88 00

Ir. Joop Kaldeway
Nijenoord 1
3552 AS Utrecht
joop.kaldeway@hu.nl

This research can be considered part of the larger research conducted by HU-CTI regarding agent-technology and multi-agents systems, specifically the product agent by Dr. Van Moergestel.

1.1 Problem to be addressed

As medical science progresses, many debilitating and potentially fatal conditions are becoming easily preventable in the presence of first-world healthcare. The success of treatment largely depends on quick action being taken: as more time passes before adequate care is provided, the chance of recovery quickly decreases and the loss of quality-adjusted life years (QALY) increases as lack of oxygen causes brain cells to permanently die. Acute cardiovascular diseases[1] are among the most prevalent diseases in the western world: In the Netherlands alone, approximately 15 000 people suffer cardiac arrest outside of hospitals each

[1] Brain- and heart attacks and related afflictions.

year (Nederlandse Hartstichting [NH], 2014b). In 2012, 29 000 people suffered a Myocardial Infarction (MI) and 44 000 people were affected by a Cerebrovascular accident (CVA) (NH, 2014a). All of these show a direct correlation between the time to onset of treatment and QALY in victims.

1.2 Proposed solution (hypothesis)

This study aims to explore the possibilities of making monitoring vital functions in "high-risk" individuals[2] more accessible and affordable by utilising agent technology to create cheap and replaceable dedicated monitors. By proposing a "medical agent", this study hopes to decrease response-time leading to increased QALY following an attack. Furthermore, a "medical agent" could potentially allow people to live independently when without it they would be forced to rely on a retirement home to provide constant attention. This would reduce both the costs on society as well as leading to better psychological health (World Health Organisation [WHO], 2013) in patients.

To solve this problem, this research proposes a medical agent or medical Multi-Agent System (MAS) programmed to monitor relevant vital functions and use its knowledge of the patients situation to anticipate emergencies, communicating them to the patient, caregivers and/or medical authority. This research focuses on the above-described cardiovascular diseases, but should in no means be limited to this kind of ailment. The proposed model could be extended to watch for a variety of problems, *e.g.* pulmonary embolism, acute dyspnea, (chronic) obstructive pulmonary disease, asthma exacerbation, epilepsy and diabetesmellitus. For the sake of manageability, this research will address a single area. Should the results be as expected, further research could be initiated to expand the scope. A medical agent should be, by design, extendable: additional sensors and knowledge could be incorporated into the framework using a common architecture to address different threats using a single core design.

[2] Those with an elevated risk of acute cardiovascular diseases, *e.g.* elderly and those with a history of such attacks.

2 Analysis and research statement

To verify the proposed solution described above, a number of questions will need to be addressed after which a proof-of-concept can be designed to ascertain the validity of the envisioned model.

2.1 Research questions

The main question this research will aim to answer can be summarised as "**How could agent technology contribute to increase the number of quality-adjusted life years (QALY) in acute cardiovascular disease?**". To answer this question, several sub-questions will need to be asked:

RQ1 Which biological factors are indicative of imminent acute cardiovascular disease?

RQ2 How could these factors be discreetly monitored?

RQ3 Considering the results of RQ1 and RQ2, what would be the most appropriate design for a medical agent?

RQ4 How can a proposed medical agent be trained to recognise and respond to alarming measurements?

RQ5 What would be the requirements in hardware and software of such an agent?

RQ6 What is the relation between the product agent suggested by Van Moergestel et al. and the medical agent suggested in this research?

RQ7 How is important patient information stored and communicated in current medical care?

RQ8 How could a "medical agent" be securely linked to / incorporated into existing systems in medical care?

RQ9 How could this agent be tested to ensure sufficient reliability?

3 Method of approach

In order to find a conclusive answer to my research questions, several forms of research will need to be utilised. This section outlines the various methods I plan on applying on a per-question basis.

Proof-of-concept

In order to accurately judge the validity of the answers acquired during these research steps, several questions will also entail furthering development of a proof-of-concept: As soon as questions are deemed to be acceptably answerered the results will be incorporated into the prototype, and thus tested before being fully accepted. This approach implies a substantial amount of intertwining between research and the development of the proof-of-concept. Therefore, my planning on advancement of the prototype will closely follow the research process.

3.1 Biological factors

Which biological factors are indicative of imminent acute cardiovascular disease? (RQ 1)

To answer this question, a degree of medical knowledge is required. In order to acquire sufficient knowledge on the subject, I plan on conducting expert interviews at the Hogeschool Utrecht Centre for Innovation in Healthcare[1] and the Hogeschool Utrecht Faculty of Healthcare. Dr. Van Moergestel has offered to help me find (a) suitable expert(s).

[1] http://www.research.hu.nl/Kenniscentra/Innovatie-van-Zorgverlening.aspx

3.2 Monitoring

How could these factors be discreetly monitored? (RQ2)

Sensors for this purpose are being researched by the Hogeschool Utrecht. Specifically, I plan on interviewing Remko van der Lugt[2] (lectorate co-design) and Franc van der Bent[3] (microsystem-technology) because of their extensive knowledge on the subject.

3.3 Design

Considering the results of RQ1 and RQ2, what would be the most appropriate design for a medical agent? (RQ3)

After my first two questions have been answered, a clear image should have emerged on what to monitor and how to do it. In this phase, I plan to compare various architectural choices for the agent(s) and come up with a functional design for the system. As a clear "golden standard" for agent-based software design does not appear to have emerged yet, I expect I will need to try various methodologies to see which one works best for this project. Dr. Van Moergestel has suggested I look into the Gaia methodology (Wooldridge, Jennings & Kinny, 2000) to provide a starting point.

3.4 Reasoning

How can a proposed medical agent be trained to recognise and respond to alarming measurements? (RQ4)

Depending on the monitored factors and their relation and relevance, a mathematical or logical model should be established for the agent to properly utilise acquired data. As at this point in time the prerequisites are unknown so it would be premature to make a definitive suggestion for a model. Multi-variate statistical analysis and machine-learning could likely be utilised to ascertain the current and projected situation, after which plans will then be required to react to the perceived situation.

[2] http://www.research.hu.nl/Onderzoekers/Remko-van-der-lugt.aspx
[3] http://www.research.hu.nl/Onderzoekers/Franc-van-der-Bent.aspx

To accomplish this, a literature study should provide an answer to the stated question.

3.5 Requirements

What would be the requirements in hardware and software of such an agent? (RQ5)

After a mathematical model is agreed upon, an agent platform can be selected to support a prototype agent. This part of the research process will define requirements for a platform to be applicable, and a specific platform will need to be selected for the proof-of-concept. Further development could easily be moved to another platform if required (*e.g.* to lessen hardware-requirements); the specific platform selected will primarily concern the development of the proof-of-concept.

Once an agent platform is selected, a hardware platform can be chosen based on the agent platform dependencies. As the mathematical model will likely exist outside of the agent platform (unless, of course, a platform happens to support it), this choice could yield additional requirements. For example, multi-variate statistical analysis will require vector-calculations, which might benefit from a dedicated graphics processing unit. These factors will need to be taken into account when selecting a preliminary hardware-platform for development of a proof-of-concept.

At this point, development of a prototype will commence implementing the chosen mathematical model on the selected platform and starting support for the sensors decided upon in phase 2.

3.6 Relation to product-agent

What is the relation between the product agent suggested by Van Moergestel et al. and the medical agent suggested in this research? (RQ6)

During the previous steps of this research, similarities and differences between the product agent and the medical agent will present itself.

Based on the level of similarity, development of the two concepts could be coordinated to support each other. These interdependencies will need to be researched by comparing design decisions and finding common ground.

In addition to overlap between the two projects, this question will also aim to find and/or confirm possibilities for the application of the product-agent in supporting the medical agent. The fact that the medical agent is an example of a patient-tailored product suggests that a product agent could handle assembly and maintenance of medical agents. As at this stage the design and requirements of the medical agents should become progressively more perceivable, an assessment can be made as to the applicability of this approach.

3.7 Incorporation into existing medical care

How is important patient information stored and communicated in current medical care? (RQ7)

The medical agent could use a myriad of ways to communicate to the user, (family) caregivers and a medical authority. Potentially specific forms of communication with medical authority could be chosen to conform to pre-existing standards used in the medical world to facilitate easy adoption and incorporation. Alternatively, new methods developed will need to be supported from both sides (*i.e.* both from the medical agent and from medical care organisations). This question aims to determine the current standards to which the agent should confirm, and will be answered by expert interview earlier in this research: As this question does not build upon any previous questions, it can be asked in an expert interview when RQ1 is being answered.

How could a "medical agent" be securely linked to / incorporated into existing medical care? (RQ8)

Depending on which standards emerge when answering RQ7, some literature study might be required to familiarise myself with the subject. The results will then be implemented into the prototype.

3.8 Reliability

How could this agent be tested to ensure sufficient reliability? (RQ9)

Several parts of the medical agent system will need to be tested to ensure reliability. Firstly, the knowledge of a medical agent will need to be verified on creation, and at regular intervals during operation. Secondly, hardware and software operating the medical agent will require regular monitoring for defects and flaws. Lastly, the communication channels used by a medical agent will require attention to ensure vital messages are delivered and acknowledged. All of this should be implemented into the proof-of-concept.

With the final functionality implemented, the remaining step is to test the reliability of the prototype. For this, the system will be considered to consist of two separate entities, each to be tested in its own way. The reasoning behind this split is mostly ethical and is described in Section 6.1. Testing is split in the following categories:

Sensor testing: unit tests and simulations

The sensor-related code will be tested by applying sensors to healthy individuals and comparing the measurements to the actual values as measured by calibrated sensors.

Logic testing: expert consultation

The logic behind the agent will be tested by feeding it simulated measurements and comparing its assessments to those made by trained medical personnel.

3.9 Overview of research methods used

This table summarises the planned methods and expected results for every research question:

RQ	Method	Expected results
1	Expert interviews	Know biological factors to monitor; relations between factors; indication of cut-off values
2	Expert interviews	Know which sensors to use; how to read those sensors
3	Literature study and functional design	Specify agent design and architecture
4	Literature study and implementation in POC[4]	Settle on required mathematics / logic
5	Implementation in POC	Choose a platform; determine hardware requirements; implement architecture and logic in prototype
6	Functional analysis and comparative study	Comparison of approaches; assessment how both projects can benefit each other; determine possibility of code sharing
7	Expert interviews	Overview of present medical infrastructure
8	Literature study and Implementation in POC	Determine on communicating measured values
9	Unit testing and simulations (sensors); expert consultation (logic)	Verify validity of proposed solution

[4] proof-of-concept

4 Goal and expected results

The primary goal of this research project is determining the applicability of agent-technology in medical monitoring. To validate the stated hypothesis, a working proof-of-concept is to be developed. To accommodate the prototype, further research will be required. In addition to answers to the above-stated questions, this research aims to deliver a prototype medical agent demonstrating the following capabilities:

- Ability to reliably monitor a number of vital statistics[1].
- Ability to determine a valid assessment of the patients health based on these measurements.
- Ability to formulate a plan based on the perceived patient-condition.
- Ability to carry out these plans.

The prototype medical agent should be executable both in a simulated environment (providing a safe way to conduct tests and observe agent behaviour) and on real hardware connected to real sensors (demonstrating the validity of the concept).

[1] The number and identity of which will be determined by answering the first research question

5 Preconditions and risks

In order to successfully complete the above-described steps, a number of preconditions must be met. This chapter lists those preconditions and their associated risks.

5.1 Contact with knowledge sources

Dr. Van Moergestel, FG

As this research depends upon medical knowledge not available at the Faculty of Nature and Technology, outside sources are required to find an answer to the first three research questions. At the Utrecht University of Applied Science (HU, Hogeschool Utrecht) this subject is taught by the Faculty of Healthcare (FG, Faculteit Gezondheidszorg). Dr. Van Moergestel has offered to mediate on my behalf. Failure to acquire or interpret knowledge at this point would have a large impact on the success of the entire project and can be considered a risk. Should a situation arise where the required information can not be acquired, an alternative course should be considered focussing on the technical aspect of the agent while substituting assumptions for medical knowledge. The result would then be a prototype unfit for its designed purpose, but potentially usable as a basis for further research when the missing information can be procured.

5.2 Hardware availability

Dr. Van Moergestel

As a prototype medical agent will be required for a conclusive validation of the stated hypothesis, a potential problem could arise if required hardware should prove (temporarily) difficult to procure. As long as hardware requirements are determined and communicated at an early

stage this should not prove a problem, however last-minute alterations or additions could prove a risk and should therefore be avoided.

5.3 Relation to the product agent

Dr. Van Moergestel

The medical agent-research is a subsidiary to the product-agent research efforts, its continuation is directly dependent on the continuation of its parent. At this point, these risks appear to be very small, but should nonetheless be considered in a conclusive assessment of risks.

5.4 Personal risks

Brian van der Bijl

Lastly, my personal situation constitutes a risk to the successful conclusion of this research project. As the past few years have shown me, my own mental health is not infallible and could therefore be considered a risk. Additionally, my mother could be considered a "terminal" Parkinson-patient; twice in the past half year, drastic changes in her situation have cost me a lot of time and energy and severely limited my professional and academic capabilities. This too should be considered a valid risk in the success of this research project.

6 Further considerations

6.1 Ethics

The intended result of this research will need to be able to take medical responsibility to satisfy its set conditions. As such, a consideration of ethics is of great importance — most importantly in the testing phase. To ensure an ethical approach here, the testing will be split in two separate topics:

- Sensor testing
- Logic testing

Sensor testing

The sensors and the code required to read them can easily be tested on healthy subjects, as there is no requirement to trust the unfinished prototype to make any critical assessments regarding subject health. For normal (healthy) measurements, monitored values can be compared to values acquired by trusted secondary sensors. Values outside of the healthy range will need to be *a*) simulated or *b*) extrapolated from healthy measurements .

Logic testing

The actual agent interpreting the measurements can be fully tested on generated results, as the source of its input should be irrelevant to the logic applied to assess the situation and respond to it. The agent's reaction to different simulated measurements can safely be observed and evaluated by sufficiently skilled medical personnel.

7 Planning

7.1 Weekly planning / communication

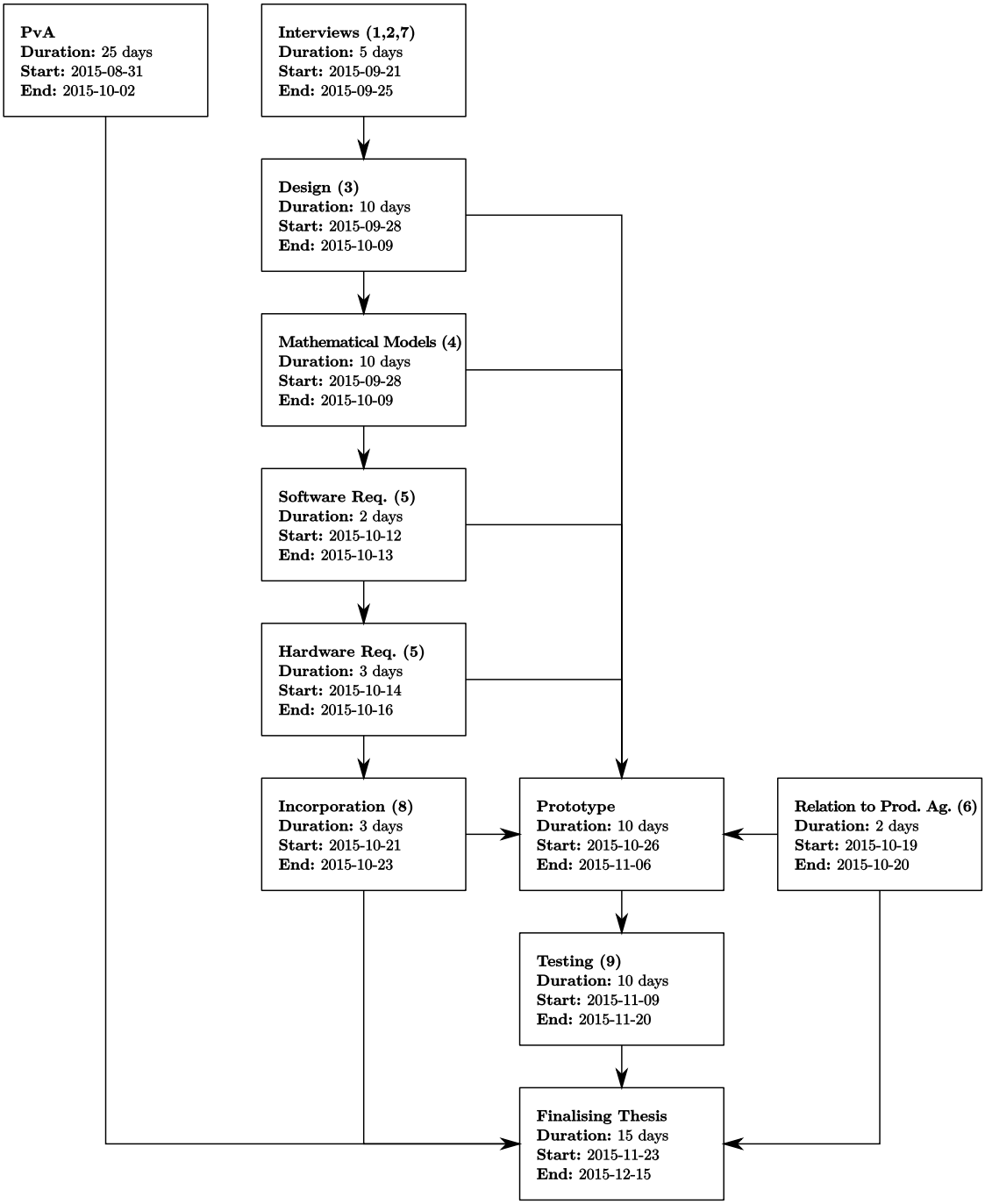
To ensure sufficient communication for the research to progress I have an appointment with Dr. Van Moergestel twice a week to discuss my goals and results. These meetings are planned on Tuesday afternoon and Thursday afternoon, though this is not a hard rule because of Dr. Van Moergestel's busy schedule.

To discuss my progress, I meet with Ir. Kaldeway at least once a week. As I also help him out as TA in one of his classes thrice a week I can easily get in an extra appointment should this be required.

7.2 Milestones

2015-09-25	RQ1 - Medical knowledge
2015-09-25	RQ2 - Sensors
2015-09-25	RQ7 - Communication in medical care
2015-10-02	Final version Project Plan
2015-10-09	RQ3 - Design
2015-10-09	RQ4 - Mathematical foundation
2015-10-16	RQ5 - Requirements
2015-10-16	First thesis draft
2015-10-20	RQ6 - Relation to product agent
2015-10-23	RQ8 - Incorporation
2015-11-06	Proof-of-concept
2015-11-20	RQ9 - Testing
2015-11-20	Second thesis draft
2015-12-15	Final thesis version

7.3 PERT chart of milestones



Glossary

cardiac arrest sudden stop in blood circulation due to the failure of the heart to contract effectively or at all. 1

CVA cerebrovascular accident (brain attack): poor blood flow in the brain resulting in cell death. 2

MAS multi-agent system: a system composed of multiple intelligent agents interacting within an environment. 2

MI myocardial infarction (heart attack): cessation of blood flow resulting in damage to the heart muscle. 2

QALY quality-adjusted life years. 1–3

Bibliography

Nederlandse Hartstichting. (2014a). Hart- en vaatziekten in Nederland 2013. Retrieved June 4, 2015, from https://www.hartstichting.nl/downloads/cijferboek-2013

Nederlandse Hartstichting. (2014b). Hartstilstand. Retrieved June 4, 2015, from https://www.hartstichting.nl/hartziekten/hartstilstand

Wooldridge, M., Jennings, N. R. & Kinny, D. (2000, September). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, *3*(3), 285–312. doi:10.1023/A:1010071910869

World Health Organisation. (2013). Mental health and older adults (fact sheet n°381). Retrieved June 4, 2015, from http://www.who.int/mediacentre/factsheets/fs381/en/