A policy based MBU control system using OSGi

Kommer Braber

Version 1.00

University of Twente:
Faculty of Electrical Engineering
Mathematics and Computer Science
ASNA GROUP

Graduation Committee:
Dr.Ir. I.A. Widya
Dr.Ir. B.J.F. van Beijnum
H. Mei Msc.

Enschede, November 21, 2007

Preface

This document contains the master thesis of Kommer Braber and is the result of eigth months of research and writing. This thesis is the final project of the study Computer Science. The research has been performed for the Architecture of Network Applications chair of the Faculty Electrical Engineering, Mathematics and Computer Science.

I would like to thank my supervisors, Ing Widya, Bert-Jan van Beijnum and Hailiang Mei for their patience, supervision and support during my research and writing. The discussions and feedback helped me to focus on the problems and showed me how to improve my accurateness.

In particular I would like to thank my girlfriend, Jannie, and my parents for their support and trust. As last I would like to thank my fellow students, Robin, Hendrik, Wouter and Mischa for a pleasant study time and help with with difficulties during courses and this thesis.

Kommer Braber
November 21, 2007

Abstract

This thesis presents a policy-based control mechanism for the Mobile Base Unit (MBU), a component of a Mobile Healthcare (M-Health) system. The MBU functions as a gateway between a body area network and the network used by health service providers. A body area network consists of wired sensors and actuators on a human body. The current specification of the MBU does not enable the adaptation of changes from the outside, e.g. user preference changes or environment changes.

Previous research showed the design of a policy-based control mechanism which enables the adaptability of an MBU [1]. It introduced an architecture that facilitates policies to be evaluated. A policy is a rule which decides to take action in case certain external changes are met.

As an implementation of the policy-based control mechanism, a service oriented architecture can be used. In such an architecture, functionalities are put into atomic services. These services communicate with each other. The OSGi framework is an example of a service oriented architecture. An advantage of this architecture is the facilities it provides for services.

We apply a policy-based control mechanism to control the MBU. This mechanism includes a refined version of the earlier researched policy-based architecture. We provide simple but realistic specifications and implementations of policies. Also an implementation of the mechanism in OSGi is presented. A prototype of this implementation demonstrates the flexibility of the policy-based mechanism and the advantages of service oriented architectures.

Contents

Chapter 1

Introduction

This chapter presents the problem description, approach and the outline of this thesis. This chapter is structured as follows: the first section presents a description of the problem. Section 1.2 explains the approach that is used in the development of this thesis. The last section states the outline and structure of this thesis by presenting a brief overview of the chapters.

1.1 Problem description

The mobility of people with a health condition that needs monitoring or check-ups regularly, e.g. a chronical disease, is often limited. Mobile health care, or M-health tries to provide a solution to improve the quality of life of these persons by introducing mobile services in the health area. These services try to limit the number of visits to or overnight stays at health care centers. A second advantage of using such services is continuous monitoring of patients which gives a better view of their health status.

M-health can be seen as the integration of a number of technologies. Medical sensors and actuators, wireless communication and mobile computing are combined into a medical assistance system. The goal of an M-health system is to provide patient care wherever the patient or the physician is. The MobiHealth system [2], its successor HealthService24 [3] and Awareness Teletreatment [4] are implementations of the M-health concept.

The MobiHealth system, visualized in Figure 1.1, consists of two parts, the body area network (BAN) which resides on the body of the patient and a health monitoring system which enables a doctor to remotely monitor the patient [5]. A body area network consists of sensors, actuators and a mobile base unit (MBU). Sensors capture vital signs from the patients body and actuators enforce a e.g. (electro)mechanical effect on a device at the patients body. The

MBU acts as a gateway between the health monitoring system and the body area network, but without a specialized control mechanism to adapt to changes in the environment. The communication between the MBU and the health monitoring system is based on available wireless communication channels, e.g. GPRS and UMTS.

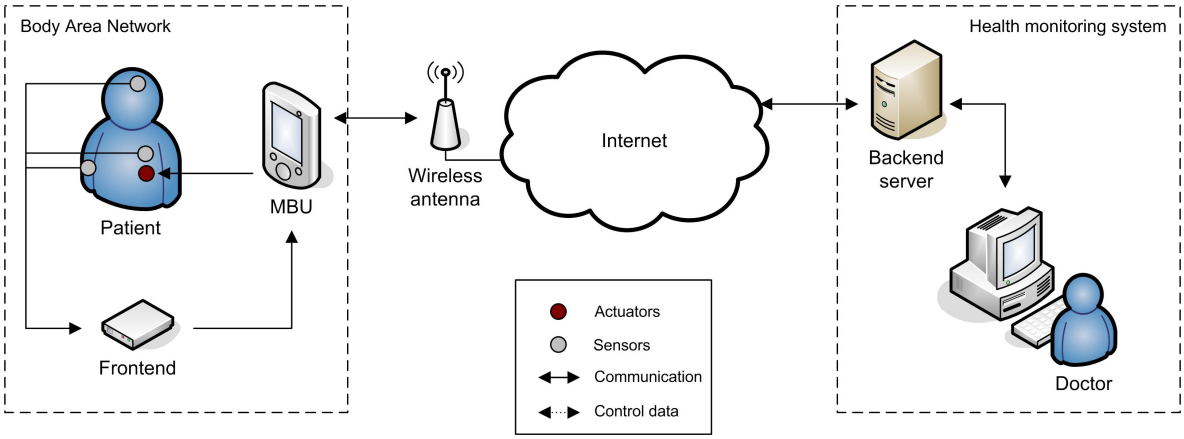


Figure 1.1: Schematic representation of the Mobihealth infrastructure [6].

Patients that are equipped with sensors, actuators and an MBU, are free to move around. While the patient is roaming, the environment of the MBU will change: e.g. the quality of a wireless communication channel may change or a new channel becomes available. The lack of a control mechanism makes it impossible for the MBU to adapt to these changes. When, for example, the communication channel that is in use, is no longer available, it is not possible to select another communication channel. This results in captured vital signs from sensors that cannot be transmitted to the health monitoring system and thus the remote monitoring of a patient comes to a halt.

A second problem is the changing of user settings while the MBU is in use. Users specify preferences towards the quality of the sensor information and the used communication channel. These preferences are translated to settings and specified towards the MBU before any sensor information is transmitted. When users want to update their preference, the MBU cannot adapt, because it cannot control the flow of sensor information that it forwards and thus cannot process any changed settings.

To provide a solution to the above problems a mechanism is needed to control the behaviour of the MBU. This control mechanism needs to influence the behaviour in such a way that the MBU adapts to preferences of the users and changes in the environment.

In our work, we follow a policy-based approach and propose a policy-based control mechanism. Such a mechanism uses policies to influence the behaviour of the system it controls. In this thesis we focus on how a policy-based control mechanism can facilitate the MBU in matching

the sensor information and capacity demands against the available communication channels and selecting the correct settings.

The above problem description results in the following research question:

How should a policy-based MBU control mechanism, that supports environment changes and user preferences, be specified and implemented?

The following section describes the approach that is used to find a solution to the described problem.

1.2 Approach

A policy-based control mechanism consists of two parts: policies and the architecture of a policy-based control system. Policies are rules on which decisions are based and the architecture facilitates the evaluation of the policies.

Literature states several methods on how policies can be defined and specified. Examples are XML policies [7], Ponder [8] and IETF policies [9, 10]. In this thesis we will follow the concept of the IETF policy-based control and use their model as a starting point.

The IETF policy model is mainly used in the network management and security field. However, we will apply the IETF policy model to an application, the control system of the MBU. We will add some new components and concepts to the IETF policy model. The IETF policy-based control model decomposes a system in two entities: the controlled entity and the controlling entity. The controlling entity enforces its decisions onto the controlled entity, which adapt its behaviour based on the decisions. The controlling entity is using policies, which are defined as 'if-condition-then-action' expressions, to come to a decision. The decision is specified as actions.

There are several possibilities how a policy-based system can be implemented. Previous research showed that the use of a class loader to dynamically load policies, which are based on the IETF policy model, can improve the usability of a policy-based system [1]. We follow current implementation developments and will design an architecture which uses services. Such an architecture puts atomic functionality of a program into a service. Services can communicate with each other to form a program. The Open Standard Gateway initiative (OSGi) specified a framework, which usage is growing rapidly, will be used as a base for the architecture. This java-based framework offers a standardized structure which enables the easy and fast development of programs, and abstracts their inter-communication.

1.3 Outline

This thesis consists of nine chapters, where we describe our work and the research issues that have been dealt with. The structure of this thesis is the following:

- Chapter 2 describes an M-health application, Mobihealth, for which a policy based-control system and policies will be designed. It also includes a list of requirements which have to be incorporated in the control system.
- Chapter 3 discusses the concepts around policies and policy-based control based on literature.
- Chapter 4 describes the OSGi framework and the components of this framework.
- Chapter 5 describes the specification of the policies. This section also provides examples of policies.
- Chapter 7 presents the architecture of the policy-based control mechanism.
- Chapter 8 describes the implementation of the policy-based control mechanism in the OSGi framework. It also includes examples of the implementation.
- Chapter 9 gives the conclusions and recommendations of this thesis. Possible future work is also described in this chapter.

Chapter 2

Mobihealth: an M-health application

In this chapter we derive the design requirements of the control system based on thorough analsis of an M-health application: Mobihealth. The chapter consists of the following parts: first an overview of the Mobihealth system will be described, including the goal and the global infrastructure. Second, each of the Mobihealth components, including their inter-communication, is explained. The third part will go in-depth into the bandwidth availability of the communication channels types and the bandwidth requirements of the Mobihealth system. The fourth part will describe the requirements of the policy-based control mechanism. The fifth part will give a summary of this chapter.

2.1 Overview

The Mobihealth project is a mobile healthcare project initiated in 2001 that aimed at designing a mobile environment that enables patients to be monitored while not being confined to a hospital. The project has been the work of fourteen partners, including the University of Twente, and was sponsored by the European Commission. Further work on the infrastructure of Mobihealth is currently performed in the AWARENESS project[1].

The initial goal of the Mobihealth project was to evaluate the ability of 2.5G and 3G[2] communication technologies to support innovative mobile health services. The main delivery of the project was an assessment of the suitability of GPRS and UMTS to support such services. Initial tests showed that these technologies were usable for mobile health services [5].

[1] AWARENESS: Context AWARE mobile NEtworks and ServiceS: `http://awareness.freeband.nl/`

[2] 2.5G and 3G: the 2.5 and 3rd generation wireless telephone technology.

The Mobihealth infrastructure is shown in Figure 1.1. It consists of two parts: the Body Area Network (BAN) and a health monitoring system. The BAN consists of a number of wireless sensors, actuators, a frontend and a Mobile Base Unit (MBU). The sensors and actuators are placed on the body of a patient. A wide variety of sensors and actuators is available like a heartbeat monitor, respiration monitor, GPS information, etcetera. The sensors are battery-powered and are connected to a frontend which converts the sensor information to the desired digital format. The frontend and the actuators communicate with the Mobile Base Unit (MBU) through Bluetooth or Zigbee. The health monitoring system consists of a backend server and the terminal of a doctor. The backend server receives and stores sensor data. A doctor can access and retrieve the sensor data. If the doctor wants to activate an actuator, it can send a message to the backend server. The backend server forwards these messages to the BAN.

The MBU collects data from the sensors and sends requests to the actuators. It is also used for local processing of data and acts as a gateway for the external communication [5]. The communication between the MBU, sensors and actuators is called intra-BAN communication while communication between the MBU and the healthcare center is called extra-BAN communication [11].

Trials of the Mobihealth project have been running in several countries including The Netherlands. At each trail several sensors were used for different medical tests. The results of the trials looked promising and users of the Mobihealth system stated that a fully functional product would be very useful [12]. The results also indicated that healthcare professionals, e.g. doctors, find it difficult to specify user requirements of M-health services in terms of possibilities of new technologies [13]. This resulted in a best-effort implementation without the specification of quality of service. This best-effort implementation does not enable users to specify any preferences and thus any quality of service. The control system that will developed in this thesis will enable the specifying of preferences and thus quality of service.

To make the overview more clear, the components of the mobihealth system and their interactions, will be explained in more detail. The next section will describe these components. Section 2.4 will discuss the communication between these components.

2.2 Mobihealth components

There are three main groups of components: the health monitoring system, the sensors and actuators and the MBU. Each of the components will be explained in the following paragraphs.

2.2.1 Health monitoring system

The health monitoring system receives information about the health of the patient from the MBU. This information contains the data from the sensors that are worn by the patient. This data is used to monitor the patient. The monitoring can require certain actions taken by a doctor in the healthcare centre. For example, the patient has to take more medicine or an ambulance has to be send to bring the patient to a hospital. The interpretation of the sensor information is outside the scope of this thesis.

2.2.2 Sensors and actuators

Mobihealth receives information from the patient by making use of sensors and actuators. A sensor is a device that captures information from the patient, for example the hearth rate. An actuator is a device that can be controlled by a doctor in the healthcare centre, for example an electromechanical device like a pacemaker.

As described in Chapter 1, the focus of this thesis is the development of policies and a policy-based control system. This sytem will control the flow of sensor and actuator information. So it is important to know the functioning of these devices in order to control their flow of information. Currently a wide variety of sensors and actuators are available. To develop the system, it is not required to know the functioning of every sensor and actuator. Only the amount of generated information has to be known to be able to develop the control system. Still it is good to know some extra information concerning a.o. the type of sensor to get a good view of what kind of information flows through the Mobihealth system. Currently there are very much sensors and actuators available. Therefore a subset of these devices will be described and used in the design process. To make this subset as representative as possible, the devices that are mostly used in the Mobihealth trials will be included. The next paragraphs will give an explanation of the functioning of each of the most used sensors and actuators.

Electrocardiogram (ECG) An electrocardiogram will be made when a doctor wants to record the bioelectrical signal that is generated by the heart of a patient. Bioelectrical signals of the heart consists of various individual signals depending on the position of sensors on the body of the patient. A standard method has been developed where and how the sensors should be placed on a patients body. A pair of sensors transmit one signal and is called a lead. The standard method states that there are twelve possible leads. A doctor decides how many leads he wants to see depending on what kind of medical measurement is performed [1].

Pulsoximeter (SPO_2) The oxygen saturation (SPO_2) and the heart rate of a patient can be measured with a pulsoximeter. The oxygen saturation is the amount of oxygen that is attached to the hemoglobin in the patients blood and is expressed as a percentage. The hearth rate is the number of times the hearth beats per minute [1].

Blood pressure (NIBP) The blood pressure or 'Non-Invasive Blood Pressure' (NIBP) is used to measure the amount of force applied to the walls of the arteries when the heart pumps blood through the body. The amount of blood pumped through the arteries and the force on their walls determine the pressure. The blood pressure sensor that is used in the Mobihealth system transmits the upper and the lower blood pressure [1].

Respiration The respiration is the number of breaths a patient takes per minute including the amount of air in- and exhaled. The measurement of the amount of used air is also called spirometry. This information can be measured by analysing the amount of expansion of the chest cavity. This expansion is caused by the in- and exhaling of air. A spirometer can calculate the amount of air in- and exhaled from the expansion of the chest [1].

Position To be able to track down a patient or record the activity or movement, a position sensor can be used. As a positioning system, the Global Positioning System (GPS) is used. GPS uses 30 different satellites which are spread across the earths atmosphere. A GPS receiver needs at least 4 GPS signals to determine its position. When a receiver is used outside with no buildings or trees nearby, up to 12 satellites can be spotted, but when inside a building the number of visible satellites depend on the type of used GPS receiver. Inside a big reïnforced steel building almost no signal is received. The calculated position can have some errors due to atmospheric disturbance and can vary from 0 to 5 meters [14].

Button An alarm button can be pressed by a patient to signal a healthcare center if there is a problem. The same button can also be used as a marker button by the patient to mark certain events. For example, when the patient feels dizzy, a mark can be send to the healthcare centre, so an analysis of the vital functions can be made by a doctor. The signal of the button does not have to be transmitted as a continuous signal, but there has to be verified that the signal is also received by the healthcare centre, because certain actions may be required, like sending an ambulance [15].

2.2.3 MBU

The MBU is the central component of the mobihealth system. It receives data from the sensors and transmits these to the healthcare centre. It also send messages received from the health monitoring system towards the actuators. The main task of the MBU is to make sure that all data is transmitted to the health monitoring system. When the speed of the outgoing connection is lower than the size of the data from the sensors and actuators, it can cache the data for a limited amount of time.

Hardware The main hardware component of the MBU is a PDA. Several types have been used like the iPAQ 3870 and the Qtek 9090 [16]. Currently the Qtek 9090 is used. This device is a smartphone which includes a 400 MHz processor and 140MB flash memory and it weighs 205 grams. Extra memory can be added using a MMC/SDIO memory card. The maximum time this device can be used for making phone calls is 270 minutes. The standby time, the time that the device can be operational but not in use, is 160 hours . Communication with the device can be done through the touchscreen or through one of the available wired or wireless connections. The available wired connection is USB and the available wireless connections are Bluetooth, IRDA (infrared) or WLAN. Other connections, e.g. GPRS and UMTS, can be made available using the memory slot [17].

Software The software that is used on the Qtek 9090 consists of two parts: the operating system and the MBU application. The operating system is Windows Mobile. The MBU application is written in Java and as a java runtime environment J9 from IBM is used.

2.3 Component interactions

The communication between components of the Mobihealth project is separated in two different parts. In Figure 1.1 the communication between components inside the BAN is called intra-BAN and the communication between a component inside the BAN and a component outside the BAN is called extra-BAN. The next two paragraphs will explain these two component interactions and the possible communication implementations.

Intra-BAN Communication between devices that are inside the BAN are wired and wireless. Analog sensors use a wired connection to connect to a front-end, the ‘mobi’. The front-end converts the analog data to a digital signal and transmits this using a wireless

connection to the MBU. Bluetooth[3] and Zigbee[4] are the possible wireless technologies that can be used between the front-end and the MBU.

Extra-BAN The communication between the healthcare center and the MBU is based on 2.5G and 3G communication protocols. Currently General Packet Radio Service (GPRS), Universal Mobile Telecommunications Syste (UMTS) and Wireless Fidelity (WiFi) are supported. The current system does not provide handoffs between different network connection type while in use. A connection type has to be selected and configured, before the MBU can transmit data. If the connection that is in-use is not available for a short period of time, the MBU caches the sensor data until the connection is set up again [11].

The extra-ban communication is based on the IP protocol which enables it to use new wireless transmission technologies when they are available. The security and reliability of the transmitted data is done in the application layer above the IP protocol [18].

2.4 Bandwidth demands

The MBU sends and receives data from the healthcare center. The amount of data that will be transmitted depends on the number of used sensors, compression of data and the available bandwidth. This section will provide a theoretical estimation of the bandwidth that is required by the sensors and the available theoretical bandwidth of the wired and wireless technologies that are used.

2.4.1 Available bandwidth

The available bandwidth of the Mobihealth system can be divided into two categories: the intra-BAN and the extra-BAN bandwidth.

Intra-BAN

Zigbee and Bluetooth are supported as intra-ban network connections. Currently only Bluetooth is implemented. Bluetooth has a number of specifications. The theoretical maximum speed of specification 1.1 is 721 kbps (download) and 58 kbps (upload) in asymmetric mode and 433 kbps in symmetric mode. A newer specification, 2.0, has a higher maximum total speed of around 3 mbps [19].

[3] http://en.wikipedia.org/wiki/Bluetooth
[4] http://en.wikipedia.org/wiki/Zigbee

Extra-BAN

The possible network connections for extra-ban communications are GPRS, UMTS and WiFi. Below an overview of the available bandwidth of each connection will be described.

GPRS GPRS is a 2G communication protocol that was the first widely available IP based service. It is currently exceeded by the UTMS protocol. GPRS supports different encoding schemes; CS-1 to CS-4. CS-1 guarantees a high fault tolerance but a low transmission rate and CS-4 offers almost no fault tolerance but a high transmission rate. Practice shows that the lowest coding scheme can only be used when a device is stationary and very close to a base station. In most situations CS-1 and CS-2 are being used. GPRS uses shared timeslots which makes it possible for a user to use up to eight time slots at once. A timeslot can be used to download or to upload data. The shared design of the current GPRS network equipment make it almost impossible to always use eight timeslots at once. Due to hardware limitations most of the time the number of timeslots per base station is limited to 4 [20]. Table 2.1 gives an overview of the theoretical available bandwidth of GPRS.

Timeslots	**1**	**2**	**3**	**4**
CS-1	9.1	18.1	27.2	36.2
CS-2	13.4	26.8	40.2	53.6
CS-3	15.6	31.2	46.8	62.4
CS-4	21.4	42.8	54.2	85.6

Table 2.1: GPRS transmission rates (in kbps) [1].

The theoretical maximum transmission rate is 8 timeslots * 21.4 kbps (CS-4) = 171.2 kbps, but as stated before, this will almost never be available. It is better to assume a CS-2 coding and 4 timeslots as a realistic maximum throughput which will result in 53.6 kbps [21]. The Mobihealth specification states that the maximum GPRS speed of the MBU is currently 28.8 kbps for uploading and 43.2 kbps for downloading [11].

Currently GPRS has 99.5% coverage in The Netherlands. No further improvement of the GPRS coverage is expected, because the main focus of telephone companies is on UMTS deployment.

UMTS UMTS, became available in a few countries in the late 2003 and is currently being deployed in several countries. The theoretical maximum transmission rate is 1920 kbps, but this will only be possible when the mobile device is close to an antenna. When in a rural area with low antenna coverage, the maximum transmission rate can drop to 144 kbps. In a busy city, the transmission rate will be around 384 kbps [22].

UMTS does not have full coverage in The Netherlands. Currently it is operational in the big cities, but not yet in rural area's. It is expected that at the end of 2007 the UMTS coverage will be around 99%[5],[6].

WiFi WiFi is a name that is used to describe the specification of wireless Local Area Networks (LAN) based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standard. Currently three versions of this standard are used. The main difference between them is the number of used (non-)overlapping channels and the transmission power. In table 2.2 the maximum theoretical transmission rate of each standard is listed [23].

Standard	**Rate**
802.11a	54
802.11b	11
802.11g	54

Table 2.2: WiFi transmission rates (in mbps).

Currently the 802.11b and 802.11g networks are used in most situations. The 802.11a standard is not used very much, because it uses the 5Ghz band which makes the need of visibility between antennas necessary. The other standards use the 2.4GHz band and do not need visible contact between antennas. The range of an 802.11a antenna is also much lower than the 802.11b and g antennas which enables the need of more antennas if full coverage needs to be accomplished [24].

There is no large coverage of WiFi in The Netherlands. A lot of people have broadband internet at home including a WiFi connection, but outside their house, no coverage is available. Some universities offer WiFi connections for their employees and students. On busy train stations hotspots are available which can be used if the user has a subscription. Private initiatives have been launched to offer free WiFi connections in certain area's[7].

2.4.2 Required bandwidth

As stated earlier, the required bandwidth should also be known to the control system. In the next paragraphs a short explanation of the transmission rate of each of the selected sensors and actuators will be given. The measurement and calculation of the transmission rates of some sensors has been researched by Tian [1] and has not been performed again.

[5]KPN UMTS coverage: `http://www.kpn.com/kpn/show/id=716554`

[6]Vodafone UMTS coverage: `http://www.vodafone.nl/prive/klantenservice/netwerk_en_dekking`

[7]Stichting Wireless Leiden: `http://www.wirelessleiden.nl/`

Electrocardiogram In the Mobihealth system, ECG leads are digitized before they are transmitted towards the healthcare centre. In the digitizing process, some leads can be combined together in channels to reduce the transmission rate. Every analog lead is sampled before it is transmitted. The sampling rate can be varied between 128Hz and 1024Hz. Each sample has a size of 24 bits. Depending on the demands of a doctor, one or more channels are transmitted. Table 2.3 shows the transmission rates for the number of channels and the sampling rate.

	Sampling rate			
# channels	**128Hz**	**256Hz**	**512Hz**	**1024Hz**
2	6.1	12.3	24.6	49.2
3	9.2	18.4	36.9	73.7
4	12.3	24.5	49.2	98.3
5	15.4	30.7	61.4	122.9
6	18.4	36.9	73.7	147.5
7	21.5	43.0	86.0	172.0
8	24.6	49.2	98.3	196.7
9	27.7	55.3	110.6	221.2

Table 2.3: ECG transmission rates in kbps [1].

Pulsoximeter (SPO_2) The Mobihealth system samples the SPO_2 values with a rate of 16Hz, 32Hz or 128Hz. One sample has a size of 24 bits. In table 2.4 the transmission rate of one pulsoximeter is shown.

	Sampling rate		
# channels	**16Hz**	**32Hz**	**128Hz**
1	0.384	0.768	3.702

Table 2.4: Pulsoximeter transmission rates in kbps [1]

Blood pressure (NIBP) The blood pressure values are sampled once per second (1Hz) with a sample size of 24 bits. The transmission rate is then calculated as 0.006kbps.

Respiration The Mobihealth system uses a sampling rate of 16Hz or 64Hz for the respiration sensor. Only one sensor is used for the measurement. One sample of data is 24 bits long. The transmission rate can then be calculated as 0.39 kbps for 16Hz and 1.54 kbps for 64Hz sampling rate.

Position A GPS coordinate consists of a longitudinal and latitudinal position, i.e. S60°42'036, W61°06'182. This results in a bit size of 29 bits or 4 bytes. It is enough to transmit a GPS coordinate with a frequency of 1hz, which results in a bandwidth requirement of 4bps.

Button The size of the button signal could be 1 bit, but sometimes the current position will also be transmitted. It is best to use 4 bytes for the position and 1 byte for the alarm message, so the required bandwidth will be a maximum of 5 bytes. The transmission of the signal is non-continuous.

2.4.3 Bandwidth summary

Table 2.5 shows the bandwidth that is required for the Mobihealth sensors when then highest sample rate is used. The ECG data has a very high impact on the bandwidth requirements of the Mobihealth system. If these are not required by the healthcare center for monitoring, the total required bandwidth is much lower and GPRS can be used more often.

Sensor	Bandwidth
ECG	221.2
Respiration	1.54
Blood pressure	0.01
Pulsoximeter	3.7
Button	0.01
Position	0.01
Total	226.47

Table 2.5: Required bandwidth (in kbps)

Connection	Bandwidth
GPRS	53.6
UMTS	384
WiFi	3072[a]

Table 2.6: Available bandwidth (in kbps)

[a]The WiFi bandwidth is limited to the bluetooth speed of the intra-ban-communication

Table 2.6 shows the total amount of bandwidth that is available in most standard situations for the discussed network connection types.

It can be concluded from the required and available bandwidth that it is not possible to always transmit all the required sensor data with the available bandwidth. Thus a selection of sensor data, or the granularity per sensor has to be adjusted to the network situation.

2.5 Requirements of the control system

Based on the analysis that is performed in this chapter and on the problem description, described in Section 1.1, we can formulate requirements which have to be met by the control system that will be designed. The following paragraphs will elaborate the requirements.

The current design of the MBU states that it is a wireless and portable device that is worn by the patient. The wireless and portable functionality makes the MBU limited in its capabilities. The control system should not be resource consuming else the battery of the MBU will wear out too fast.

Chapter 1 described two types of input which need to be used by the control system. These types are information from the environment and information from the users of the system. The analysis describes two different users: the doctor and the patient. They need to be able to state preferences towards the control system.

The goal of the control system is delivering a certain quality of service to the users of the system. The doctor wants to have a minimum level of granularity of the sensor data and the patient wants to influence the selection of the connection between the MBU and the healthcare centre.

Summary The previous paragraphs can be summed up into the following list of requirements.

- The policy-based control system has to control the forwarding of sensor information based on the MBU's environment;
- The policy-based control system needs to adapt the MBU based on the preferences of the doctor and the patient;
- The control system needs to be lightweigthed.

2.6 Summary

In this chapter we have described the concepts and model of the Mobihealth system. The components and their interactions have made clear which part of the system should be controlled by a policy-based control system. The communication demands of the entities have been described including a detailed list of bandwidth demands. We also described a set of requirements for the control system that will be designed.

Chapter 3

Policies and policy-based models

Chapter 2 described the Mobihealth system and clarified the problem description. As described in chapter 1 the solution of the problem will follow the IETF approach thus a policy-based control system will have to be modeled and specified.

Before design and specification of a policy-based control system can be described, it should be clear what such a system and a policy exactly is. This is accomplished by analysing literature.

The analysis consist of three parts. First an overview of policy-based systems and the area's where they are used, is given. Next the definition of a policy, its representation, properties and structure will be described. Models, which use the policy, are explained to clarify how a policy is used in a policy-based system. The last part of this chapter maps the described policy model onto the Mobihealth system architecture.

3.1 Introduction

The rise of policy-based systems is inspired by the need to flexibly manage network technology. The various configuration options and changing network topologies have driven the need of a method of configuring the network technology without interference of a manager. A possible method that is gaining support from standards bodies (like the IETF) over the last years is policy-based management [25]. Policy-based management is still under research and is being used in a lot of different networking areas.

In policy-based control systems, a set of decision making technologies are integrated into the management or control components of these systems [26]. An advantage of this approach is the simplification of the complex task of managing or controlling systems that have to adapt to their context. Only the goal of the managing has to be specified inside a policy and not

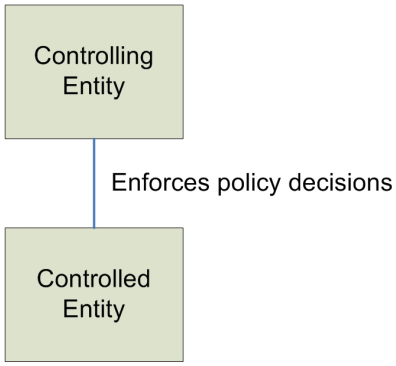


Figure 3.1: Controlling and controlled entity

the underlying detailed instructions how to execute the policy. Another advantage is the improved scalability and flexibility for the management system. Scalability is improved by uniformly applying the same policy to large sets of devices and objects, while flexibility is achieved by separating the policy from the implementation of the managed system. A policy can be changed dynamically, thus changing the behavior and strategy of a system, without modifying its implementation or interrupting its operation [27]. Policies split the behavior of the system from the systems implementation [25]. A visualization of this can be seen in Figure 3.1. The controlling entity enforces certain policies onto the controlled entity.

The last years a policy-based approach is also being used in distributed systems. Tanenbaum et al [28] state that before policy-based systems were introduced into distributed systems, the components of the whole system were only logically separated, but implemented as one, huge program. This made it hard to replace or adapt a component without affecting the entire system. Nowadays distributed systems are split in two, separating the policy from the mechanism which has to execute the policy. In the future it is expected that a separation between the policy specification and the policy-based control system will be implemented, so a user can specify his own policy in the form of a component that can be plugged into the controlling entity.

Another example of a policy based system is described by Wang et al [29]. The paper describes that switching seamlessly between networks is still not fully possible. A model is proposed which uses policies to describe which network should be chosen in what kind of situation. In a policy a user can make a trade-off between network characteristics, costs, performance, power management, etcetera. The described example is similar to the problem that has to be solved in this thesis. A clear definition of a policy has to be stated, before further similarities can be given.

3.2 Policy definitions

Policies are implemented in many different technologies and a lot of them use a different definition. Even in computer networking, different definitions are used in different areas.

The Internet Engineering Task Force (IETF) tries to produce high quality engineering documents that influence the design and usage of the internet. They have released several RFC's[1] covering policies and policy-based systems. The definitions that they gave to the word 'policy' are:

- The combination of rules and services where rules define the criteria for resource access and usage [9].
- A definite goal, course or method of action to guide and determine present and future decisions [30].
- A set of rules to administer, manage, and control access to network resources [30].

Published papers, that cover policy-based systems, use other definitions of the word 'policy':

- A statement about a systems behavior that will be translated and then incorporated into the system to change its behavior at run time [25].
- A rule that can be used to change the behavior of a system [8].
- Declarative rules governing choices in a system's behavior [31].
- Rule governing choices in behavior of the system. Change system behavior without modifying implementation and permit adaptable systems [32].
- A rule that describes Actions to be taken when certain Conditions happen [33].
- A rule which describes obligated actions to be taken once its condition is satisfied whenever its events happen [1].

A definition of the word 'policy' which fits the needs of this thesis but also incorporates some of the ideas of the above definitions is:

> *A rule which describes actions to be taken when conditions are satisfied.*

The definition states that a policy consists of conditions and actions. An *action* determines the desired behavior of the controlled system enforced onto the controlled entity. A *condition* states which prerequisites have to be met before the desired action may be enforced.

Literature specifies more elements than the condition and action rules, for instance policy targets (entities where actions are enforced), policy subjects (entities where conditions are

[1] RFC: *request for comment* documents are a series of memoranda encompassing new research, innovations, and methodologies applicable to internet technologies.

evaluated), triggers (entities that initiate the evaluation of the policy conditions) and goals (the intention strived by the policy) [33]. In the model that will be developed, the controlling and controlled entities are both physically on the MBU, and the MBU is the only entity of our design concern. This removes the need of defining a subject, trigger and target in the design of the policy control system.

In terms of the mobihealth system, conditions of a policy can be seen as e.g. a refinement of the granularity of sensor data, a new wireless network connection appearing or the patient unable to afford an expensive connection. Actions can be viewed as enforcing the system to transmit less data or the selection of another wireless network connection.

To go deeper into the definition of a policy and the actions and conditions that are required, its representation will be researched in the next section.

3.3 Policy representation

Sloman states that it is important that a good policy representation type is chosen, because an overkill just makes the burden on system developers, policy maintainers and policy selectors very high. Therefore, in this section a description of the possible policy representations is given [32].

The amount of behavior of the system that can be modified with a policy is limited by the representation of the policy. The amount of possible changes depends on the need for control of aspects of the system. According to Cox representation depends on the level of abstraction that is used. This level of abstraction can be accomplished by specifying a hierarchial policy structure. Figure 3.2 shows the hierarchical policy model as described by Cox and INTAP [25, 34].

The highest level of abstraction is *human-friendly*. On this level, a policy is described in a form that can be easily read and understood by humans. It does not describe the network equipment details like the lower level network interface. An example of this level is: *management must do ensure the performance on the network.*

Equipment-independent is the hierarchy level which is independent of the used hardware and is not end-user-friendly to read. It specifies which rules correspond with the human-friendly rules and have to be enforced on hardware level. An example of this level is: *bandwidth management must on Util > 70% do increase VP by 10% on network.*

The third level, *equipment-specific*, is also not human-friendly to read, but now it is also specific for a type of equipment. An example of this level is: *[VP_tuner_01] must on Util > 70% do increase VP by 10% on lowQoS VPs.*

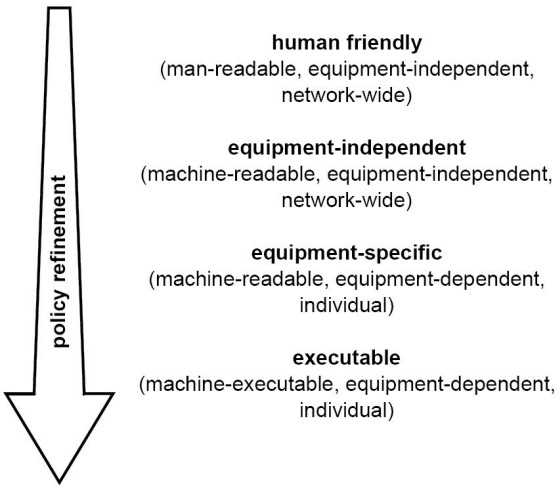


Figure 3.2: Policy refinement hierarchy

The lowest level, *executable*, contains the policy which can be directly executed on the hardware of the equipment. An example of this level is: *[VP_tuner_01] must on Util > 70% do increase VP by 10% on (VP1, VP2, VP3).*

The lowest level representation can be represented to the control system in various different ways. A basic representation is called rule-based and uses if-then conditions. Sometimes this is also called event-based [32]. The policy is static and is activated during some change in incoming information. An example of this representation is:

```
if (CONDITION == 1)
    do {ACTION 1}
else if (CONDITION == 2)
    do {ACTION 2}
else if (CONDITION == 3)
```

More complex rules, like nested conditions are possible, but are limited to the fixed set of variables known to the policy control system. It is not possible for a policy to extend the available fixed set of variables.

When more advanced behavior modification is required, like calculations on certain incoming information or probability calculation for the best output connection type, more complex policy representations are needed. A possible solution has been described by Tian [1]. In her model, the functionality of the policies is enhanced to provide a more advanced method of defining policies. An example of the more complex behavior of a policy is:

```
//if the costs are too high, switch to another network
if (0.5 * total_umts_time + 0.1 * total_gprs_time >= 15) {
```

```
    if (current_network == "umts")
        //Switching of network.
        switch_to_gprs;
}
else {
    //Switch to expensive network
    if (current_network == "gprs")
        //Switching of network.
        switch_to_umts;
}
```

In the example, the maximum price of the network traffic is defined as 15. Before the conditions are checked, some calculation on the current cost is performed. When the costs are too high, a cheaper network connection is selected. The action of the switching itself is delegated to an enforcement entity. It is possible to use more complex calculations to do, i.e. performance analysis of the network connection to get the desired and best result.

3.4 Policy properties

Policy based systems are widely used because of their good properties, however there are also some drawbacks. In this section we will summarize advantages and disadvantages of policy properties. The properties that are considered are persistency, flexibility, scalability, verification and conflicts. Also the possible types of policies are discussed.

Persistency Policies are called persistent, because the controlled entity is not only told what action should be done, but the action is enforced. The result of the action is also monitored during its operation. Basic control systems, like the if-then commands, look at a situation at startup, make a decision, take an action and stop controlling the system. When the environment of the controlled entity changes, no new decision is made. A policy based control system keeps gathering information from the environment and validates the enforcement of the current policy. If the environment changes too much, a new policy will be enforced onto the controlled entity. Policies affect the behavior of a system continuously until they are revoked or replaced by other policies [33].

Flexibility The flexibility of policies is the main reason why it is being used in a lot of different technological fields. Policies can flexibly the control system, because the system can adapt to dynamically changing environments. When new policies are installed into the system, new behavior of the controlled entity can be enforced. Three different forms of flexibility can be distinguished :

- *Adaptive behavior*: The actions that will be enforced, depend on the conditions that have to be met by the different environment changes;
- *Flexible control*: The conditions and actions can be fine-tuned at runtime. The behavior can thus be altered by updating the policy rules;
- *Flexible evolution*: New policies can be installed in the controlling entity which makes it possible to specify new behavior. This flexibility does not require the control system to be stopped when these policies are installed and enforced [33].

Scalability The scalability of a system is its ability to handle growing amounts of work in a graceful manner and its ability to flexibly change its size or capabilities. A policy based system is very scalable, because it is possible to administer more than one control system with the same policy repository. It is also possible to use one control system to control more than one controlled system, i.e. a doctor can specify settings for multiple MBU's at once. Another advantage of the scalability of policies is the separation between the controlling and controlled system. This enables the controlling system to be located somewhere else than the controlled system which enables the controlling system to easily manage more than one controlled system [35].

Verification Verification of a product makes sure that it satisfies the required or intended design, i.e. if the product is build according to the specification. In policy systems, verification is an important factor if users have to be able to specify new policies at run-time. The new policies have to be verified if they are specified according to the policy design. The controlling entity has to support the required policy specification, but the policy also may not use or specify variables that are not in the vocabulary of the control system. For example, in a mobihealth policy, it has to be checked that a specified sensor is available at the patients BAN. If a sensor is configured that is not valid, it has to be clear what action should be taken [36].

When using compiled java class files as a policy representation, verification is done before a policy is uploaded into the policy repository. The programmed policy has to be compiled and the java compiler does the type checking and verification of the source file. The compiler also checks if the required methods are present and if the arguments of these methods are valid.

Conflicts Conflicts in control systems occur when two or more entities want the enforcement of a different configuration. In policy control systems, this can occur when two or more policies are used which results in conflicting actions. When conflicting policies are inserted into the control system, inconsistency of the systems behavior can occur, because it is not clear which

policy is executed first or last and which actions will be enforced. The systems behavior can then look irregular and strange [37]

The control system should be designed in such a way that it can handle policy conflict situations and that it can decide which policy should be enforced and which one should not. When a policy conflict is detected, the control system should resolve the conflict situation and select the appropriate policy [38, 39]

Literature suggests some solutions as a solution or to prevent policy conflicts. Lupu states that a possible solution of conflict problems is to change policies in such a way that they do not overlap anymore. A disadvantage of this method is the drawback in functionality. Sometimes different policies want to be used which give conflicts when used at the same time, but work correctly when not used together [40].

In the policy information core model of the IETF another solution is proposed. When policies are developed, a priority should be added. If a conflict between two policies is detected, the policy with the highest priority will be used in favor of the policy with the lower priority. Policies which cover the same target system should never have the same priority assigned to them to prevent conflicts between policies with the same priority[10].

Types The Open Distributed Processing (ODP) standardization initiative [41, 33] has developed a framework which introduces three different types of policies: permission, obligation and prohibition.

- Permission policies contain actions which state that a certain behavior is allowed to occur, but this behavior is not obliged.
- Obligation policies contain actions which state that a certain behavior is required to occur, and thus is mandatory.
- Prohibition policies contain actions which state that a certain behavior is not allowed to occur. This is the negative form of obligation policies.

The policies that have to be developed for this master thesis require mandatory behaviour, thus obligation policies will be used. When an action is selected, it is required to occur. For example, when a new network connection is selected by the policy, the MBU is required to switch to this new network connection.

3.5 A policy structure

The structure of a policy defines how the elements of a policy are organized. The IETF have defined a standard structure of these elements in the IETF core information model [10]. In this information model a schema is defined which shows the relationship between elements of a policy. This schema is shown in Figure 3.3.

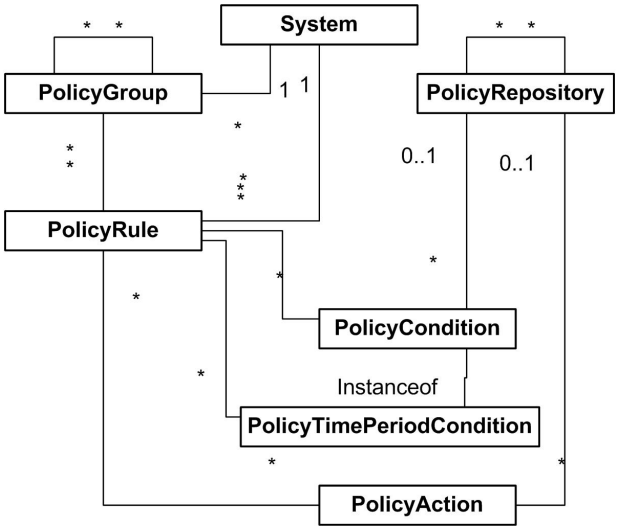


Figure 3.3: The IETF policy structure [10]

The boxes in the figure represent the elements of the structure and the lines represent their associations. Each association includes a cardinality of each of the connected elements. The possible cardinalities are *, *1* and *0..1*.

The main entity of the schema is the PolicyRule. This entity is the representation of the rule that is mentioned in our policy definition. A PolicyRule contains one or more PolicyActions and PolicyConditions. A PolicyCondition represents the conditions that have to be met before an action is enforced. A PolicyAction contains an action. A special condition is the PolicyTimePeriodCondition. This entity is an instance of a PolicyCondition and restricts the condition based on time. The condition is only valid in a certain timeslot.

The fourth association of the PolicyRule is with the entity System. This entity is considered a weak aggregation. A weak aggregation is used in the core information model to enable the designer to specify a naming scope. The association between PolicyRule and System thus only indicates that an instance of PolicyRule is named within the scope of a System.

The last association of the PolicyRule is with the entity PolicyGroup. A PolicyGroup may consist of zero or more PolicyRules or zero or more PolicyGroups. This enables the aggregation of PolicyRules that belong together.

Both the PolicyCondition and the PolicyAction may be present in a PolicyRepository. The PolicyRepository keeps a list of these possible entities.

A number of entities and associations contain properties. The entity PolicyRule has a property which states how the relation between the PolicyConditions of this rule are composed. It indicates if the associated conditions are in conjunctive normal form (CNF) or in disjunctive normal form (DNF). The association between PolicyConditions and their PolicyRule have two properties which make the association unique. The first property is an integer which partitions the conditions into groups and the second property is a Boolean which indicates if a condition should be negated.

The following example shows the use of the above properties. Suppose we have a PolicyRule that has an association with four PolicyConditions. The association between the rule and the conditions have the following properties:

```
C1: GroupNumber = 1, ConditionNegated = FALSE
C2: GroupNumber = 1, ConditionNegated = TRUE
C3: GroupNumber = 1, ConditionNegated = TRUE
C4: GroupNumber = 2, ConditionNegated = FALSE
```

If the PolicyRule has property DNF, the overall condition is:

((NOT C1) && C2 && C3) || (NOT C4)

and when the property is CNF, the overall condition is:

((NOT C1) || C2 || C3) && (NOT C4)

These conditions both specify the condition of a PolicyRule umambiguous.

In the described condition form, representing nested conditions is not possible. However it is possible to rewrite every nested condition to a DNF or CNF. When for example the nested condition ((A || B) && C) is specified, where A, B and C are single expressions, this condition can be rewritten as ((A && C)||(B && C)).

3.5.1 Extended IETF policy structure

The IETF policy structure, described in the previous section, does not incorporate a goal specification. It only lets designers specify a condition and an action. Bearden [42] defines an information model which extends the IETF information model and incorporates the goal specification. From the viewpoint of a system administrator, the IETF information model represent "what" needs to be achieved in terms of the system behaviour. When looked from the viewpoint of an end-user, these rules only specify "how" the desired behaviour can be achieved. "What" kind of behaviour is strived after is not clear for the end-user. The extention

Bearden proposes enables the end-user to specify "what" has to be achieved in terms of goals. It also specifies the relation between goals and policy rules and thus the relation between "how" and "what".

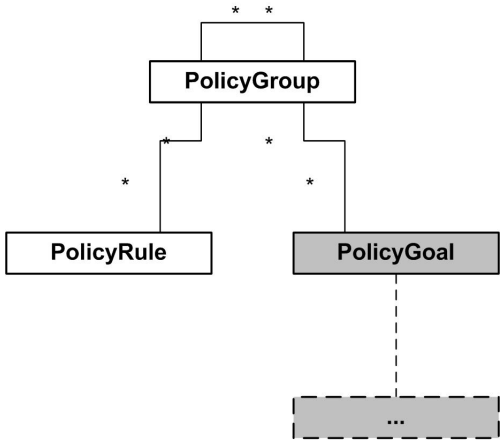


Figure 3.4: The extended IETF policy structure

The goal component in the extended information model adds extra entities to the structure model as can be seen in Figure 3.4. The new entities are coloured gray. The top level entity of the goal extension is the PolicyGoal. It has an association with the existing entity PolicyGroup. This extension enables the defining of zero or more goals to a group which already contains zero or more rules. This method enables the designers to specify the "what" next to the "how". The complete decomposition of the entity PolicyGoal will not be described here as the proposed decomposition by Bearden is too complex for the Mobihealth problem.

3.6 Policy-based models

As seen before, Figure 3.1 represents a high level schematic model of a policy system. It visualizes the basics of a policy based system. The controlling entity makes decisions based on policies and enforces these onto the controlled entities. This separation is on the logical level and does not have to be at the physical level.

A more extensive policy model has been specified by Lymberopoulos [36] and can be found in Figure 3.5. The controlled entity or system is kept as a cloud, because this can be any possible system as long as it enforces the selected policy. The controlling entity has been split up into a policy database, a system administrator, a monitoring and event service, and the controlling system. The policy database holds a list of the available policies. The system administrator can read the current configuration and set some variables at runtime. The monitoring and event service keeps track of information of the controlled system. The controlling system

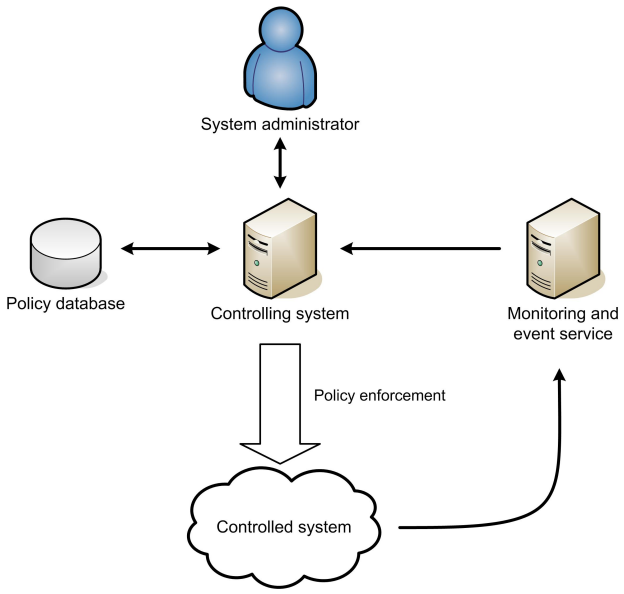


Figure 3.5: A policy model [36]

makes decisions based on the surrounding entities and policies, and enforces these onto the controlled system

The next part of this section will specify in-depth policy-based models from the IETF, distributed systems and an enhanced IETF model.

3.6.1 IETF

The IETF policy framework working group has specified a policy management architecture which is widely used in policy control systems [10, 9]. The main focus of the group is to specify a framework for providing policy based control over control decisions. For now their focus is on IP-based quality of service networks. The RFCs that they have published formalize the entities of Figure 3.5 and 3.1. The main architectural elements of the framework are the decision point and the enforcement point, which are located in the controlling and the controlled system. The decision point gathers information from resources and decides which actions should be enforced. An important note is that the framework does not any specific policy behavior or the use of specific policy types.

In Figure 3.6 the IETF framework is shown. It consists of four entities.

The definition of each of the entities of the IETF policy framework is explained in the next paragraphs.

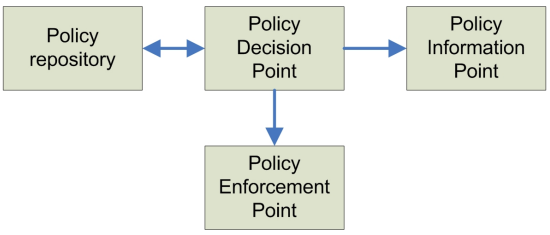


Figure 3.6: The IETF policy framework

Policy Decision Point (PDP) The PDP represents the entity that controls the policy system by deciding which behavior should be enforced by the Policy Enforcement Point. The PDP evaluates policies that are stored in the Policy Repository based on the information from the Policy Information Point. If the current system state, based on the policy evaluation, requires a change, the PDP decides to enforce a new action based on a policy.

Policy Enforcement Point (PEP) The PEP represents the entity that enforces the policies on the controlled system. The PEP receives decisions from the PDP and enforces these. The method how this is enforced is not specified in the model. Bi-directional communication between the PDP and the PEP enable both entities to revoke current enforced policies if this is needed.

Policy Repository (PR) The PR is a database which contains a list of policies. These policies do not all have to be active at a certain time. The PR allows the PDP to download policies at runtime when new decisions have to be made. The PR can be used for more than one policy control system at the same time.

Policy Information Point (PIP) A PIP is a service that gathers information from the surroundings of the control system that are needed for the PDP to make a choice which policy needs to be enforced. There can be more than one PIP in a system. Another commonly used name for PIP is Environment Monitor (EM). Examples for the types of possible information that will be gathered are: battery power, current network connection type and amount of memory left.

The location of the entities can be in one physical system, but it is also possible to separate them. For instance, the PDP could reside in a control system and the PEP could reside in a external system that has to be controlled. It is wise to at least logically separate the logical entities [28], to maintain a clear view of the control system.

Physical location of the PDP

The described IETF framework specifies how the components are logically separated and how they are connected to eachother. The framework also lets modelers decide what the physical location of the PDP should be.

The first option is to use an external PDP. The physical location of the PDP is not placed in the same physical location as the PEP as visualized in Figure 3.7. Policies are evaluated at the remote location of the PDP. The decisions are sent over an external communication channel to the PEP. An advantage of this option is the use of the PDP for more than one PEP. The PDP can make a decision that can be enforced at multiple PEPs. A disadvantage is the need for an external communication channel. If the PEP wants to receive a new decision and this communication channel is not available, no new decision can be made.

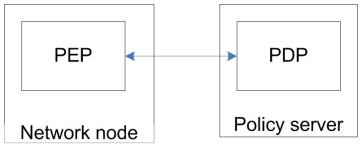


Figure 3.7: External PDP

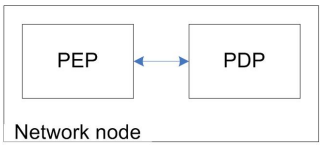


Figure 3.8: Internal PDP

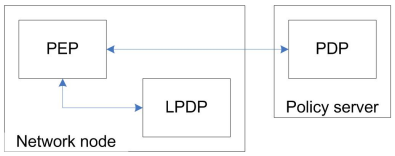


Figure 3.9: Combined PDP

Another possibility is the integration of the physical location of the PEP and the PDP. This is especially good when the PDP has to make a lot of decisions that have to be made locally. For example, when the communication channel is down very often. This option is visualized in Figure 3.8.

The third option is a combination of the other two methods. As shown in Figure 3.9, a local PDP, the LPDP, and an external PDP is used. The external PDP is the master PDP. Its decisions have priority over the decisions of the LPDP, but the external PDP can delegate certain decisions to the LPDP.

Communication

Communication between entities takes place using a Message Exchange Pattern (MEP). There are two major MEPs that are used widely: request-response and one-way [43].

Request-response is two-way communication. An entity send a request to another entity and always receives a response. This response can be send in a synchronous or asynchronous pattern. In a synchronous pattern the requesting entity keeps waiting until it receives a response. In an asynchronous pattern the requesting entity continues functioning. The response can be received at any later time.

In one-way communication an entity sends a message to another entity. The receiving entity hasnt send a request for the message. The message can be sned with or without guarantee. With guarantee the receiving entity will always receive the message and without guarantee it may receive the message. The guarantee does not state that the receiving entity is always listening for the message, thus the sending entity does not know if the receiving entity receives the message.

A widely used one-way communication architecture is event-based messaging. In event-based messaging an extra entity is in use as shown in Figure 3.10. When an entity wants to receive events from another entity, it registers itself at the Event admin. The Event admin holds a registry of all entities that want to receive a notification of an event. When an entity wants to send an event, it sends this to the Event admin. The Event admin looks in its registry which entities want to receive events from the sending entity and forwards the event to them.

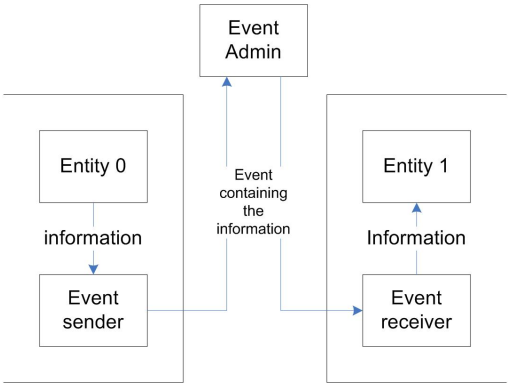


Figure 3.10: Event-based messaging

An advantage of the request-response MEP is the fact that no extra components are required for the communication. A disadvantage is the obligation to always send a response, even if this is not necessary for the functioning of the system. An advantage of the one-way communication architecture is the small message size. No information that is required for the response message has to be transmitted. A disadvantage is the unknown effect of a message. The sender never knows what the receiver did with the message.

The communication between the entities in the IETF framework is described in the next paragraphs.

PDP<−>PEP Both the PDP and the PEP can decide that a new policy decision has to be made [9]. When the PEP cannot enforce a decision that is received from the PDP, it will transmit an invalidation message to the PDP. The PDP will decide that a new decision has to be made, based on the PEP invalidation message or its own information. The new decision will be transmitted to the PEP which will enforce it.

PDP<−>PR When the PDP decides that new policies should be used instead of the current ones, it invokes a request and transmits this to the PR. This request states which policies the PDP want to receive. The PR will then send the requested policies to the PDP.

PDP<−>EM The PDP receives updates from the EM when environment parameters change. The PDP can also request information from the EM when it wants to know the value of a certain parameter. The EM will, when such a request is received, return the requested parameter.

3.6.2 Enhanced IETF

A shortcoming of the IETF model is the influence of users at run-time. No possible user-adjustment of configuration options is described in the IETF model. Tian described a model which incorporates user preferences into the IETF model to make it possible for users to state user preferences. These user preferences can contain information to change the behavior of the system. Theys can be adjusted at run-time to which makes system very flexible [1].

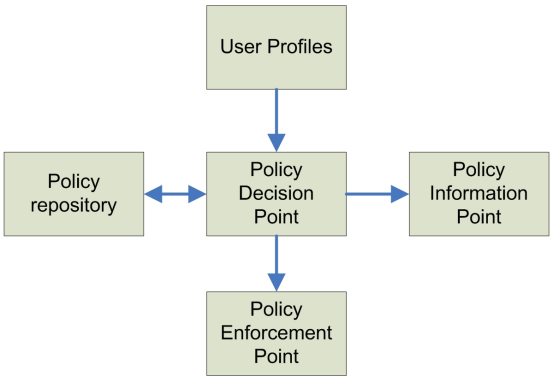


Figure 3.11: The enhanced IETF model

Figure 3.11 shows the enhanced IETF model. The communication of the EM, PR and PEP are the same as in the IETF model. Users can specify their preferences or profile at the User Preferences (UP) entity. The UP bundles preferences of multiple users and transmits these to the PDP. The PDP uses these preferences to make a selection which policies should be used in the decision making process.

User preferences

There are several possible methods on how the internal functioning of the user preferences component can be modeled. The following paragraphs will elaborate on them.

The first method is a lightweight component. The user preferences from the users are forwarded by this component to the PDP. The UP does not add or modify any preferences before they are forwarded.

The second method makes some calculations using the user preferences and adds extra information to them before they are send to the PDP. The mangling of the preferences include removing preferences conflicts and making use of certain criterions to analyze user preferences like adding priority information to each preference [1].

The third method uses a meta policy system. A meta policy system is a policy system on top of the normal policy system. This meta policy system uses policies to select which policies should be used in the normal policy system. This makes it possible for users to specify complex preferences which select normal policies based on, for example, a day or a certain time [1].

3.6.3 Alternative model

To give more insight into the policy system and how problems of self-management of systems are solved in other models, one alternative model is described: the feedback control model. This model will be explained briefly and compared to the enhanced policy model.

Feedback control

Another model which influences runtime variables and settings is the feedback control model. Feedback control systems have been used in various engineering fields and these systems are gradually finding their way into computing systems [44]. The basic idea behind such a system can be seen in Figure 3.12.

The core of the feedback control system are the managed entities; compared to the IETF model, the *controlled entities*. These entities have some initial configuration with added corrections received from the configuration point. The monitoring of the system is executed by the *metric estimator*, like the PIP in the IETF model. The analysis of the measured data is done by the *analyzer* (PDP) which also has some reference input data to use as comparison information. The results of the analysis can specify no changes are needed in the current enforced actions, but it is also possible that some of the current configuration values have to be changed. These changes are done in the *configurator* (PEP) and can include actions like adding an extra replica or switching a server.

3.7 Mobihealth and policies

The described policies and their model have to fit into the mobihealth system. The next paragraphs answer the research question which representation and model would be a good

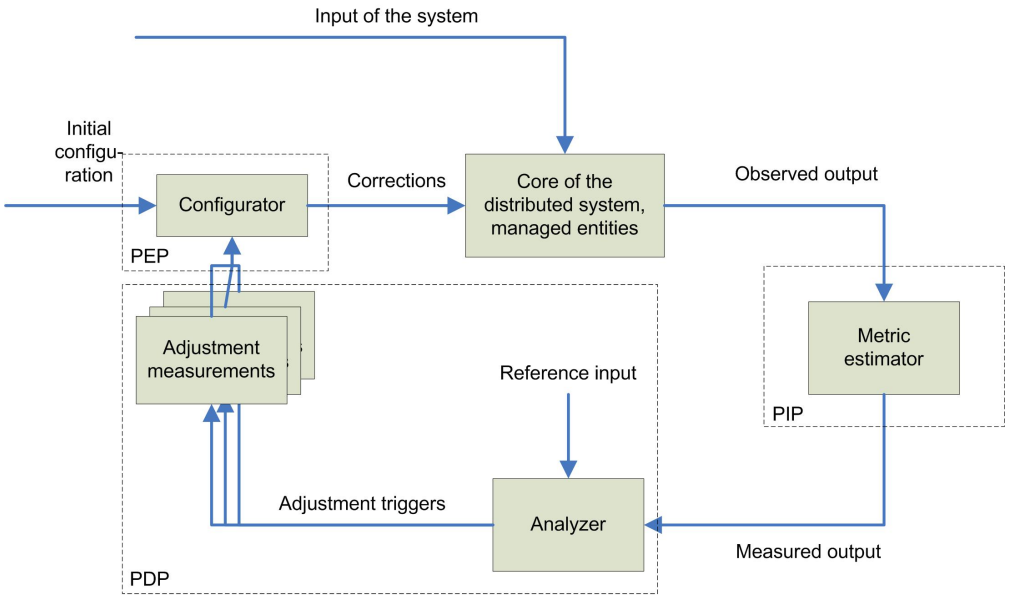


Figure 3.12: The logical organization of a feedback control system [28]

choice.

Representation The representation of a policy is very important, because too much functionality lays a high burden on policy developers and too less functionality makes the flexibility of the control system too limited. The users of the mobihealth system are currently not able to change any of the configuration of the MBU and its sensors. A policy based control system would be wise to incorporate into the system, because the environment of the BAN is constantly changing and the users of the system like to specify some constraints. They do not want to specify exactly what kind of policy they want to be enforced, but like to state rules like 'The granularity of the ECG is too low' or 'I only want to spend €5 per day'. A representation level with at least two levels is a best fit for this problem.

Mobihealth and the policy-based IETF model Before a policy specification and a policy-based architecture can be given, the policy-based IETF model should be integrated into the architecture of the Mobihealth system. As explained in section 3.1, a policy control model consists of a controlling entity and a controlled entity. Figure 3.13 maps the entities of Figure 3.1 onto the policy model and the mobihealth system. The controlled entity is the mobihealth system and the controlling entity is the policy control system (PCS).

If we look at the Mobihealth system at a lower level, we see that the MBU is the component that has to be controlled by the policy model. Figure 3.15 shows this refinement. The policy

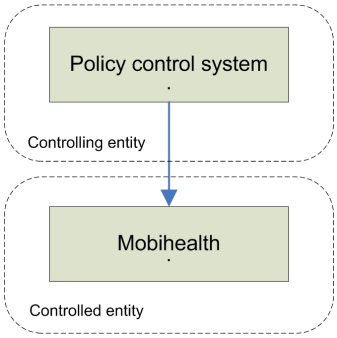


Figure 3.13: Policy model mapping

control system will make a decision and communicate this to the MBU. The MBU will have to execute this decision.

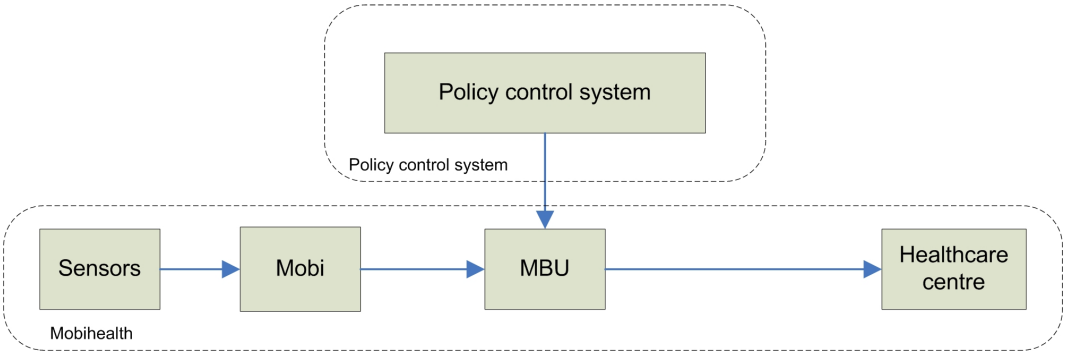


Figure 3.14: MBU refinement

If we refine the policy control model further, we can map Figure 3.11 onto the Figure above. This results in Figure 3.15. In this figure the five entities from the enhanced IETF model are mapped onto the policy control system and the MBU. The MBU has to execute the decision of the policy control system and thus the PEP is placed inside the MBU. The other four entities make the decision and are placed inside the policy control system.

3.8 Summary

In this chapter we have described the concepts and models of policy systems, based on literature study. We have seen that different policy representation, models and structures exist, each with their own goal and scope. Properties of policies have been described to explain the advantages and disadvantages of them. We also have showed how the IETF model should be integrated into the Mobihealth system.

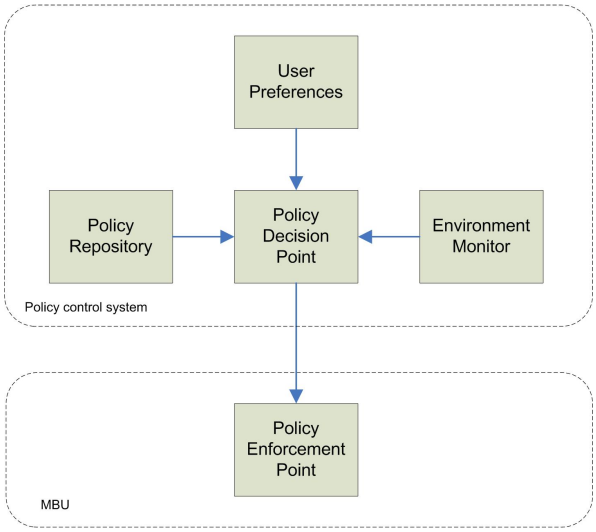


Figure 3.15: Lower level mapping

Chapter 4

The OSGi framework: a service oriented architecture

This chapter will give an analysis of the OSGi framework. An analysis of this framework is required, because the implementation of the policy and the policy architecture will be done in this framework. At the end of this chapter it should be clear what the OSGi framework is and how it can be used.

The analysis of the OSGi framework consists of several parts. Section 4.1 shows and describes an overview of the OSGi framework and its goal is described. Section 4.2 elaborates on each of the layers of the framework. Section 4.3 briefly describes the core services and the normal services that are specified next to the framework. Section 4.4 describes the different implementations of the OSGi framework that are available. Section 4.5 discusses briefly how policies can be modelled and implemented using the OSGi framework.

4.1 Overview

The Open Services Gateway Initiative (OSGi), currently known as the OSGi Alliance, have specified a framework which enables developers of software services to manage these in a standardized way. The mission of the OSGi Alliance is to enable the deployment of services over wide area networks to local networks and devices [45]. The OSGi framework tries to be platform independent. The inter-service communication is being made easier for developers by supplying a standard communication interface for each service. It is supported by a great number of companies and benefits from an active industrial and free community [46].

A service, called a bundle, can be installed, updated, started, stopped and deleted from the framework at runtime. Dependencies on other services can be stated including version numbers. A service can communicate seamless with another service even when they are connected with each other through a LAN or WAN. Services can publish their existence inside the framework which enables other services to find them in an easier, standardized way [46]. The standardized structure of a bundle has resulted in the exchangeability of bundles between different framework implementations.

An advantage of the OSGi framework is the use of only one java virtual machine (JVM) on each computer. Bundles are loaded inside the framework using the java class loader. This method lowers the overall load of the framework on devices that run the framework.

The OSGi Alliance does not release a fully working implementation of their specification, but only the core implementation interfaces and a test suite. The implementation is left to other companies or organizations. In section 4.4 a comparison of the available implementations is shown.

In Figure 4.1 the OSGi and system structure is shown. Each bundle interacts with the OSGi framework. It is possible for a bundle to interact with the operating system or the JVM, but this is not recommended. Multiple bundles reside inside one JVM which simplifies the communication between these bundles.

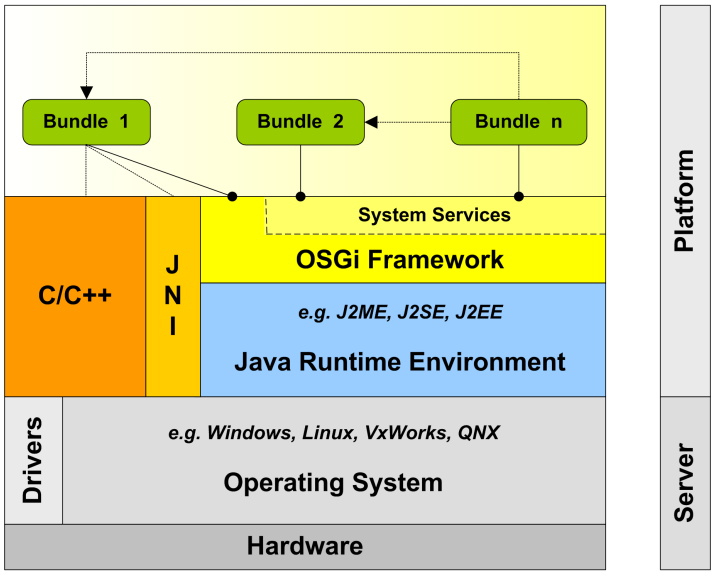


Figure 4.1: OSGi and system structure [47].

The rest of this chapter will elaborate on the framework model that is specified by the OSGi Alliance, explain the different layers, and elaborate on the different available core services. A list of the optional services is also given. A comparison of the various implementations will be

stated and the requirements on the MBU and the framework are given. Also an analysis on how policies can be implemented in OSGi is given. At the end of this chapter the conclusion is given which will state the best OSGi framework for the required policy model.

4.2 OSGi framework layer model

Figure 4.2 shows the layer model that is used by OSGi. The framework specification specifies the functions and purpose of each of these layers. As the Figure shows, the four layers work on top of eachother. Each installed bundle will make use of each of the four layers.

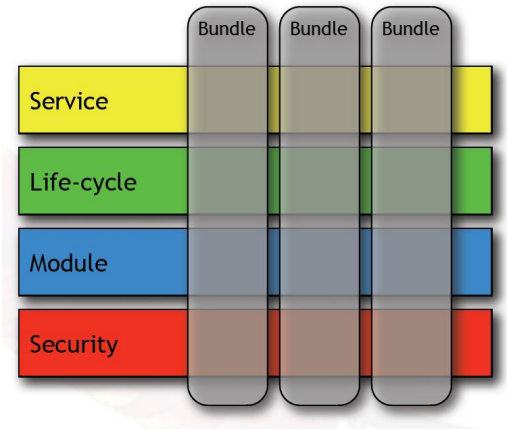


Figure 4.2: OSGi framework layer model.

In the next paragraphs an elaboration of the four different framework layers will be given.

Security layer The lowest layer in the framework is the security layer. It is similar to the Java 2 security model[1]. The security layer consists of two parts: the code authentication and the security model. The code authentication can be done according to the location of a bundle or by the signer of a bundle. A location can have a certificate which specifies its security level and a bundle can have a security certificate inside its package files. The permissions of a bundle are specified before runtime in a policy file and can be changed on runtime with a special service, the (conditional) permission admin. The OSGi specification states that the security layer is not required for a framework implementation; it is defined as optional [48]. This 'optionality' assures the lightweight of the framework. On resource-constrained locations the security layer can be left out.

[1]The Java 2 security model can be found at `http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-specTOC.fm.html`

Module layer On top of the security layer, the module layer is placed. This layer keeps track of the installed packages, bundles and native code. Some of these packages are exported so that other bundles can make use of them and some packages are private and only visible to a specific bundle.

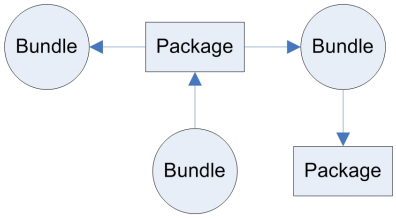


Figure 4.3: The module layer [49]

Figure 4.3 shows how bundles can depend on packages and packages can depend on bundles. A bundle can join and leave the framework whenever it wants to. Each bundle that depends on another bundle has to take precautions for the fact that other bundles can be removed at any time. The exported packages have a version number, so an importer can specify which version range of the package is required. The framework will check and resolve these dependencies at runtime. The advantage of this method is the protection of implementations; only API's or stubs[2] of packages are exported and thus visible and usable for other packages. Each bundle is loaded inside its own class loader, which ensures that no namespace conflicts will occur.

Lifecycle layer The lifecycle layer provides an API to control the life cycle and security of each individual bundle. The lifecyle model of a bundle is specified in Figure 4.4. A bundle can be installed, uninstalled, started, stopped and updated. When a bundle is installed, its code is inserted into the framework and loaded inside a separate class loader.

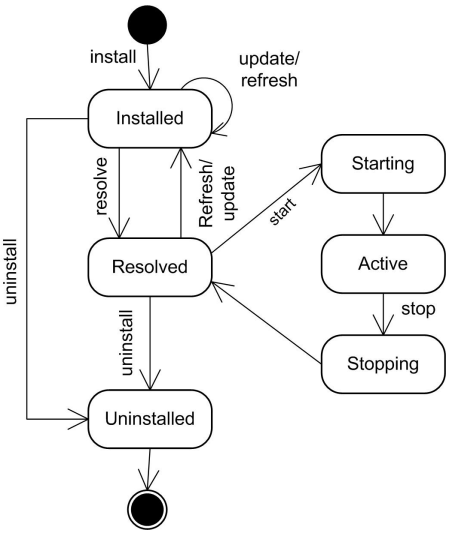


Figure 4.4: The lifecycle model [50]

Before a bundle can be started, it's dependencies have to be met. It can depend on other bundles or services. The framework will check these dependencies and tries to install or start any required bundles. After these resolvements, the bundle can be uninstalled, updated or started. Uninstalling will remove the bundle from the framework and will notify all depending bundles of this fact. Updating will install a new version of the bundle into the framework. The old version of the bundle will be kept active inside the framework if other bundles depend on the specific version of the updated bundle. When a bundle is started, its 'start()'-method is called which will start the bundle. After the bundle has been started, all running bundles will be notified of the start. When a bundle is stopped, the services of a bundle will be eliminated, but the packages that the bundle provides will still be available for other bundles.

[2] A stub is a class file which only contains an interface of the supported services

Service layer The highest layer of the OSGi framework is the service layer. It contains a registry of all running services. Services can publish or unpublish itself in the registry. Each bundle can register zero or more services to the registry. It will tell the service registry that other bundles can use the object that registers the service for service calls. Other bundles can search of a service by looking for the interface name of the desired service. When a service registers itself, extra properties, next to its name, can be specified which enables more advanced search methods using an LDAP[3] based query language. In Figure 4.5 the service layer is visualized.

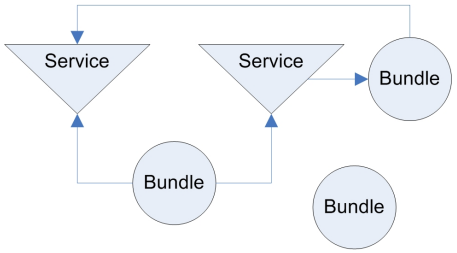


Figure 4.5: The service layer [50]

It is also possible to register a service factory as a service, instead of a service object. A service factory allows the service to be customized for each service user. When a bundle requests a service that is registered as a service factory, it will not receive the object of the service but an instantiation of this object. The object can now be customized for the bundles needs. Another advantage is the notification of stopped bundles, i.e. when a bundle is stopped or it drops a service, the service factory is notified of this event and can act on this.

4.3 OSGi services

The OSGi alliance have specified a number of standardized services. These services support developers in the rapid development of applications. The services are divided into two categories: core services and other services. Both categories are described in the rest of this Section.

4.3.1 Core services

On top of the framework layers a number of core services are specified by the OSGi alliance. Each OSGi certified implementation needs to implement these services. The OSGi alliance only specified the required core services and their interfaces, but does not state how they should be implemented. The available core services are explained in the next paragraphs.

The core services do not have to be enabled in a running framework. For example when a security service is not required in a framework, this service does not have to be installed and started.

[3]Lightweight directory access protocol: `http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol`

Package admin service Bundles can export packages to other bundles. This export creates a dependency between the exported package and the bundle that uses the package. When the state of the exported package changes, a decision has to be taken on what to do with the bundles that are using the exported package [48].

The package admin service provides an administration interface which can be used to modify the state of bundles. The service can influence the internal structure of the framework. When a bundle state is changed (update, refresh, etcetera), an event is generated. A specialized service, the management agent, can listen to these events and decide what to do. The decision can then be enforced through the administration interface. For example, the bundles that use an exported package can be paused or stopped. To enlighten the implementation of the management agent, a policy for managing package sharing can be installed.

Start level service The purpose of the start level service is to control what bundles will be started and stopped and specify when these changes will occur. When an OSGi framework is started, certain standard services will be started. Each of these services have a start level assigned to it. The start level service will ensure that these services are started in the desired sequence. A management agent can be specified which can influence the start level of services. It is also possible to specify a maximum start level. All packages that have a higher start level will not be started at startup.

Conditional permission admin service The conditional permission admin service is used to enable the control of permissions of services by management services. This control will be in effect immediately which makes a restart of the framework unnecessary.

The permission management in OSGi is based on the conditional permission model. This model matches permissions onto bundles using pre-specified or user-defined conditions. A big advantage of this model is the sharing of sets of permissions between bundles and services based on their location and their signature. These conditions can also be used to enable a set of permissions when an external condition is met, for example when a password is entered and validated.

The conditional permission admin service maintains a table of tuples which contains bundles, conditions and permissions. A management agent can manage these conditions and permissions by enumerating, adding or deleting them.

Before this service was required in a framework implementation, the permission admin service was used. Currently both services are supported. The permission admin service is defined as legacy and will be deprecated in a future release of the OSGi framework specification.

Permission admin service The permission admin service defines a default set of permissions for each bundle. Next to this default set, a bundle specific permission set can be specified. The permission admin service enforces the defined permissions. A management agent should provide the bundle specific permissions.

URL handlers service The URL handlers service standardizes the registering of new URL schemes. In Java it is not possible to use a URL scheme in different bundles. This approach is not desirable in OSGi, because a URL scheme should be usable by different bundles. The current Java URL handler implementation is hidden from bundles and a new service, which can handle new distributed URL handler registrations, is being specified. The service will maintain a list of all registered handlers and will enable bundles to revoke, reinstall or use the handlers.

4.3.2 Other services

Next to the core services, the OSGi alliance has specified a list of system and protocol services and their specification. These services are not mandatory for the framework implementation but are considered necessary in almost each system to function properly.

The goal of this thesis is to design a policy control system using OSGi. The design of the system in OSGi will not require the use of other services than the core services. This makes removes the need to have a complete and elaborated list of the available other services. To make the list of available services complete, the names will be mentioned. The website of the OSGi Alliance can be visited when further information on these services is required.

The other services are: Log service, Configuration Admin Service, Device Access Service, User Admin Service, IO Connector Service, Preferences Service, Component Runtime, Deployment Admin, Event Admin, Application Admin, Http Service, UPnP Service, DMT Admin. Their specification will not be elaborated in this thesis. Their specification can be found at the OSGi website[4]

4.4 Framework implementations

Before an implementation of a policy control system can be made, an OSGi framework implementation has to be selected. There are several OSGi framework implementations available, some are closed source and others are open source. It is not possible to

[4]OSGi javadoc including standard services specification: `http://www2.osgi.org/javadoc/r4/`

download closed source implementations, thus only open source implementations are possible candidates.

4.4.1 Available implementations

There are four open source implementations: Apache Felix[5], Eclipse Equinox[6], Knopflerfish[7] and Osxa[8]. The support of the various OSGi framework components is not fully available in each of the implementations. Campanelli [51] made a comparison of these implementations. Table 4.1 shows the result. OK: the implementation has full support for the layer or service; OK-: there is almost full support, a few specific optional things are missing; Partial: Incomplete or untested but usable; -: Missing in the current version.

			Framework specific				Core services				
Implementation	Version	OSGi certified	Security layer	Module layer	Lifecycle layer	Service layer	Package admin	Conditional permission	Permission admin	Start level service	URL Handler
Apache Felix	0.9	No	Partial	Ok-	Ok-	Ok-	Ok-	-	-	Ok	Ok
Knopflerfish	2.0	No	Ok-	Ok	Ok	Ok	Ok	-	Ok-	Ok-	Ok-
Eclipse Equinox	3.2	Yes	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok	Ok
Osxa	-	No	-	Partial	Partial	Partial	-	-	-	-	-

Table 4.1: OSGi open source implementation comparison

From the table, the conclusion that the Eclipse implementation should be used, could be drawn, but possibly not all layers or services are required for the implementation of our policy model. The advantage of using an implementation that does not support all layers or services is the complexity. Less layers or services means less bundles to take into account.

The following paragraphs explain some (background) information of the open source implementations. The website of the Osxa implementation is offline which makes testing this framework impossible. Osxa will therefore not be included in the comparison.

[5] `http://http://cwiki.apache.org/FELIX/`
[6] `http://www.eclipse.org/equinox/`
[7] `http://www.knopflerfish.org/`
[8] `http://www.osxa.org/`

Apache Felix Apache Felix is the successor of the Oscar[9] project. The development of Oscar has stopped. The last version of Oscar was based on the 3rd release of the OSGi framework. Apache Felix is based on the 4th release.

Apache Felix is currently in the incubator status of the Apache Software Foundation (ASF). This means that is does not have full support from the ASF, but it is trying to receive that. It also means that its implementation and support of the community is not yet stable and sufficient. A disadvantage is the possibility of long delays of new versions of the framework when bugs arise or new OSGi framework specifications are released. Another disadvantage is the poor manual on the website of Felix. An advantage is the small size of the package. The only interface available for developers is text-based. No GUI is available.

An advantage of developing bundles with Apache Felix is the use of the Maven plugin. This plugin simplifies the creation of bundle files and generates the required activator code files. A disadvantage of the Apache Felix framework is that it looks like the development of this framework has slowed down to almost a halt and the availability of very brief documentation.

The Felix framework version 0.8.0 has a size of 305kb and includes the earlier mentioned supported core services and layers. The basic framework does not include any optional services. It is not clear which Java Runtime Environment (JRE) is required for the execution of the basic framework.

Knopflerfish Knopflerfish is an OSGi framework implementation maintained by Gatespace Telematics[10]. In 2003 was decided that the Knopflerfish implementation should be moved from closed source to open source. Gatespace Telematics is currently the primary maintainer and sponsor of the open source project and contributing developers to support the project.

Knopflerfish contains a console and a GUI implementation, which makes it easy to 'see what you do'. Good documentation is online which helps the user to create a new bundle with ease. A plugin is available which makes it easy to program a new bundle in the Eclipse Software Development Kit (SDK). Templates are available which enables rapid bundle development.

The size of the knopflerfish framework version 2.0.0 is 287kb and includes the supported layers and core services. The non-core services that are implemented are available through an online bundle repository or can be included into the local bundle repository. The framework needs at least JRE version 1.2.2 as execution environment.

[9] `http://oscar.objectweb.org/`
[10] `http://www.gatespacetelematics.com/`

Eclipse Equinox Eclipse Equinox is the only open source implementation that has received an OSGi certificate for release 4.0. The Equinox implementation is also used in the Eclipse SDK as a run-time plugin loader. The Eclipse SDK has a special package which makes the programming and testing of a bundle very easy. All standard files and classes are generated and a bundle can be tested inside Eclipse without installing Equinox.

No GUI implementation of Equinox is available but this is not necessary, because every bundle can be tested inside the GUI of the Eclipse SDK. Very much documentation and reference bundles are available to help the programmer with setting up good bundles. A disadvantage of using the Eclipse SDK is the burden it brings on the workstation of the developer. A relatively new computer is needed in order to make normal use of the SDK.

The size of the Equinox framework version 3.2.2, including all required layers and core services is 853kb. This also includes some of the optional services like logging. Equinox can use a wide variety of JRE's from 1.1 until the current version 1.6. The version that will be used, depends on the requirements of the bundles.

4.4.2 Selected implementation for the prototype

The requirements of the OSGi framework should be clear before any design on an implementation is done. This section describes these requirements. It will describe which basic layers and services are required for the control system to function properly.

As shown in table 4.1, not every layer or service is fully implemented in the available implementations. The design of the system using OSGi will require the implementation of some of the available layers and core services. These requirements are based on the current MBU application and on the analysis of the mobihealth system.

There are four available layers: module, lifecycle, service and security. Each of these layers needs to be available, because each of them is needed in the system that has to be developed. The module, lifecycle and service layer are the basic layers that implement the core functionality of OSGi and without them the framework does not function properly. The security layer is not required for the framework to function, but it is required when some kind of security needs to be implemented. The security of the framework and bundles is outside the scope of this thesis and will therefore not be discussed.

There are five core services. The package admin, start level and URL handler are basic services that need to be implemented, else the framework will not function properly. The other two services, permission admin and conditional permission admin control the security of the framework. The permission admin states the permissions per bundle and the conditional permission admin lets bundles change these permissions. The policy system that has to be

designed has to be protected from unauthorized changes, but the system itself does not have to change these permissions at runtime. Thus the permission admin is required and the conditional permission admin is not.

If above framework requirements are matched against Table 4.1, we see that the Apache Felix framework does not implement the required layers and services and thus cannot be used. The Knopflerfish framework has almost full support for the required layers and services and the Eclipse Equinox framework has full support.

Testing of each of the framework implementations on the current hardware and software of the MBU is a good method to clarify the requirements of each of the implementations but these test falls outside the scope of this thesis and are therefore not conducted.

The exchangeability of bundles between framework implementations offers developers to use different frameworks for development and execution environments. It is possible to develop bundles with the Eclipse Equinox framework and run them on the Knopflerfish framework. The choice which framework is used for development depends on the preferences of the developer.

Eclipse enables developers to develop OSGi bundles for the Equinox framework with great ease. A lot of the generation of bundle specific code is done by Eclipse and compiling and running of bundles inside the framework is made very easy‘. These advantages over Knopflerfish make that we will use the Eclipse environment for bundle development.

4.5 OSGi and policies

The dynamic part of the policy model are the different policies. These policies have to be loaded and interpreted at run-time. The obvious method to do this in OSGi would be to implement each policy as a bundle. Each of these policy bundles will have to have a standard interface to make standard communication with all the policies possible. The advantage of making each policy a bundle is the ease in which policies can be loaded into the PDP.

Another advantage of using the OSGi framework for policies is the ease on which policies can be installed or activated. If a policy is deactivated or removed, the policy decision point can be informed of this fact and act on this. It does not have to check this on its own every now and then.

The communication between the different entities in a policy model are also simplified with the OSGi framework, because a registry is available where services, and thus the available policies, can be searched. Entities can also register to an event registry. When an entity generates an event, other entities are automatically informed of this event. For example:

when a new connection type comes available, its policy information point will send out an event. The policy decision point will have registered to the event of the environment monitor and will be automatically informed of the new situation.

4.6 Summary

This chapter has showed us an overview and description of the OSGi framework and its layers and services. We also discussed the different implementations of the framework and their advantages and disadvantages. A discussion what the requirements of the control system are, is given including a choice which implementation will be used (Eclipse Equinox). A short description of how policies can use the framework to their advantage is discussed.

Chapter 5

Policy specification for an MBU control system

The policy-based control system that will be designed in this thesis will control the MBU, as described in the requirements (Section 2.5). As explained in Chapter 3, the policy-based control mechanism consists of two parts: the policies and an architecture which facilitates the execution of the policies. This chapter will describe and discuss the specification of the policies. The following chapter describes and explains the architecture.

This chapter is divided into two parts. Section 5.1 shows and discusses the policy structure. Section 5.2 describes the policy specification and some examples of policies.

5.1 Policy structure

The policy structure that is used as a solution to the problem description of Section 2.6 is shown in Figure 5.1. The structure is expressed in a pseudo UML class diagram. The term 'pseudo' is used because a class (square box in Figure 5.1) represents a policy or policy set component, which are not necessarily an object class which contains methods. We borrow the association constructs of the UML class diagram.

The policy structures that are discussed in the analysis of policy (Section 3.5) are used as a base for this policy structure. The main components of the IETF Policy Core Information Model (PCIM), Section 3.5, are being reused in our policy structure. Also the idea behind the goal component is being used from the extended policy structure (3.5.1).

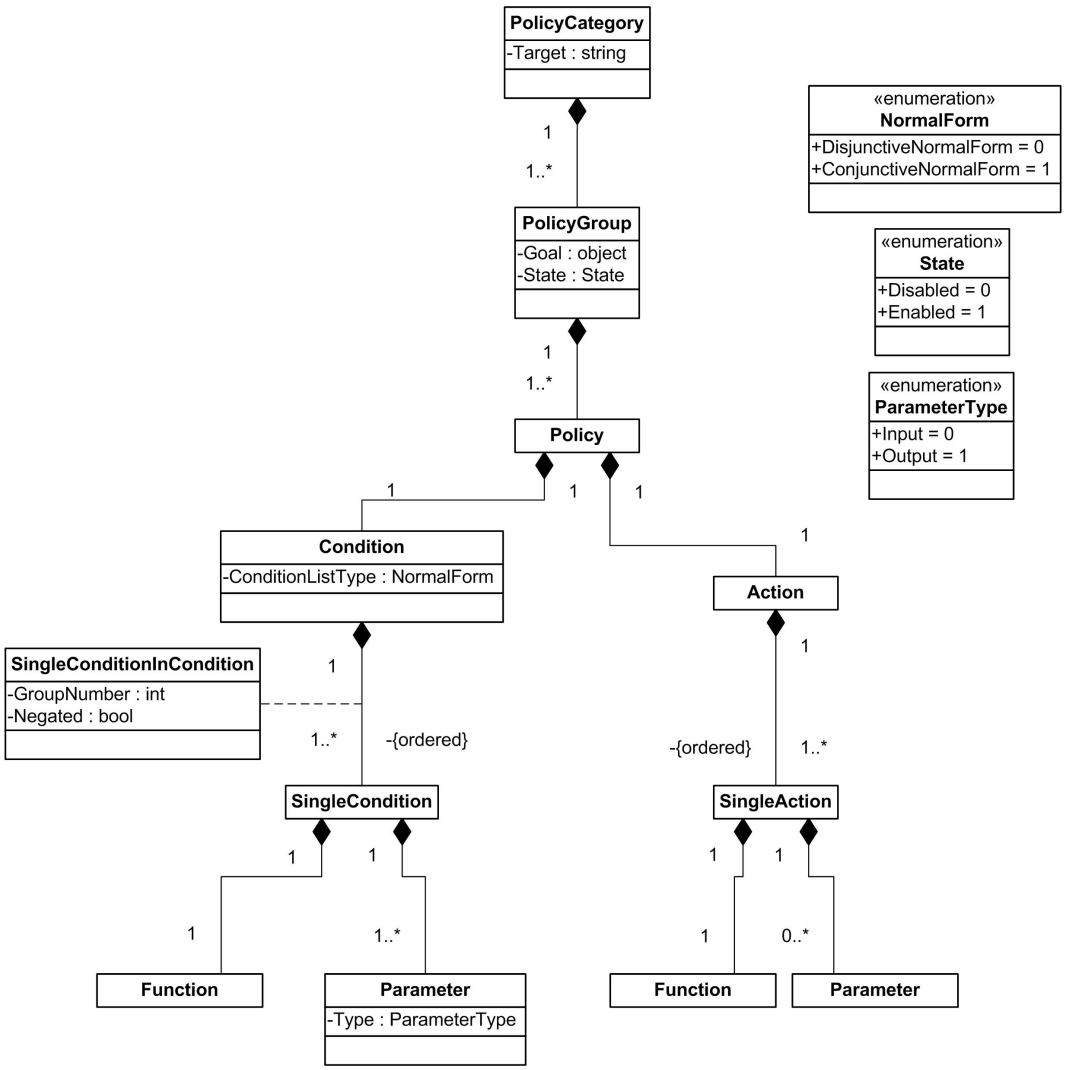


Figure 5.1: Pseudo UML class diagram of the policy (set) structure

The described components have different orientation towards the system. The components PolicyCategory and PolicyGroup are end user oriented; they are used in the conversion from user preferences towards policy structure components. The other components are system oriented; they are used in the policy-based system to make decisions.

The following paragraphs will explain the components of the policy structure using a bottom-up approach. If a component has a corresponding component in the IETF PCIM, a comparison between these two will be described. The last two paragraphs will explain how the two representation levels are integrated into the policy structure and how user preferences influence the policy.

Function The component Function represents a boolean function. A function is related to one SingleCondition or one SingleAction. This component is not present in the PCIM, because the IETF abstracts from the elements that are present inside a single condition.

Parameter A Parameter represents a variable or a constant that can be used as a parameter for a function. There are two parameter components present in our policy structure. The Parameter component below the component SingleCondition has one attribute, Type. This attribute states if the parameter is an input or an output parameter. When the Type of a Parameter is input, the Parameter is used as input for the Function component. When the Type of a Parameter is output, the Parameter is used as output for the Function component. The Parameter component below the component SingleAction has zero attributes. A Parameter is related to one SingleCondition or one SingleAction.

SingleCondition A SingleCondition represents a combination of one Function and one or more Parameters. One or more SingleConditions are related to one Condition. The relation between a SingleCondition and Condition has the constraint 'ordered'. This order number is required to enforce the execution of conditions in a certain order. A SingleCondition with orderNumber 1 is executed before a SingleCondition with orderNumber 2. Some SingleConditions have to be executed before others, because SingleConditions are allowed to re-use parameters. For example, a SingleCondition that is executed first, has a parameter 'A' which has type output. A SingleCondition that is executed later, can use the parameter 'A' as input parameter. This enables SingleCondition's to pass on information to others. In the PCIM a SingleCondition is called PolicyCondition and also has zero attributes.

SingleConditionInCondition A SingleConditionInCondition specifies the relation between a SingleCondition and a Condition. Every SingleConditionInCondition contains two attributes. The first attribute, GroupNumber, states the number of the group which the SingleCondition belongs to. A group is used by the component Policy to distinct different condition groups. The functioning of the groups and their interactions, is explained at the description of the Condition component. The second attribute, ConditionNegated, states if the result of the SingleCondition should be negated. In the PCIM a SingleConditionInCondition is called PolicyConditionInPolicyRule and contains the attributes GroupNumber and ConditionNegated. The ordering, which is present in our structure, is not specified in the PCIM. We have chosen to add this attribute to enable more complex Conditions.

Condition A Condition is the aggregation of one or more SingleConditions. A Condition contains one attribute: ConditionListType. This attribute states the normal form of the condition list and is of the type 'NormalForm'. The enumeration 'NormalForm' can have the values DisjunctiveNormalForm (0) and ConjunctiveNormalForm (1), as described in Section 3.5. It uses the attributes of the related SingleConditionInCondition components to form a condition. A description of how this component relates to the PCIM, is presented at the description of the Policy component.

The GroupNumber of the component SingleConditionInCondition is used to specify which SingleConditions are in which group. This grouping is required to use the conjunctive and disjunctive normal form as described in Section 3.5. The functioning of the groups is exactly the same as described in the referred Section.

SingleAction An Action represents an action that will be enforced. This action consists of one Function and zero or more Parameters. One or more SingleActions are related to one Action. The relation between a SingleActions and Action has the constraint 'ordered'. This order number is required to enforce the execution of actions in a certain order. In the PCIM a SingleAction is called PolicyAction and has zero attributes. The ordering of SingleActions is not present in the PCIM and cannot be specified. For us it is necessary to have this ordering, to make sure certain SingleActions are enforced before others.

Action An Action represents one or more SingleActions. It uses the attribute of the component SingleActionInAction to determine in which order the SingleActions should be enforced.

Policy The policy component represents a policy as described in Section 3.2 of the policy analysis. In the PCIM the components Policy, Condition and Action are integrated into one component. We have chosen to model three separate components instead of one, to make the relation between them more clear. Our model explicitly links one Condition to one Policy and one Action to one Policy.

PolicyGroup A PolicyGroup is a set of one or more Policy components. Policies that are combined into one PolicyGroup share the same goal. This goal is an attribute of a policy group. It represents the goal that the policy group wants to strive after. The goal is represented by an object. Such an object can contain multiple subgoals, e.g. 'delay=low' and 'accuracy_ECG = high'. The second attribute represents the state of a group. A state is an enumeration which has the values Enabled (1) or Disabled (0). The purpose of this attribute will be explained at the description of the component PolicyCategory. The PCIM

also contains a PolicyGroup component. It has the same intention as our PolicyGroup, but it does not contain the attributes Goal and State.

PolicyCategory The PolicyCategory component bundles one or more PolicyGroups together into one category. PolicyGroups that influence the same controlled target (part of the controlled system) are combined in the same category. For example, PolicyGroups that influence the selection of a communication channel, are grouped together into one PolicyCategory. This bundling of groups is done to have a clear view which PolicyGroups influence which target. It is important for a target to be independent from other targets. If it is dependent to another target, conflicts can occur due to different actions that will be enforced at the target. A PolicyGroup has one attribute, Target, which represents the name of the target that the group influences.

This component is represented differently in the PCIM. To combine multiple PolicyGroups together in the PCIM, a PolicyGroup is related to zero or more PolicyGroups (Section 3.5). We choose not to use this method, because a target cannot be specified in a PolicyGroup in the PCIM.

Figure 5.2 shows two instances of PolicyCategory and five instances of PolicyGroup. A interconnection between a PolicyCategory and a PolicyGroup means that the groups share the same category. In other words, groups that share the same category control the same part of the controlled system. The figure also shows the use of the attribute State in the component PolicyGroup. To make sure only one goal is strived after at a certain moment in time, only one group may have the State 'enabled'. The rest of the groups is required to have the State 'disabled'.

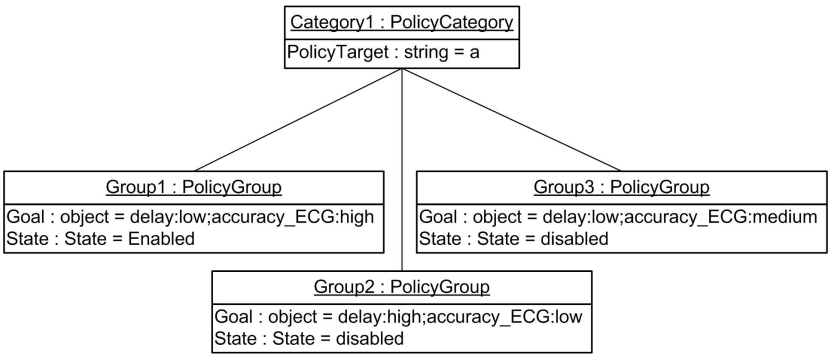


Figure 5.2: Instances of groups and categories

The attribute 'state' is not required when only PolicyGroups which are 'enabled', are loaded inside the PDP. If this is the case, all policies with the state 'disabled' are not present in the PDP and thus will not be triggered.

User preferences As described in Section 3.6.2, users need to be able to state their preferences. These preferences have to be linked to the actual policies. Preferences are stated by selecting a goal for each of the available PolicyCategories. In other words: every PolicyCategory will have one PolicyGroup that is enabled. If a user states another preference, the state of the current enabled PolicyGroup has to be disabled and the state of the new PolicyGroup, that belongs to the goal stated in the preference, has to be enabled.

It is also possible that more than one user specifies a preference for a single PolicyCategory. If this is the case it is possible that the preferences of the users interfere with eachother. It is therefore important that priorities are assigned to each of the preferences if more than one is specified. This priority helps the control system to select the correct goal. When for example there are two users, A and B, and there is one PolicyCategory, which influences a target. Both users influence this PolicyCategory. If user A states a preference 'Low', and user B states a preference 'High', the control system does not know which goal to select, because the preferences conflict with eachother. If the preferences have priorities, for example user A has priority over user B, the control system does know what to do with the preferences and which goal has to be selected for the PolicyCategory.

5.2 Policy specification

This section will describe the policy specification, using the policy structure of the previous section, for the problem described in section 1.1. The specification will be described using a top-down approach. Attributes that are described in the policy structure that do not yet have known values, will be described first. Next the generic policies for the possible targets are described. Section 5.2.5 gives examples of the policy specification

5.2.1 Target

As described in the previous section, it is important to select a proper part of the Mobihealth system as a target. Such a target should be an independent part of the system, else multiple targets interfere with eachother. The Mobihealth system has two independent targets: sensor and connection. At the target 'sensor' the sensor data is mangled and at the target 'connection' the connection that is used to transmit sensor data is selected.

5.2.2 Goal

The controls of the two described targets will each have a number of goals. The number of goals that need to be specified, depend on the number of variables and their possible values

that can be specified in a user preference. For example, if there is one variable 'ECG sample rate' and the possible values for this variable are '16' and '32', the number of goals will be $(number of variables) * (number of values for each variable) = (1) * (2) = 2$.

As described in Section 2.5, users will specify preferences. These user preferences are used to select a goal for each target. The requirements of the system state the need for two users to specify their preferences: the doctor and the patient. There are also two targets on which these users may have influence. The influence of each user on a target is visualized in Figure 5.3. The doctor states preferences on the sensors. These preferences are needed to select a connection and to select the configuration of sensor data. The doctor will thus influence both targets. The patient has influence on the connection, because he only influences the connection that will be chosen. Both users have influence on the connection category so this category needs to be prioritized.

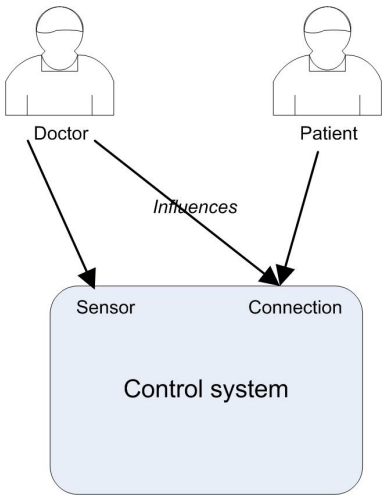


Figure 5.3: User influences on the targets

As described in Section 5.1, a target that is influenced by more than one user needs to have prioritized preferences. There are two possible options: the preferences of the patient are more important than the preferences of the doctor and vice versa.

If the preferences of the patient are more important than those of the doctor, it is possible for the patient to specify such preferences that a connection will be used that does not satisfy the minimum sensor sample rates, defined by the doctor. The sensor data that is received by the doctor is then below the defined minimum and has no use for the doctor.

When the preferences of the doctor are more important than those of the patient, the transmitting of (at least) the minimum sensor sampling rate is more important than the patients preferences. This makes sure the doctor always receives valid sensor data. It is thus clear that the preferences of the doctor have priority over the preferences of the patient.

The following two paragraphs will discuss and define the possible goals for each of the targets.

Sensor Before the possible goals for the target 'sensor' can be described, a number of sensor properties should be known. These will be discussed in the following list.

- *Supported sensors* There is a variety of sensors available in the mobihealth system as described in Section 2.2.2. Each of the sensors that will be supported by the policy specification is required to have a preference stated by the doctor, else it isn't possible for the control system to make a decision for each specific sensor.
- *Groups* As described in the previous section, certain sensors belong to eachother. If the data of one sensor cannot be transmitted, the data of the sensors that belong to the same group is not required anymore and do not have to be transmitted. This property is specified by assigning a group number to each sensor. If group numbers of sensors are the same, they belong to eachother.
- *Resolution* The possible goals of the sensor category are based on the number of supported sensors and the resolution options of these sensors. We choose to have three resolutions for each sensor: minimum, preferred minimum, preferred average. These resolutions are chosen based on the requirements (Section 2.5). Minimum states the absolute minimum resolution of the sensor data. Below this level the sensor data is so innacurate that it has no value for a doctor. The preferred minimum represents the minimum resolution that is acceptable for the doctor. The preferred average is the resolution the doctor would like to see as the average level.
  As described in Section 2.4.2, the resolution of a sensor depends on the sample rate and sample size of a sensor. The sample size is a fixed number, but the sample rate can be varied. Each sensor has a different set of sample rates that are available for use.
- *Priority* If more than one sensor is specified in a preference, it is required to prioritize them. This prioritizing is required to have clear which sensor is more important than another sensor. A sensor with a higher priority than another sensor will have its goal strived after before the other sensor.

The described properties result in the following variables that have to be defined for the goal of each sensor:

(Minimum sample rate, Preferred minimum sample rate,Preferred average sample rate, Priority, Group number)

The possible values for each of the variables depend on the type of sensor and the amount of sensors that are defined in a goal. It is required that the minimum sample rate is smaller of equal than the preferred minimum sample rate and that the preferred minimum sample rate is smaller or equal than the preferred average sample rate.

The selection of which sample rates are used in a goal depend on the preferences of a doctor. To keep the preference specification as simple as possible for a doctor, labels will be used instead of the sample rates. Each sample rate will have an attached label which corresponds to its value. There is for example a sensor with five possible sample rates: 4, 8, 16, 32 and 64. This results in five different labels for these sample rates: very low, low, medium, high, very high. These labels correspond with the actual sample rate values, so low corresponds to 4, etcetera. A doctor will now fill in a goal for this sensor by specifying the three possible sample rate values using these labels, e.g. a doctor specifies (very low, low, medium, 1, 1) as preference for this sensor if he requires a low resolution. This preference specifies that the following goal should be used: (4, 8, 16, 1, 1). If he always want to have the highest resolution of a sensor at any moment, he will specify the following goal as preference: (very high, very high, very high, 1, 1), which corresponds to the goal: (64, 64, 64, 1, 1).

Connection The possible goals for the target 'connection' are based on two user preferences. The preferences of the doctor, which state information of the amount of required bandwidth, and the preferences of the patient, which state certain demands on the used connection. The preferences a doctor has to supply are defined in the previous paragraph. The goal a patient has to supply will state which connections are allowed to be used.

The settings that are available for each connection define the possible variables that are used in a goal. These variables can be used to specify a certain preference. For example: when the cost per byte for each connection is known, the patient can state a preference which contains a restriction on the cost per byte.

There are two possible options for describing the values of a variable. The first option states the exact value. For example, the patient specifies the maximum cost per byte at 10 cents. An advantage of this option is the exact known value that is allowed for each variable. A disadvantage is the flexibility of this method. When, for example, the doctor wants to have a higher resolution, more money has to be spend on a connection, while this is not possible if the policy system wants to conform to the specified goal.

The second option is the specification of relative values of the variables. This option will not state the exact value of a variable, but a goal that has to be strived after. When, for example, 'cheap' is specified as a goal for the cost of the connection, the policy tries to select the cheapest connection. When a doctor wants a higher resolution for a sensor, it is now possible to adapt the external connection to this setting, because the 'cheapest' connection has to be be selected.

The variables that will be supported by the policy depend on the requirements of a patient. If, for example, the patient wants to specify exactly which connection may be used and which

one may not be, it is required to have a list of allowed connections in the preference. In the following part of this thesis we will use 'relative cost' as an example for the patient preference. The possible values of this variable are defined as 'Cheap', 'Moderate' and 'Expensive'. These values are chosen as an example and more can be added if this is required.

5.2.3 Generic policy

The next step in the implementation of the policy specification is the defining of required policies, based on the possible goals.

As described in the previous paragraphs, the difference between certain goals is defined by values of the variables. The variables are the same for each goal and thus the structure of each of the goals, that belong to the same target, is the same. Therefore a generic policy will be defined that will be the same for all the goals that belong to the same target. The following paragraphs will define generic policies for each of the targets.

Sensor The Policy of the 'sensor' target determines how the available bandwidth is distributed among the available sensors. Before the generic policy is presented, the method that is used to allocate the available bandwidth to sensors should be known.

There are different algorithms on how to allocate available bandwidth to the sensors. Some examples are:

- Allocate the amount of available bandwidth that corresponds with the variable 'preferred average sample rate' to the sensor with the highest priority. Repeat the allocation for the sensor with the next highest priority. Continue until all bandwidth is allocated;
- Distribute the available bandwidth completely balanced over all sensors;
- Distribute the available bandwidth using such a method that all sensors have some bandwidth allocated, but make some differentiation in the allocations based on the priorities of the sensors.

We choose to use the last algorithm based on the requirements (Section 2.5). The requirements state that the doctor wants it's preferences strived after. Therefore the doctor would like to have the best possible solution which conforms the best to his preferences. The algorithm tries to distribute the available bandwidth over the sensors, but leaves some room for the priorities of the sensors. Thus the algorithm tries to conform to the preferences. The following paragraphs will explain the used algorithm in more detail.

Figure 5.4 shows a bar which represents the total amount of bandwidth that is required by all sensors for each of the possible resolutions. These resolutions are defined in the preferences

of the doctor. The total amount of required bandwidth for each resolution is calculated by multiplying the sample size and the sample rate of each sensor and sum up all these values. This results in a bandwidth requirement for each of the sample rate categories. When the goal of a sensor is updated, these values have to be recalculated, because the amount of required bandwidth for each category is changed.

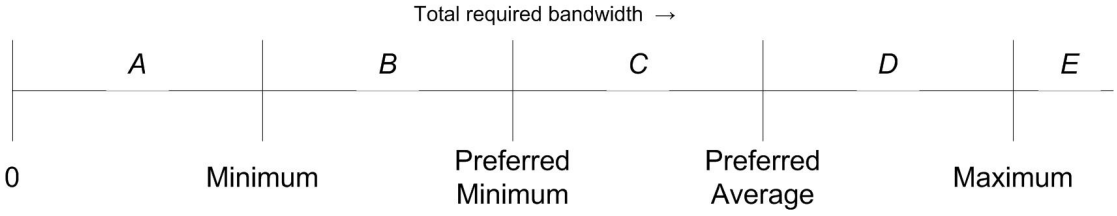


Figure 5.4: Total required bandwidth categories

The total amount of required bandwidth is variable, because it depends on the preferences of the doctor. Therefore only the names of the categories are shown in the figure. An extra sample rate, maximum, is added to the bar. This sample rate is the rate which is used by the sensor to transmit data to the MBU. When there is enough bandwidth available, policies will use this value to allocate the maximum amount of bandwidth to a sensor. An example of Figure 5.4 is described in the following paragraph.

Example As an example, preferences of a doctor are shown. These preferences contain two sensors and are shown in Table 5.1:

Sensor	**Minimum**	**Preferred minimum**	**Preferred average**	**Priority**	**Group**
ECG	low	medium	medium	1	1
SPO2	medium	high	very high	2	2

Table 5.1: Example of doctor preferences

These preferences can be translated to the goals, as shown in Table 5.2

Sensor	**Minimum**	**Preferred minimum**	**Preferred average**	**Priority**	**Group**
ECG	128	256	256	1	1
SPO2	32	64	128	2	2

Table 5.2: Example of doctor preferences translated to goals

The sample size of these sensors, described in Section 2.4.2, and the sample rates for each category will result in the total bandwidth requirements visualized in Figure 5.5. When, for example, the available bandwidth is 10 kbps, and the bandwidth requirements are as in the

figure, the generic policy of category D will be used. If the doctor changes his preferences, the total bandwidth requirements changes. Then new bandwidth numbers have to be calculated for each category.

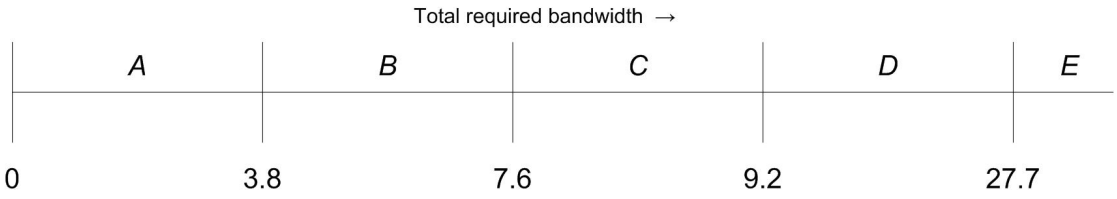


Figure 5.5: Example of total required bandwidth in kbps

Generic policies Each of the categories of Figure 5.4 will require a different generic policy, because they require a different allocation of bandwidth.

Table 5.3 shows what the generic policy should do when the available bandwidth is in a certain category. This generic policy is based on the earlier described algorithm.

Category	**Policy**
E	Transmit all sensor data.
B-D	Allocate the left value of the category to all sensors and allocate the extra available bandwidth to sensors based on their group and priorities but do not allocate more to a sensor than the right value of the category.
A	Allocate bandwidth to sensors based on the sensor groups and priorities. If a sensor in a group has a sample rate below the minimum resolution, allocate a sample rate of 0 to the whole group.

Table 5.3: Generic sensor policies

The triggering of the above policies should take place when: the doctor states updated preferences for the sensor category or the amount of available bandwidth changes.

Connection The policy with target 'connection' selects the connection that will be used to transmit the sensor data. As stated earlier in this section, the preferences of the doctor have priority over the preferences of the patient. To combine both preferences into one goal, the preference of the patient has to be matched onto a part of the preferences of the doctor. As an example to explain the matching of the patients preference on the doctors preference, Table 5.4 is presented. As shown in this table, the patients preference determines which of the available sample rates will be used. The selected sample rate is used to determine how much bandwidth is required for each sensor.

Patient preference	Sensor sample rate
Cheap	Preferred minimum
Moderate	Preferred average
Expensive	Maximum

Table 5.4: Preferences matching

When for example the patient selects 'cheap' as its preference, the amount of required bandwidth of the sensor preferences category 'preferred minimum' will be used as a minimum amount of bandwidth that needs to be available. When the example of Figure 5.5 is used, the amount of available bandwidth for a selected connection should then be at least 7.6 kbps.

The generic policy for the connection category will follow the next pattern.

- Calculate the minimum required bandwidth based on the preferences of the doctor and the patient;
- Select the connections that conform to the required bandwidth;
- Select the cheapest connection from the previous list;
- If the connection is not already in use, switch to the selected connection.

The triggering of this generic policy should take place when: the doctor states updated preferences for the connection category; the patient states updated preferences for the connection category or the connection settings change

5.2.4 Condition and Action

The next step is to define conditions and actions that form the generic policy defined in the previous paragraph. The conditions and actions consist of functions and parameters. The following two paragraphs will decompose the generic policies into functions and show the use of the functions in the generic policies for each of the PolicyCategories. The last paragraph will describe the parameters. The environment variables will be available to each function and will therefore not be passed on as a function parameter.

Connection The generic policy with target 'connection', described in the previous paragraph, can be rewritten to a policy expression containing functions and parameters. The result is shown in table 5.5. It matches every step of the generic policy onto a function. The pseudo code of these functions can be found in Appendix A.

Task	Function
Calculate the minimum required bandwidth	CalculateRequiredBW(DoctorPreferences)
Select the available connections	AvailableConnections()
Select connections that conform to the required bandwidth	SelectPossibleConnections(requiredBW, availableConnections,result)
Select the cheapest connection	SelectCheapestConnection(possibleConnections, result)
Switch to a connection	SwitchToConnection(connection)

Table 5.5: Connection policy functions

Sensor The generic policy with 'target' can also be transformed into pseudo code, however the five different bandwidth categories each require a different policy. Therefore there will be five policies that together represent one goal. Table 5.6 shows the required functions. Each task that is defined in the generic policy is translated to a function name which will be used in the actual pseudo code. Pseudo code of each of these functions can be found in Appendix A.

Task	Function
Compare bandwidths	IsFirstSmaller(bw1,bw2)
Select optimal configuration for category 'X'	SelectOptimalSensorDataConfigurationX(DoctorPreferences, AvailableBandwidth,result)
Enforce a sensor data configuration	EnforceConfigurationOnSensorData(result)

Table 5.6: Sensor policy functions

Parameters The functions that are described in the previous two paragraphs need to know information from the environment of the system. This information is used to make a decision. Figure 5.6 shows a graph that contains the minimum variables that should be known to the policies. These minimum variables are derived from the goals of the generic policies. The PolicyCategory sensor controls the amount of bandwidth is allowed to be transmitted for each sensor. It is thus required to know the bandwidth of each sensor, and especially the sample size and the sample rate. The PolicyCategory connection control the selection of the connection and thus should know the amount of bandwidth that is supplied by each connection. In the policy implementation the goal of the patient also states information on the cost of a connection. Therefore also the cost per byte should be known to the system.

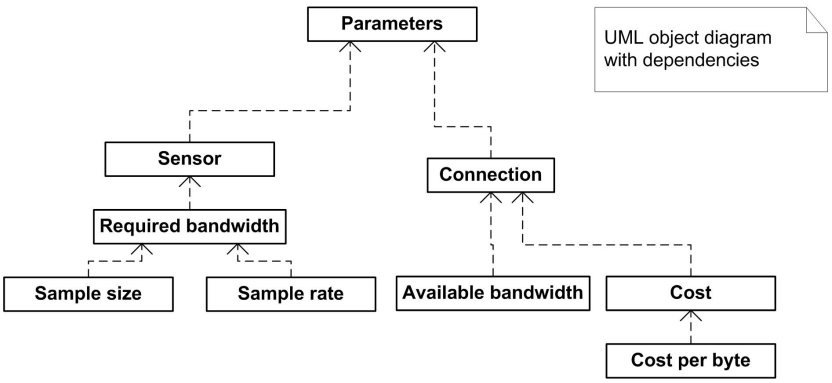


Figure 5.6: Parameter dependency graph

If the preferences of the patient will contain more restrictions on variables of a connection, it is required to extend the graph. When for example, the patient wants to specify the 'Time To Live' of a network packet, a variable 'TTL should be added to the goal specification and to the dependency graph.

5.2.5 Examples

This Section will give the pseudo code of two policy examples. One for each PolicyCategory. All pseudo code of the specified policies can be found in Appendix A.

Connection policy

The following pseudo code will describe the combining of the functions into a policy. The pseudo code represents the generic connection policy when the goal of the patient is 'cheap' and is the same for all connection policies that strive after this goal. The variable 'doctorPreference' contains the preferences of the doctor and makes the generic connection policy variable. When different preferences are specified by the doctor, the policy will result in a different action that should be enforced. The pseudo code for the other goals of the patient can be found in Appendix A.

```
if (  (event == "ConnectionVariableChanged" ||
       event == "UserPreferenceChanged") &&
       SelectPossibleConnections(CalculatePreferredMinimumRequiredBW(doctorPreference),
                                 AvailableConnections(),
                                 result) &&
       currentConnection != selectCheapestConnection(result,result2))
then
{
    SwitchToConnection(result2);
}
```

The pseudo code contains some extra variables and conditions. The 'event' variable contains the event that generated the triggering of the policy. If a connection variable or a user preference has changed, the policy should be executed. The variable 'currentConnection' contains the name of the connection that is currently in use. The variable 'result' and 'result2' are used to pass information from one SingleCondition to another SingleCondition or SingleAction.

Sensor policy

The following pseudo code is an example of how a generic sensor policy looks like. The example consists of a policy which belongs to bandwidth category D. This policy checks during its execution if the available bandwidth is between the total preferred average sensor bandwidth and the maximum sensor bandwidth. If this is the case, an optimal sensordata configuration is chosen and enforced. The following abbreviations are used in the pseudocode: bw = bandwidth; sbw = sample bandwidth; sample bandwidth = sample rate * sample size. Pseudo code of the other four bandwidth categories can be found in Appendix A.

```
if (  (event == "ConnectionVariableChanged") ||
       event== "SensorVariableChanged") &&
       IsFirstSmaller(CalculatePreferredAverageBW(doctorPreference),
                      BandwidthAvailable()) &&
       IsFirstSmaller(BandwidthAvailable(),
                      CalculateMaximumBW(doctorPreference)) &&
       SelectOptimalSensorDataConfigurationD(doctorPreference,
                                             BandwidthAvailable(),result) )
then
{
    EnforceConfigurationOnSensorData(result);
}
```

The pseudo code contains some extra variables and conditions. The 'event' variable contains the event that generated the triggering of the policy. If a connection variable or a user preference has changed, the policy should be executed. The variable doctorPreference contains the preferences stated by the doctor. The variable 'result' is used to pass information from a SingleCondition to a SingleAction.

Chapter 6

An architecture for a policy-based MBU control system

The previous chapter described the specification and implementation of policies. This chapter will describe the policy-based control system architecture which uses the implemented policies. The design of this architecture is based on the in Section 3.6.1 discussed IETF framework.

The design of the architecture will follow a top-down, or step-wise refinement approach. First a high level overview, or external system perspective, is given including the interactions between the system and the environment. In this perspective the system is looked upon as a black box. The next step is the design of the internal system perspective. In this perspective a design of the system is described, consisting of a graphical representation of the architecture, an explanation of the components and their interactions. The third section describes the inner functioning of internal system perspective components by refining them once more. The last section will give the conclusion of this chapter.

6.1 External system perspective

The external system perspective of the architecture gives an overview of the components that will interact with the system. The following paragraphs will describe these components and the interactions they have with the system.

The generic policy model, described in Chapter 3, states two different input components which will feed information to the system: users and environment information. If the requirements (Section 2.5) are matched against onto these two input types, two user input components and one environment information component are required. The two input components the doctor

and the patient and the environment information component is Mobihealth. The components and their interactions are visualized in Figure 6.1.

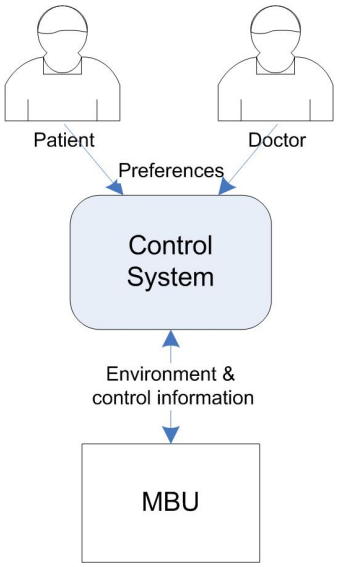


Figure 6.1: External system perspective

The tasks of each of the four components, based on the requirements (Section 2.5) and analysis (Chapter 3) is:

- **Patient** - States its preferences towards the functioning of the system;
- **Doctor** - States its preferences towards the functioning of the system;
- **MBU** - Enforces decisions and transmits information regarding its environment;
- **Control system** - Makes decisions based on the preferences of the patient and the doctor and the environment information from Mobihealth.

To explain the external behaviour of the system, three UML sequence diagrams (Figure 6.2, 6.3 and 6.4) are modelled. These diagrams are based on the possible input of the system and will state the possible output. As shown in the external system perspective, there are three inputs which can result in the output of control information. Figure 6.2 shows the sequence diagram when the patient specifies preferences, Figure 6.3 shows the sequence diagram when the doctor specifies preferences and Figure 6.4 shows what happens when the environment of the MBU chanhes. As shown, each of the inputs can result in a decision, but this decision is not mandatory. The communication between the patient, doctor, MBU and the control system is based on event-based messaging.

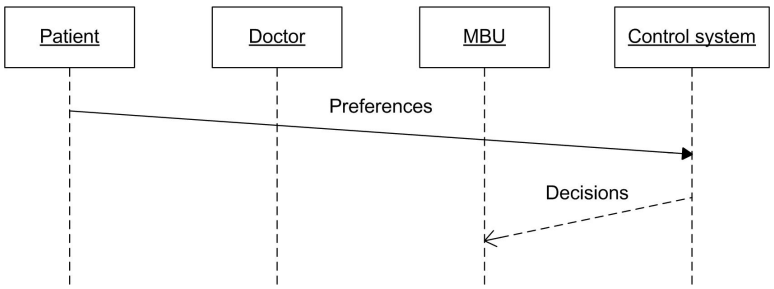


Figure 6.2: Sequence diagram based on patient input

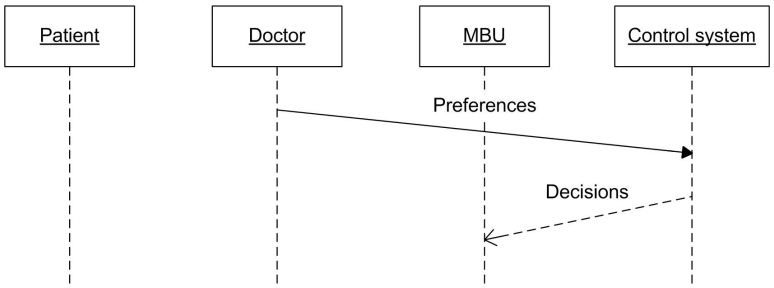


Figure 6.3: Sequence diagram based on doctor input

6.2 Internal system perspective

The internal system perspective refines the system component of the previous paragraph into several components. This perspective is based on the policy model that is described in section 3.6.2 and is visualized in Figure 6.5.

The overview consists of six components. Five of these components are based on the generic components of the enhanced IETF model. Their functioning is discussed in section 3.6.2. The behaviour and interactions of the six components is modelled as two sequence diagrams, based on the possible input of the system. These inputs are user preferences and environment information. The preferences of the doctor and the patient are grouped together, because their input triggers the same sequence through the system.

Figure 6.6 describes the behaviour of the system when a user preference is send to the system. The user preference is received by the UPM. The UPM proccesses the preference and sends the result to the PDP. The PDP then requests the PolicyGroup from the PR that belongs to the preference. The PDP then sends one or more decisions to the PDPP which translates the decisions to machine language expressions that can be executed by the PEP and forwards these.

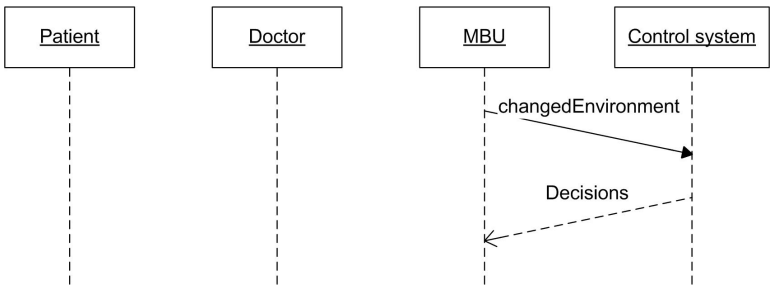


Figure 6.4: Sequence diagram based on Mobihealth input

The behaviour described in Figure 6.6 can also be realised using policies. The input of the PDP is then used as input for special policies. These special policies decide which policies should be installed by the PDP to make decisions towards the controlled system. We have chosen not to use this system, and thus make the PDP more complex, because our method makes a more clear distinction between the user preferences processing and policies or decisions that control the controlled system.

Figure 6.7 describes the behaviour of the system when environment information is received. The environment information is received by the EM. When information has been changed, the PDP is informed. The PDP then may send one or more decisions to the PEP.

Communication between components As described in section 3.6.1, the communication between components can take place using the request-response or the event-based mechanism. The mechanism that is used depends on the communication between entities. When two-way communication takes place, the request-response mechanism will be used, and when one-way communication is used, the event-based mechanism will be used.

Event-based messaging will be using the following event message structure:

$name, < parameter, value >^*$

The name represents the name of the event. The tuple, consisting of a parameter and its value, contains event specific information. The parameter contains the name of the parameter and the value contains the value of the parameter. Each event contains one or more tuples. When an event with a certain name is send, all components that have registered itself to the event registry with that name, will receive the event.

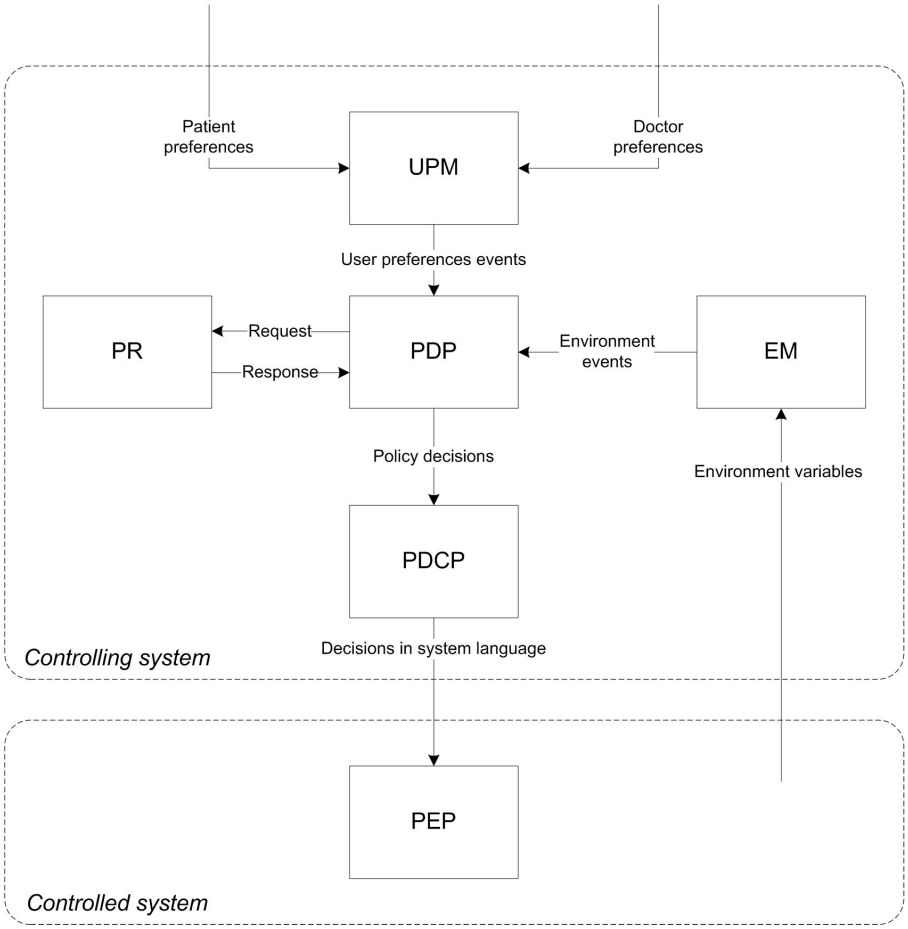


Figure 6.5: Overview of the specification

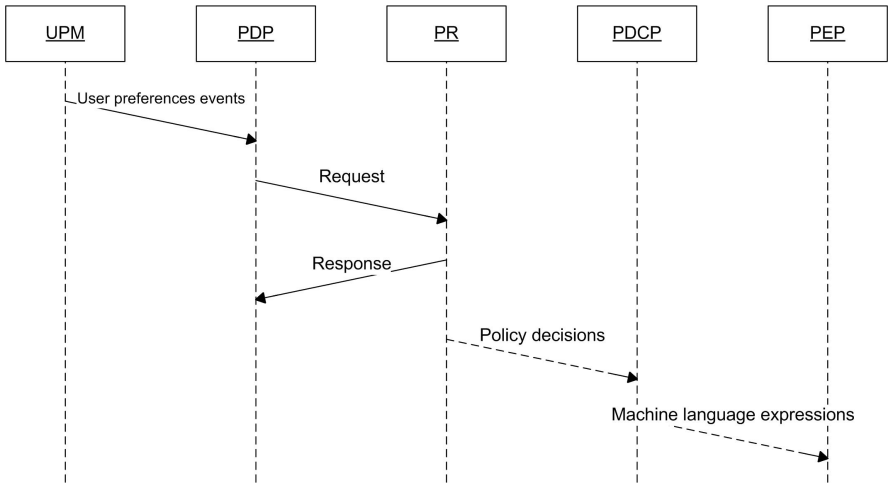


Figure 6.6: Sequence diagram of user preference input

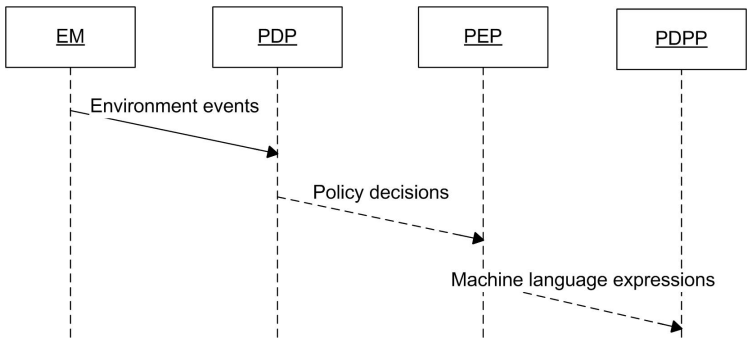


Figure 6.7: Sequence diagram of environment information input

An example of an event message is:

$ConnectionChanged, < Type, GPRS >, < ID, 001 >, < Available, False >$

This event message is send from the EM towards the PDP. The event message tells the PDP that connection information has changed and in this case that a connection with ID 001 and of type GPRS is not available.

Physical location of the PDP The IETF policy framework states some possible implementations concerning the physical intergration of the PEP and the PDP. The physical location of the PDP can be, as stated in section 3.6.1, at three locations: external, internal and combined. The requirements (Section 2.5) state the need for the policy-based control system to run on the MBU. As stated in section 2.2.3, the MBU is currently a PDA. This PDA has a communication connection to communicate to external sources. It is not guaranteed that this connection is always available. This makes the need for a PDP that is physically available on the MBU. Figures 3.7, 3.8 and 3.9 showed that only the combined PDP or the internal PDP are the options with the PDP physically in the system. The requirements also state the need to keep the control system as lighweigth as possible. Therefore we choose the simplest physical PDP location, the internal PDP, for this control system.

6.3 Refined internal system perspective

To refine Figure 6.5 further, each of the five components will be decomposed. The specification of each of the components, including interactions inside each component, will be given. In the following sections we describe the details of each of the components that are part of the overview of the specification.

6.3.1 User preferences monitor

As stated in section 3.6.2, there are three possible user preferences components: the forwarding UP, the weigthed UP and the meta policy UP. The problem analysis in chapter 1 state the scope of this master thesis. The main focus of this thesis is on policies and not on the policy model. Therefore we choose the most basic solution for the user preferences component, the forwarding UP.

The user preferences monitor component is triggered when a new user preference is made by the patient or the doctor. Due to constrains on this thesis, the user preferences monitor only forwards these messages to the PDP. Future work on a user preferences monitor can enhance this component by, for example, adding some weighing to the preferences.

6.3.2 Environment monitor

The environment monitor receives information from the environment of the mobihealth system. It caches the information as variables and will send an event to the PDP if the environment has changed. This event contains the name of the variable that has changed since the last event and all the variables that are known to the EM. The variables that are known by the EM are the variables that are described in Section 5.2.4.

6.3.3 Policy repository

The policy repository consists of two components as can be seen in Figure 6.8. Incoming requestsprocessed by the *PolicyGroup Retriever*. This component processes the incoming query to match the database query language of the other component, the *PolicyGroup Database*. When the query is processed, it is forwarded to the PolicyGroup Database. This component looks up which PolicyGroup belongs to the request and returns the PolicyGroup to the retriever. The retriever forwards the response to the requesting component.

6.3.4 Policy decision point

In our system the Policy Decision Point (PDP) is the component where user preferences are processed, policies reside and policies are evaluated. All events from the UPM and the EM are received by the PDP. Figure 6.9 shows the inner components of the PDP. The PDP contains two parts: the *PolicyGroup updater* and the *PolicyGroup evaluator*. Each part processes one type of input.

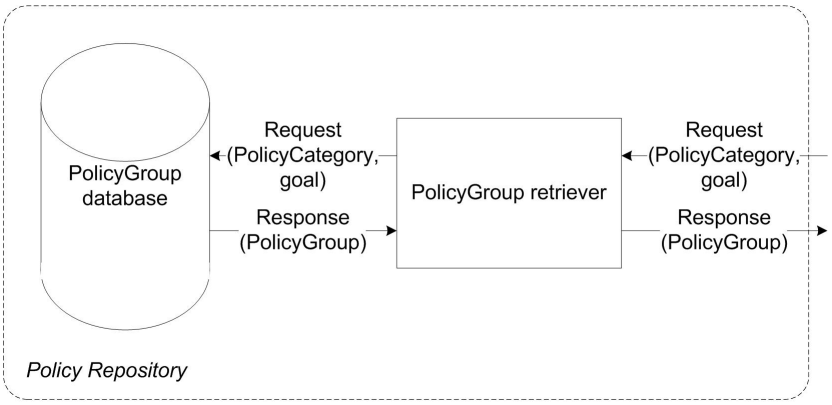


Figure 6.8: Decomposition of the PR

The *PolicyGroup updater* responds to events from the UPM. When such an event is received, the *PolicyGroup retriever* will request a PolicyGroup from the PR based on the PolicyCategory and goal, specified in a user preference. When a PolicyGroup is retrieved, it will forward this to the *PolicyGroup installer*. This component will uninstall the old PolicyGroup and install the new PolicyGroup. The installer will then send an event to the *PolicyGroup trigger* stating that policies have changed.

The *PolicyGroup evaluator* responds to events from the EM, or to the *PolicGroup installer*. When such an event is received, the policy trigger will forward this event to the PolicyGroups. These PolicyGroups reside inside the PDP. If the evaluation of a PolicyGroup results in one or more decisions, the PolicyGroup will send an event to the *policy decision combiner*. This component will collect the decisions from policies and forward these to the PEP.

As described earlier, it is also possible to incorporate the two parts into one. This results in a meta-policy system which selects which policies have to be installed into the PDP. An advantage of this method is the less complex PDP, but a disadvantage is the intertwined policy decision system, which is used by the normal policies and the meta-policies, which makes it more difficult to understand the system.

6.3.5 Policy Decision Conversion Point (PDCP)

The PDCP receives decisions that are made by the PDP. It changes the decisions in such a way that the controlled system can understand the decisions. When, for example, the decision is 'switch to GPRS', the PEP will rewrite this decision to, for example, $disableCurrentConnection(); enableGprs()$. The converted decision is send to the PEP.

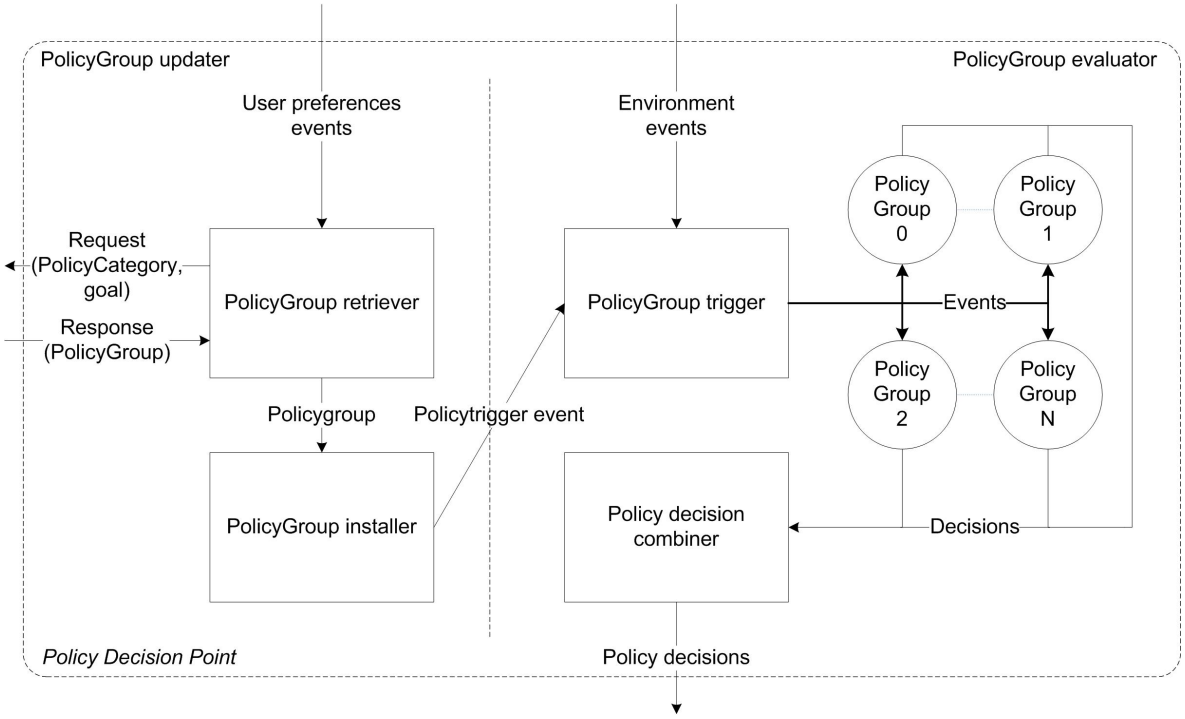


Figure 6.9: Specification of the PDP

This component is not present in other policy models. We introduce this component to make it possible for multiple systems to use the same policies. When there are, for example, two types of MBU, the same policies can be used to make decisions, but different PDCP's are required to translate the decisions to the machine language of each MBU.

6.3.6 Policy Enforcement Point

The PEP receives the transformed decisions from the PDPP and will execute these onto the MBU. The execution of the decision is outside the scope of this thesis. Instead of enforcing the actual decision, the PEP will only show a message to the end-user, which contains the decision.

Chapter 7

Implementation of policies and the policy-based control system in the OSGi framework

In the previous chapters a policy specification and architecture is presented. This chapter will describe and discuss how these are implemented in the OSGi framework.

This chapter contains three parts. First, we describe how the policy itself is implemented in OSGi. The second part present the implementation of the policy-based architecture. The last part will present the prototype that combines these two implementations.

7.1 Policy implementation in OSGi

The description of the policy implementation in OSGi is divided into three parts. The first section describes how the policy is implemented into bundles. The second section explains the communication between the policy bundles. The third section describes the implementation of the policies including pseudo code.

7.1.1 Policies as bundles

There are several possible modelling options on how the policy can be implemented in the OSGi bundle structure.

The strength of the OSGi framework is the installing, updating and uninstalling of bundles at runtime. This property should be used in such a way that policies benefit the most.

We will implement a PolicyGroup as a single bundle, because this has the following advantages:

- As described in Section 5.1 there will be one enabled PolicyGroup for each PolicyCategory at a certain moment in time. A PolicyGroup is implemented as one bundle, which makes it easy for the control system to enable or disable a single PolicyGroup. If a group has the state enabled, the corresponding bundle is installed and started in the OSGi framework. If a group has the state disabled, the corresponding bundle is not installed in the framework. This means that only enabled groups are installed and running.
- Triggering of the execution of the policies belonging to a single PolicyGroup is easy. The control system only has to look up the bundle which contains the PolicyGroup he wants to trigger and send a trigger message to the bundle.

Bundles that represent a PolicyGroup are available at a bundle repository. Such a repository is a website which contains bundles. These bundles can be downloaded and installed in the OSGi framework.

7.1.2 Communication interface of the policy

As described in Section 6.3.4, policies communicate with one architecture component: the PDP. The PDP tells the policies, through the PolicyGroup, when they have to evaluate themselves and the policies may tell the PDP a decision. So there are two different communication types: input and output. These will be described in the next paragraphs.

Input The triggering of PolicyGroups information will be sent from the PDP towards the PolicyGroups. This information contains a list of variables from the environment of the MBU and a string which contains the event that triggered the evaluation of the PolicyGroup. We have chosen to implement this communication as a service invocation, because this enables the PDP to transparent search for PolicyGroups that are available inside the framework. This is visualized in Figure 7.1.

The interface, that will support this service, has to contain one method which will be used as the trigger. The method is called by the PDP when a policy has to be triggered. The PDP sends information to the policy group which will be used to make a decision. This information consists of two parts. A list of environment variables and a string which contains the event that triggered the execution of the policy group. The event is used by the condition inside a poliy to determine if it has to be evaluated.

Output The evaluation of a policy does not necessary have to result in a decision, as described in Section 5.2. Therefore it is better to have this decision be sent as an event message instead of contacting the PDP through a service. The PDP will have an event sink which listens to the events that are transmitted by the policies. This is visualized in Figure 7.1.

7.1.3 Policy implementation

The implementation of the policies will follow their specification 5.2. Therefore the implementation of each PolicyGroup will consist of two parts. The first part contains the generic policies and the second part the specific implementation of the policies. Both parts will be described in the next paragraphs. The last paragraph of this section will give an example.

Generic PolicyGroup The generic part contains an implementation of generic solution that is described in Section 5.2.3 and will reside in a separate bundle. This implementation is the same for all policies that belong to the same policy category. All parts of the implementation that are not generic, are made variable. These variables belong to the specific part and are not the same for each policygroup. This bundle contains the standard interface for all PolicyGroups, generic policies for all PolicyGroups and the implementation of the methods that are used inside these policies.

Specific PolicyGroup The specific part of each PolicyGroup contain the goals of the policygroup. The specification of the goals is described in Section 5.2.2. The goals are implemented in the PolicyGroups as variables. These variables are used in the generic PolicyGroup to make a decision. An example of the implementation of a specific policygroup is given in the following pseudo code.

```
public class PolicyGroupImplementation extends
                          BasicPolicyGroupImplementation {

   public PolicyGroupImplementation(BundleContext context) {
      super(context);
      policyTarget = new PolicyCategory(PolicyCategory.Connection);
   }

   public void makeDecision(String event){
      SensorCriteria[] criteria = new SensorCriteria[3];
      //parameters of SensorCriteria:
      //ID, minimum SR, pref minimum SR, pref average SR,
      //maximum SR, group number.
      //Priority of a goal is defined by its location in the
      //criteria array.
      criteria[0] = new SensorCriteria("ECG",64,128,512,1024,0);
      criteria[1] = new SensorCriteria("NIBP",8,16,32,64,1);
      criteria[2] = new SensorCriteria("SPO2",1,2,4,8,1);
      MakeDecisionConnection(event,criteria,
                                ConnectionCriteria.MODERATE);
   }
}
```

The pseudo code is a class which extends the class BasicPolicyGroupImplementation. The class BasicPolicyGroupImplementation contains all generic PolicyGroups and implements the Policy interface. This interface contains, as described earlier, one method 'makeDecision' which is triggered by the PDP. This pseudo code example has as target 'Connection', which is assigned in the constructor of the PolicyGroup.

The implementation of the 'makeDecision' method contains the goals of the policy. In this case three sensors are present in the goals of the sensors. In the implementation (a part of) a goal is called 'criteria'. After the goals of the sensor are specified, a generic policy is triggered, in this case 'MakeDecisionConnection'. This method contains the generic policy for the target 'Connection'. The parameters of this method contain the event that triggered the execution of the policy, the goals of the doctor and the goal of the patient.

7.2 OSGi implementation of the policy-based architecture

This section will describe the implementation of the policy-based architecture into the OSGi framework. Section 7.2.1 describes how the components of the architecture are implemented in OSGi. Section 7.2.2 describes how the communication between these components is implemented.

7.2.1 Architectural components as bundles

. As a base the policy-based architecture, described in Section 6.2, is used. This architecture is mapped onto the OSGi bundle structure. Just as in the previous section, there are several options on how this can be done. To maintain the advantages of the OSGi framework, we choose to use five bundles which represent the five components of the architecture. The six bundles, i.e. UPM, EM, PR, PDP, PDCP, and PEP, and the policies, are visualized in Figure 7.1.

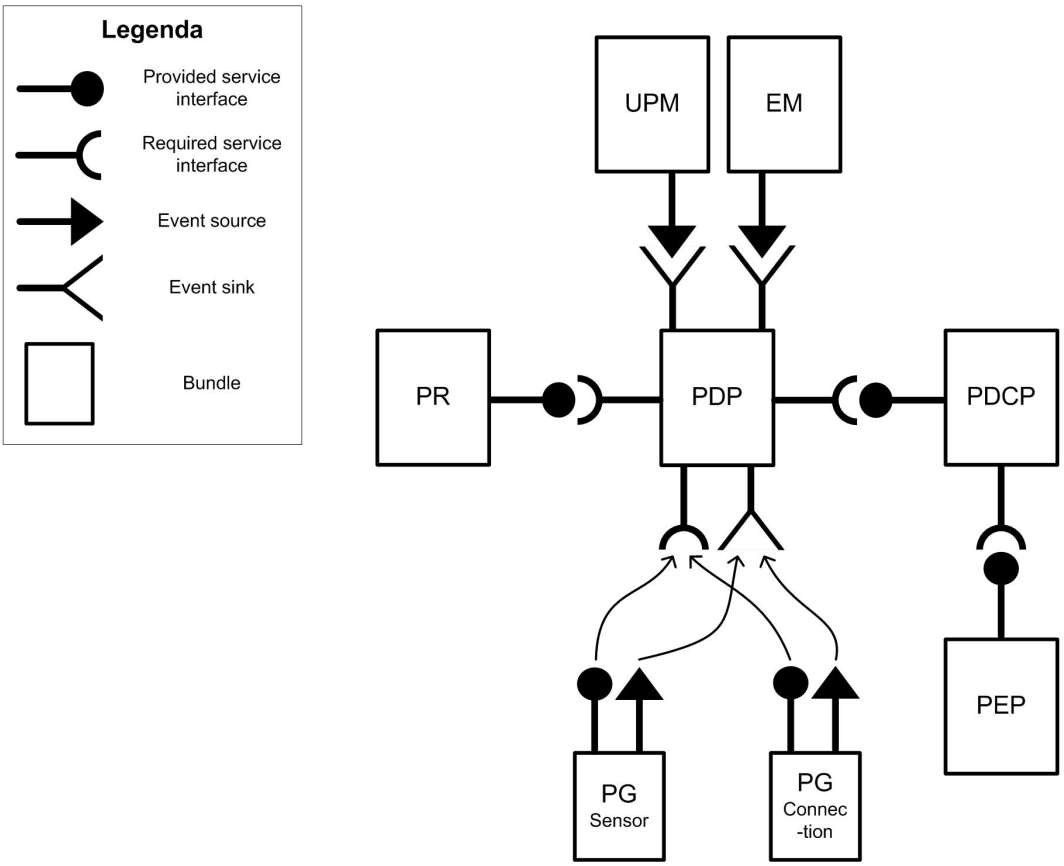


Figure 7.1: OSGi bundle model

7.2.2 Communication

As described in Section 4.5, OSGi event based communication is implemented using an event admin service and request-response based communication is implemented as communication between services. The choice which communication method is used for each interaction between bundles depends on the type of interaction that is defined in the architecture. The UP and EM component use event messages to send information towards the PDP. These event mechanisms will be implemented with the help of the OSGi event admin service. The communcation between the PDP and the PR and between the PDP and the PEP is request-response based and will be implemented as services communication.

As described earlier, services need an interface to communicate. The communication between the PDP and the PR and the PDP and the PEP are both initiated by the PDP, as visualized in Figure 6.5. This means that the PR and the PEP need to provide services.

7.3 Prototype implementation and examples

This section will describe how the prototype is implemented by describing the assumptions that are made, a UML class diagram description and some examples.

7.3.1 Assumptions

The implementation that is presented in this chapter is a prototype. Therefore some assumptions have been made to facilitate the proper functioning of the prototype. These assumptions are described and explained in the following list.

- Input: The input of the system depends on the environment of the MBU and the user preferences. The environment of the MBU has been modelled by a list of variables. This list of variables can be changed at runtime by the system administrator. This simulates the changing of the environment.
- Output: The output is based on decisions that will be enforced. This is not possible in the prototype, because there is no MBU present. Therefore the output of the system will be a message which contains the decisions that have to be enforced.

To facilitate these assumptions an extra bundle has been developed which implements a html based GUI. This GUI enables the user of the prototype to change the input and see the output of the system.

Implementation A UML class diagram of this prototype is shown in Figure 7.2. White classes are classes that are developed for this problem and grey classes are standard services which are present in the OSGi framework. Classes that are in the same bundle have a dashed square around themselves including the name of the bundle. A number of classes are not present in this Figure to keep a clear overview of the important components. The left out classes are all the Activator classes (see Section 4.2) which are used by the bundles to communicate with the OSGi framework. Also the GUI, including the extra interfaces and services that are required for its functioning, are not present in the class diagram.

7.3.2 Settings

This section describes the settings that are available in the prototype. It consists of the possible variables of the environment and the preferences of the users when the prototype is started.

Environment input The environment input is divided in two parts: sensors and connections. There are three sensors available in the prototype: ECG, NIBP and SPO2. Their sample rate and sample size is described in Table 7.1. The sample rate is the rate which is used by the sensor to transmit its data to the MBU. As explained earlier, this (incoming) sample rate cannot be changed.

Type	ID	Sample size (bits)	Sample rate	Availability
ECG	S001	24(*4 channels)	1024	1
SPO2	S002	24	128	1
NIBP	S003	24	2	1

Table 7.1: Available sensors in the prototype

There are six possible connections available in the prototype. All connections are from the type GPRS but have different coding and upload slots. Table 7.2[1] shows the different connections that are supported by the prototype. The bandwidth values are calculated using the analysis of GPRS and UMTS, described in Section 2.1.

Preferences The preferences of the doctor are specified in Table 7.3. All sensors that are available in the prototype do have a preference specified. The preference of the patient specifies the preferred relative cost, as described in Section 5.2.2. The possible preferences are 'Cheap','Moderate' and 'Expensive'. We choose to set the preference of the patient to 'Cheap' at the start of the prototype.

[1] CS: Coding Scheme; US: Upload Slots.

Type	ID	CS	US	Bandwidth (bps)	Cost per byte (cents)	Availability
GPRS	C001	1	2	18534	1	1
GPRS	C002	2	2	27443	1	1
UMTS	C003	-	-	30720	1	1
UMTS	C004	-	-	147456	1	1

Table 7.2: Available connections in the prototype

Sensor	Minimum	Preferred minimum	Preferred average	Priority	Group
ECG	Low	Medium	High	0	0
NIBP	Medium	Medium	High	1	0
SPO2	Low	Low	Low	2	1

Table 7.3: Doctor preferences

Table 7.4 shows the conversion from sensor sample rate labels, specified by the doctor, towards actual sample rates for each supported sensors. This table is required to have else it is not possible to convert a preference towards a goal.

7.3.3 Examples

The following paragraphs will first describe which decisions are made by the control system when it is started with the above settings. The last two paragraphs will show what happens when the environment changes or when user preferences change.

Initial state In the initial state of the system, the sensors, connections and preferences are specified as described above. The system will then start the system by installing PolicyGroups based on the preferences.

First the PolicyGroup with target 'Connection' will be triggered to select a communication channel. The preference of the patient is 'Cheap', so the minimum available bandwidth should be the amount of bandwidth that is required in the 'preferred minimum sample rate' category, as described in Table 5.4. The policy will now be executed and should select the cheapest connection that satisfies this condition. The minimum available bandwidth is calculated by multiplying and adding sample rates and sample sizes and is: $256 * 96 + 1 * 24 + 16 * 24 = 24984bps$. Thus only the connections with ID C002, C003 and C004 can be used, because the other connection does not have enough bandwidth available. The cost per byte of each connection is 1, therefor the connection with the lowest amount of bandwidth available, C002, should be selected by the MBU as communication channel.

ECG		NIBP		SPO2	
Label	*Sample rate*	*Label*	*Sample rate*	*Label*	*Sample rate*
Low	128	Medium	1	Low	16
Medium	256	High	2	Medium	32
High	512			High	128
Very High	1024				

Table 7.4: Sample rate labels

Next the amount of available bandwidth of connection C001, 18534 bps, has to be allocated among the sensors in the preferences. This is done according to the algorithm described in Table 5.3 and results in the allocation shown in Table 7.5.

Sensor	**Allocated sample rate**
ECG	188
NIBP	2
SPO2	18

Table 7.5: Sensor bandwidth allocation

First the value of the 'preferred minimum sample rate' will be allocated to all sensors: ECG will have a sample rate of 128, NIBP a sample rate of 2 and SPO2 a sample rate of 16. The bandwidth that is available after this allocation is: $18534 - 12696 = 5838bp$, and will be allocated to the sensors based on their priority. The ECG sensor has priority 0 and will get extra bandwidth as a first. The increase in sample rate of the ECG sensor will be $5838/96 = 60$. The bandwidth that is available after this allocation is: $5838 - 60 * 96 = 78bps$. This allocation process is repeated until the amount of available bandwidth is 0 or it is not possible to increase the sample rate of a sensor.

When the MBU has switched to connection C002, the sensor allocation policy is triggered again and the new amount of available bandwidth is distributed amongst the sensors. The result of this allocation is visible in Figure 7.3. This figure shows a screenshot of the prototype. It can be seen that the connection has been changed from C001 to C002, including the new sensor allocation.

Connection is unavailable To show the dynamics of the system, it is modelled that the current communication channel with ID C002 becomes unavailable. The control system now should select a new communication channel. The preferences of the users are still the same, so a new communication channel that satisfies the preferences should be selected. The description of the connection selection in the initial state showed that there are three valid

connections: C002, C003 and C004. C002 is no longer available, so C003 or C004 should be selected. The cost of a connection is based on their amount of available bandwidth. C003 has the least amount of available bandwidth and thus should be selected.

Figure 7.4 shows the result of the selection of a new connection and allocation of the new amount of available bandwidth among the sensors.

Preference of doctor is changed As a second example, the preference of the doctor is changed from the initial setting to a preference which is visible in Table 7.6. The control system calculates if these settings require a new connection and will allocate the current available bandwidth to the sensors based on the new preferences. Figure 7.5 shows the result of this preference change including the switching of connection and bandwidth allocation.

Sensor	Minimum	Preferred minimum	Preferred average	Priority	Group
ECG	Medium	High	Very High	0	0
NIBP	High	High	High	1	1
SPO2	Low	Medium	High	2	2

Table 7.6: Updated doctor preferences

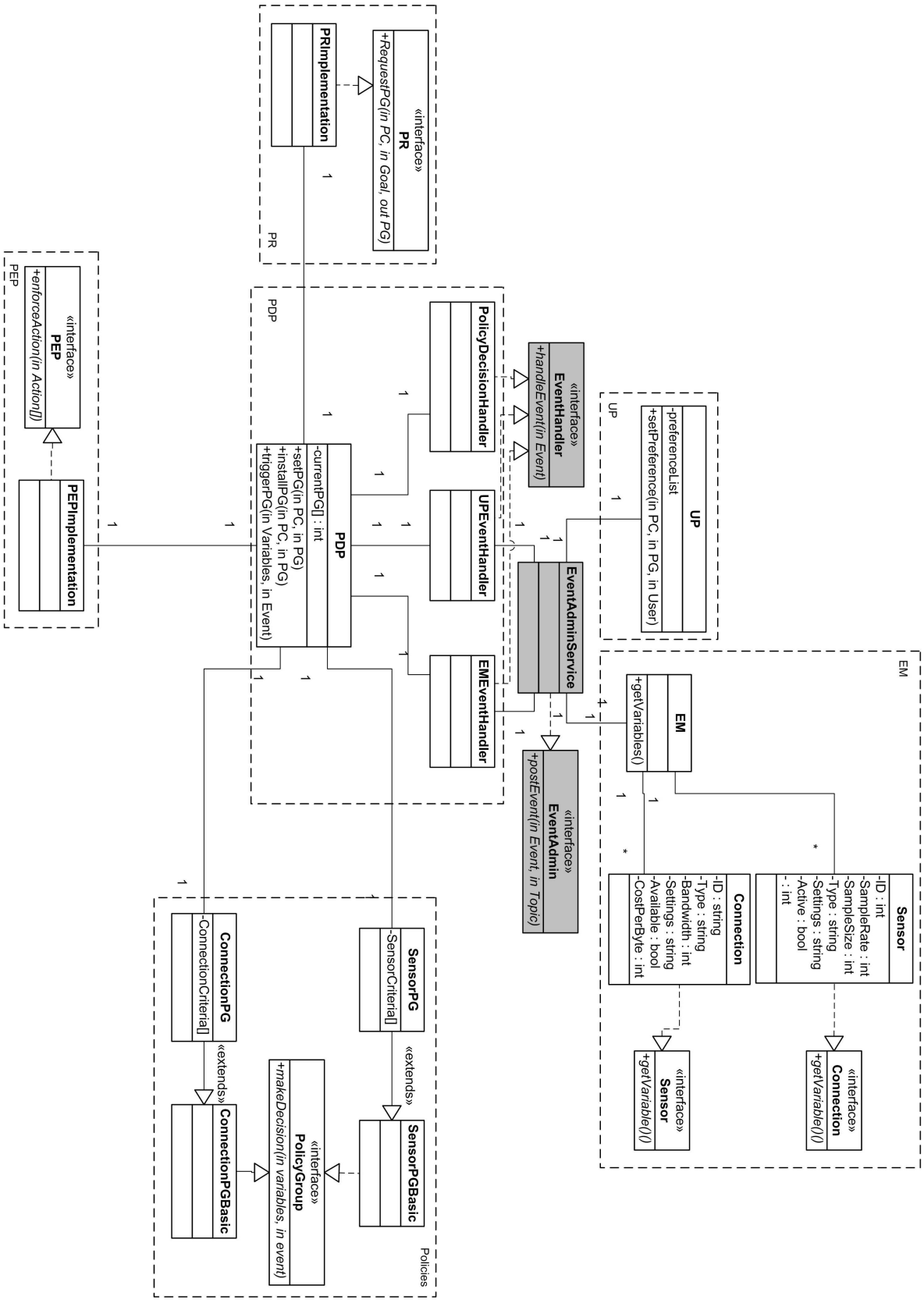
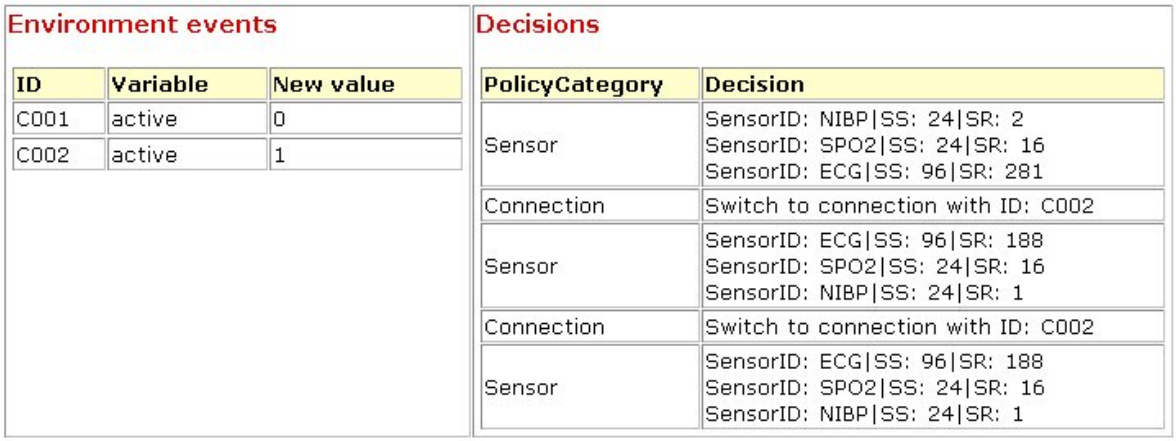


Figure 7.2: UML class diagram of the implementation of the architecture



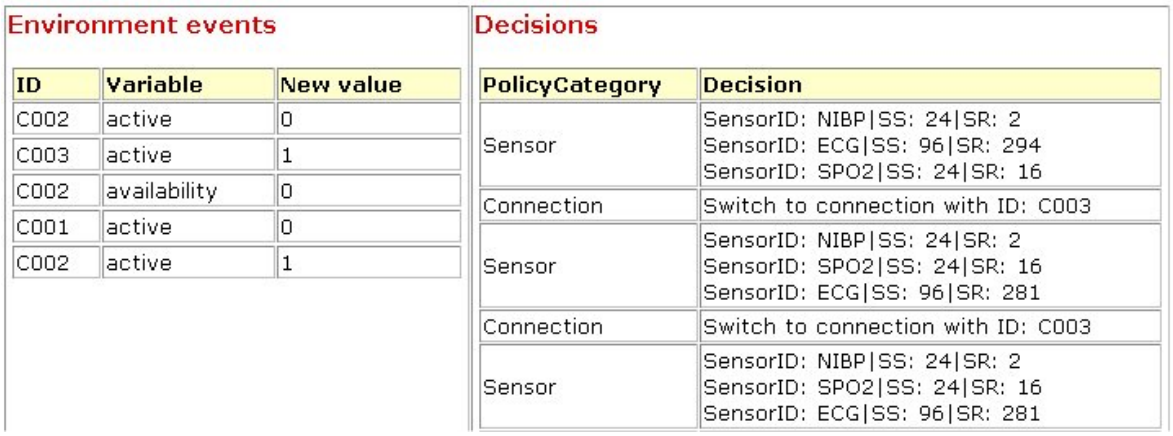
Environment events

ID	Variable	New value
C001	active	0
C002	active	1

Decisions

PolicyCategory	Decision
Sensor	SensorID: NIBP|SS: 24|SR: 2 SensorID: SPO2|SS: 24|SR: 16 SensorID: ECG|SS: 96|SR: 281
Connection	Switch to connection with ID: C002
Sensor	SensorID: ECG|SS: 96|SR: 188 SensorID: SPO2|SS: 24|SR: 16 SensorID: NIBP|SS: 24|SR: 1
Connection	Switch to connection with ID: C002
Sensor	SensorID: ECG|SS: 96|SR: 188 SensorID: SPO2|SS: 24|SR: 16 SensorID: NIBP|SS: 24|SR: 1

Figure 7.3: Screenshot of the prototype in the initial state.



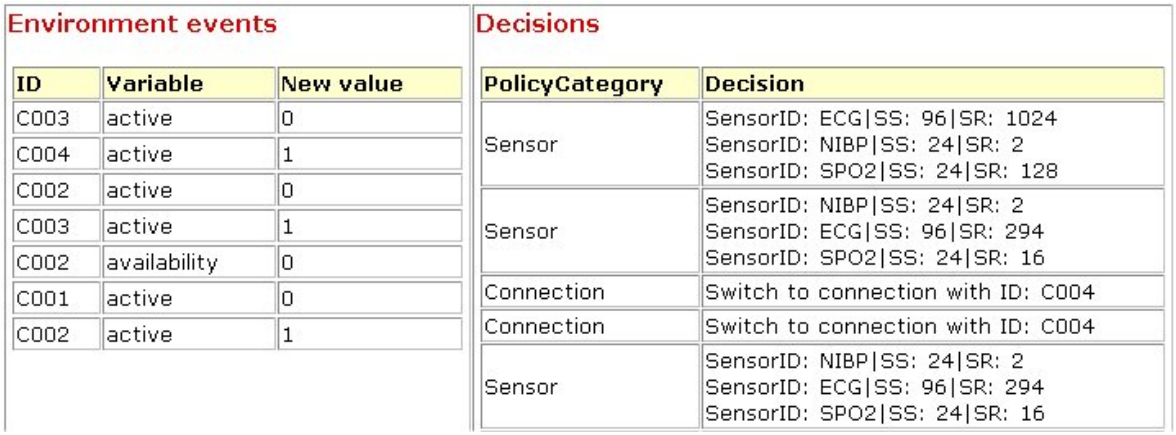
Environment events

ID	Variable	New value
C002	active	0
C003	active	1
C002	availability	0
C001	active	0
C002	active	1

Decisions

PolicyCategory	Decision
Sensor	SensorID: NIBP|SS: 24|SR: 2 SensorID: ECG|SS: 96|SR: 294 SensorID: SPO2|SS: 24|SR: 16
Connection	Switch to connection with ID: C003
Sensor	SensorID: NIBP|SS: 24|SR: 2 SensorID: SPO2|SS: 24|SR: 16 SensorID: ECG|SS: 96|SR: 281
Connection	Switch to connection with ID: C003
Sensor	SensorID: NIBP|SS: 24|SR: 2 SensorID: SPO2|SS: 24|SR: 16 SensorID: ECG|SS: 96|SR: 281

Figure 7.4: Screenshot of the prototype after C003 is selected.



Environment events

ID	Variable	New value
C003	active	0
C004	active	1
C002	active	0
C003	active	1
C002	availability	0
C001	active	0
C002	active	1

Decisions

PolicyCategory	Decision
Sensor	SensorID: ECG|SS: 96|SR: 1024 SensorID: NIBP|SS: 24|SR: 2 SensorID: SPO2|SS: 24|SR: 128
Sensor	SensorID: NIBP|SS: 24|SR: 2 SensorID: ECG|SS: 96|SR: 294 SensorID: SPO2|SS: 24|SR: 16
Connection	Switch to connection with ID: C004
Connection	Switch to connection with ID: C004
Sensor	SensorID: NIBP|SS: 24|SR: 2 SensorID: ECG|SS: 96|SR: 294 SensorID: SPO2|SS: 24|SR: 16

Figure 7.5: Screenshot of the prototype after the preference of the doctor is updated.

Chapter 8

Conclusion

In this Chapter we review the contribution of this thesis, our conclusions and describe future work. This chapter is structured as follows: Section 8.1 presents our conclusions. It also summarizes the main contribution of this thesis. Section 8.2 identifies some future work.

8.1 Conclusion and contributions

The description of the problem that is researched in this thesis is the following: *How should a policy-based MBU control mechanism, that supports environment changes and user preferences, be specified and implemented?*

In this thesis such a policy-based MBU control mechanism has been discussed and designed. This policy-based control mechanism enhances the MBU with the possibility to adapt to environment changes and user preferences. Our main contributions, described in this thesis, are:

- *Extention of the IETF policy structure.*
  The policy structure, described in the IETF policy core information model, is extended with a policy set component. This component groups together policies that influence the same target. Also a goal attribute is added to group policies in a set that share the same goal.
- *Specification of policies.*
  The extended policy structure is used to specify policies for the policy-based MBU control system.
- *Extention of the IETF policy model.*
  The IETF policy model is extended with a user preferences component which makes it

possible to influence policies based on these preferences.

- *Implementation in the OSGi framework.*
  The policy specification and architecture are implemented in the OSGi framework. The implementation showed how the specification and the architecture have to be implemented to make the best use of the framework.

The implementation of the policy specification and architecture resulted in a prototype. This prototype has been tested and showed the feasibility of the developed specification and architecture. The test results show the adaptability of the policy-based MBU control system when environment changes occur or when new preferences are expressed towards the system.

8.2 Future work

An addition to the usability of the policy-based control system is distribution. In our system a doctor has to state preferences for each patient he is monitoring. MBU's that have the same monitoring preferences cannot share these preferences. Future research could be performed on how preferences could be combined together to make it possible for a doctor to control multiple MBU's using one control system.

Our system makes it possible for a doctor to state the preferred resolution for each sensor that is worn by a patient. Often these preferences will be the same for one monitoring type, e.g. heart rhythm impairments. Research could show which monitoring types are often used and which resolutions and preferences belong to such a type. This grouping of preferences would simplify the configuration of the preferences by a doctor.

In our system a doctor can only select a preference from a list of sensor configurations. If he wants to change the resolution of a sensor, he has to select a new preference that incorporates this resolution. When the preferred resolution is not present in a preference, the doctor cannot select the required preference. It is also not possible for him to add this new preference to the list of sensor configurations. Research could show how a policy engine could facilitate the updating or adding of new preference possibilities.

Appendix A

Policy pseudo code

This appendix contains pseudo code of the generic connection policy, generic sensor policy and the functions they use. If explanation of the pseudo code is required, comment is added.

A.1 Generic connection policy

The following pseudo code is an example of a generic connection policy. This code selects a connection that has at least an available bandwidth that is required by preferred minimum sample rate, specified in the preferences of the doctor.

```
if (  (event == "ConnectionVariableChanged" ||
       event == "UserPreferenceChanged") &&
       SelectPossibleConnections(CalculatePreferredMinimumRequiredBW(doctorPreference),
                                 AvailableConnections(),
                                 result) &&
       currentConnection != selectCheapestConnection(result,result2))
then
{
    SwitchToConnection(result2);
}
```

A.2 Generic sensor policy

Abbreviations:

- bw = bandwidth;
- sbw = sample bandwidth;
- sample bandwidth = sample rate * sample size.

Bandwidth category E (available bw > maximum sbw).

```
if (  (event == "ConnectionVariableChanged") ||
       event== "SensorVariableChanged") &&
       IsFirstSmaller(CalculateMaximumBW(doctorPreference),
                      BandwidthAvailablev)) &&
       SelectOptimalSensorDataConfigurationE(doctorPreference,
                                             BandwidthAvailable(),result) )
then
{
    EnforceConfigurationOnSensorData(result);
}
```

Bandwidth category D (maximum sbw > available bw > preferred average sbw).

```
if (  (event == "ConnectionVariableChanged") ||
       event== "SensorVariableChanged") &&
       IsFirstSmaller(CalculatePreferredAverageBW(doctorPreference),
                      BandwidthAvailable()) &&
       IsFirstSmaller(BandwidthAvailable(),
                      CalculateMaximumBW(doctorPreference)) &&
       SelectOptimalSensorDataConfigurationD(doctorPreference,
                                            BandwidthAvailable(),result) )
then
{
    EnforceConfigurationOnSensorData(result);
}
```

Bandwidth category C (preferred average sbw > available bw > preferred minimum sbw).

```
if (  (event == "ConnectionVariableChanged") ||
       event== "SensorVariableChanged") &&
       IsFirstSmaller(CalculatePreferredMinimumBW(doctorPreference),
                      BandwidthAvailable()) &&
       IsFirstSmaller(BandwidthAvailable(),
                      CalculatePreferredAverageBW(doctorPreference)) &&
       SelectOptimalSensorDataConfigurationC(doctorPreference,
                                           BandwidthAvailable(),result) )
then
{
    EnforceConfigurationOnSensorData(result);
}
```

Bandwidth category B (preferred minimum sbw > available bw > minimum sbw).

```
if (  (event == "ConnectionVariableChanged") ||
       event== "SensorVariableChanged") &&
       IsFirstSmaller(CalculateMinimumBW(doctorPreference),
                      BandwidthAvailable()) &&
       IsFirstSmaller(BandwidthAvailable(),
                      CalculatePreferredMinimumBW(doctorPreference)) &&
       SelectOptimalSensorDataConfigurationB(doctorPreference,
                                             BandwidthAvailable(),result) )
then
{
    EnforceConfigurationOnSensorData(result);
}
```

Bandwidth category A (minimum sbw > available bw).

```
if (  (event == "ConnectionVariableChanged") ||
       event== "SensorVariableChanged") &&
       IsFirstSmaller(BandwidthAvailable(),
                      CalculateMinimumRequiredBW(doctorPreference)) &&
       SelectOptimalSensorDataConfigurationA(doctorPreference,
                                             BandwidthAvailable(),result) )
then
{
    EnforceConfigurationOnSensorData(result);
}
```

A.3 Function library

This section contains pseudo code of the functions that are used in the generic policies.

A.3.1 SelectPossibleConnections

This method selects the possible connections that conform to the specified required bandwidth and returns an array of valid connections.

```
protected Connection[] SelectPossibleConnections(int bandwidthRequired, Connection connections[]) {
   Vector tempreturnlist = new Vector();
   for (int i = 0; i < connections.length; i++) {
      if (!IsFirstSmaller(connections[i].getThroughput(), bandwidthRequired))
         tempreturnlist.addElement(connections[i]);
   }

   Connection[] returnlist = new Connection[1];
   if (tempreturnlist.size() > 0){
      returnlist = new Connection[tempreturnlist.size()];
      for (int i = 0; i < tempreturnlist.size(); i++){
         returnlist[i] = (Connection)tempreturnlist.elementAt(i);
      }
   }
   else {
      //The connection with the highest possible bandwidth should be returned, because there is no
      //correct connection available.
      int tempbw = 0;
      Connection tempc = new Connection();
      for (int i = 0; i < connections.length; i++) {
         if (tempbw < connections[i].getThroughput()) {
            tempbw = connections[i].getThroughput();
            tempc = connections[i];
         }
         returnlist = new Connection[1];
         returnlist[0] = tempc;
      }
   }
   return returnlist;
}
```

A.3.2 AvailableConnections

```
protected Connection[] AvailableConnections(Hashtable variables) {
   //return an array of connections that are currently available, thus have an available bandwidth
   //that is higher than 0.
}
```

A.3.3 CalculatePreferredMinimumRequiredBW

```
protected int CalculatePreferredMinimumRequiredBW(SensorCriteria criteria[]) {
   //calculate the amount of bandwidth that is required for each sensor by multiplying the sample rate
   //and sample size. Add the amounts of all sensors and return this value
   //Dit doe je door de criteria van de doctor te verwerken

   int bandwidthRequired = 0;
   for (int i = 0; i < criteria.length; i++){
      bandwidthRequired += criteria[i].getSampleRatePreferredMinimum()*criteria[i].getSampleSize();
   }
   return bandwidthRequired;
}
```

A.3.4 SelectCheapestConnection

```
protected Connection SelectCheapestConnection(Connection connections[]) {
   int tempcost = COST_INFINITE;
   Connection result = new Connection();
   for (int i = 0; i < connections.length; i++) {
      if ((connections[i].getCostPerByte()*connections[i].getThroughput()) < tempcost) {
         result = connections[i];
         tempcost = connections[i].getCostPerByte()*connections[i].getThroughput();
      }
   }
   //return the ID of the connection
   return result;
```

A.3.5 IsFirstSmaller

This method compares two bandwidth variables to check if bandwidth1 is smaller than bandwidth2. A threshold is used to make use the available bandwidth is always sufficient for the amount of required bandwidth.

```
private final int BANDWIDTH_THRESHOLD = 10; //percentage

protected boolean IsFirstSmaller(int BWAvailable, int BWRequired) {
   if (BWAvailable < BWRequired * (1+(BANDWIDTH_THRESHOLD/100)))
      return true;
   return false;
}
```

A.3.6 BandwidthAvailable

Return the amount of available bandwidth that the current connection supplies.

```
protected int BandwidthAvailable(Hashtable variables){
   return Integer.parseInt((String)variables.get("CONNECTION."+currentConnection+".bandwidth"));
}
```

A.3.7 SelectOptimalSensorDataConfigurationD

The following pseudo code is almost the same for sensor data categories B, C and E. The other sensor data categories (A and E) are slightly different, but follow the algorithm that it explained in Chapter 6.

```
public Sensor[] SelectOptimalSensorDataConfigurationD(SensorCriteria[] criteria, int bandwidthAvailable){
   //first assign the preferred bandwidth to all sensors.
   //then assign the extra bandwidth to the sensor with the highest priority
   Sensor[] result = new Sensor[criteria.length];
   for (int i = 0; i < criteria.length; i++) {
      SensorCriteria s = criteria[i];
      int bandwidthRequirement = s.getSampleRatePreferredAverage()*s.getSampleSize();
      result[i] = new Sensor(s.getId(),s.getSampleRatePreferredAverage(),s.getSampleSize());
      bandwidthAvailable -= s.getSampleRatePreferredAverage()*s.getSampleSize();
   }

   for (int i = 0; i < criteria.length; i++) {
      SensorCriteria s = criteria[i];
      int bandwidthRequirement = (s.getSampleRateMaximum()-s.getSampleRatePreferredAverage())*s.getSampleSize();
      if (bandwidthRequirement < bandwidthAvailable) {
         result[i] = new Sensor(s.getId(),(s.getSampleRateMaximum()-s.getSampleRatePreferredAverage())
                        +result[i].getSampleRate(),s.getSampleSize());
         bandwidthAvailable -= bandwidthRequirement;
      }
      else {
         //bepalen of de sr omlaag kan, zo nee, sr = 0
         if (s.getSampleSize() > bandwidthAvailable) {
            //sensor wont get any bandwidth
         }
         else {
            int allowedSampleRate = (int) Math.floor(bandwidthAvailable / s.getSampleSize());
            Sensor temp = new Sensor(s.getId(),allowedSampleRate+result[i].getSampleRate(),s.getSampleSize());
            result[i] = temp;
            bandwidthAvailable -= allowedSampleRate*s.getSampleSize();
         }
      }
   }
   return result;
}
```

Bibliography

[1] J. Tian. A policy-based mbu control system for m-health. Master's thesis, University of Twente, Faculty of Electrical Engineering Mathematics and Computer Science, ASNA GROUP, June 2006.

[2] Mobihealth. Mobihealth website, 2007, `http://mobihealth.org/`.

[3] Healthservice24. Healthservice24 website, 2007, `http://www.healthservice24.com/`.

[4] Awareness teletreatment. Awareness teletreatment website, 2007, `http://www.freeband.nl/project.cfm?id=494&language=en`.

[5] M. Baarda, V. Jones, and D. Stemerding. Ban-based m-health services: Experiences and prospects. In *Proc 4th International Conference on the Management of Healthcare & Medical Technology*, August 2005.

[6] Website cva congres innovationcentre revalidationtechniques, October `http://www.revalidatietechnologie.nl/cvacongres/images/context-aware.jpg`,2006. 2007-10-10.

[7] XMLSoap. Web services policy framework and web services policy attachment, 2007, `http://schemas.xmlsoap.org/ws/2004/09/policy/`.

[8] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. Ponder: A language for specifying security and management policies for distributed systems, October 2000.

[9] R. Yavatkar, D. Pendarakis, and R. Guerin. A framework for policy-based admission control – rfc 2753, January 2000.

[10] B. Moore, E. Ellesson, LongBoard Inc., J. Strassner, and A. Westerinen. Policy core information model – rfc 3060, Februari 2001.

[11] University of Twente, TMSI, Yucat, HP/Compaq, CMG, and UPF. Generic ban platform. Technical report, Mobihealth project, December 2002.

[12] A Melander-Wikman, M. Jansson, R. Herzog, D. Konstantas, and T. Scully. Overall evaluation of the mobihealth trials and services (d5.1), September 2004. 2007-02-27.

[13] K. E. Wac, A. T. van Halteren, and T. H. F. Broens. Context-aware qos provisioning for an m-health service platform. In C. D. Kloos, D. Larrabeiti, and A. Marin, editors, *EUNICE 2005 "Networked Applications". 11th open European Summer School, Colmenarejo, Spain*, pages 8–13, Madrid, Spain, July 2005. Universidad Carlos III de Madrid.

[14] Wikipedia. Gps, 2007, `http://en.wikipedia.org/wiki/Global_Positioning_System`.

[15] V. M. Jones, A. T. van Halteren, N. T. Dokovski, G. T. Koprinkov, J. Peuscher, R. G. A. Bults, D. Konstantas, I. A. Widya, and R. Herzog. Mobihealth: mobile services for health professionals. Technical Report TR-CTIT-06-38, Enschede, January 2006.

[16] V.M. Jones, H. Mei, T. Broens, I. Widya, and J. Peuscher. Context aware body area networks for telemedicine, `http://wwwhome.cs.utwente.nl/~meih/papers/PCM2007.pdf`, 2007.

[17] Portable gear. Qtek 9090 hardware specification. `http://www.portablegear.nl/reviews/Qtek_9090.htm`,2007.

[18] V. Jones, A. van Halteren, R. Bults, D. Konstantas, and I. Widya. Mobihealth: Mobile healthcare. Workshop paper, May 2003, `http://www.itu.int/itudoc/itu-t/workshop/e-health/wcon/s6con002.pdf`.

[19] Wikipedia. Bluetooth, 2007, `http://en.wikipedia.org/wiki/Bluetooth`.

[20] M. Riegel. Internet access by gsm & gprs, November 2001.

[21] R. Hillebrand and T. Wierlemann. Guidelines for the mobile internet, January 2003, `http://mobileinternetguide.org/html/ch01s01s03.html`.

[22] UMTS World. Overview of the universal mobile telecommunication system, 2007, `http://www.umtsworld.com/technology/overview.htm#a1`.

[23] Que Publishing. Comparing 802.11a, b, and g: Channels and interference, 2007, `http://www.quepublishing.com/articles/article.asp?p=413459&rl=1`.

[24] Wikipedia. Ieee 802.11 standard, 2007, `http://nl.wikipedia.org/wiki/IEEE_802.11`.

[25] M.D.J. Cox and R.G. Davison. Concepts, activities and issues of policy-based communications management communications management. *BT Technology Journal*, 17(3):155–169, July 1999.

[26] S. Calo and M. Sloman. Policy-based management of networks and services. *Journal of Network and Systems Management*, 11(3):249–252, September 2003.

[27] N.C. Damianou. *A Policy Framework for Management of Distributed Systems.* PhD thesis, Imperial College of Science, Technology and Medicine; University of London; Department of Computing, February 2002.

[28] A.S. Tanenbaum and M. van Steen. *Distributed systems, principles and paradigms.* Pearson Education Inc., 2 edition, 2007.

[29] H.J. Wang, R.H. Katz, and J. Giese. Policy-enabled handoffs across heterogeneous wireless networks. *Mobile Computing Systems and Applications*, pages 51–60, February 1999.

[30] A. Westerinen, J. Schnizlein, and et al. Terminology for policy-based management – rfc 3198, November 2001.

[31] R. Montanari, E. Lupu, and C. Stefanelli. Policy-based dynamic reconfiguration of mobile-code applications. *Computer*, 37(7):73–80, July 2004.

[32] POLICY 2004 panel (Sloman M.). Policy-based management: the holy grail?, 2004.

[33] Malohat Ibrohimovna Kamilova. *Policy-based handoffs across intermediate multimedia content providers in the wireless internet.* PhD thesis, University of Twente, October 2004.

[34] INTAP. Survey on policy-based networking, 2001.

[35] Wikipedia. Scalability, 2007, `http://en.wikipedia.org/wiki/Scalability`.

[36] L.A. Lumberopoulos. *An adaptive policy based framework for network management.* PhD thesis, University of London, Imperial college London, Department of Computing, October 2004.

[37] M. Cox and R. R. Davison. Concepts, activities and issues of policy-based communications management. *BT Technology Journal*, 17(3):155–169, July 1999.

[38] W. Zhuang, Y.S. Gan, K.J. Loh, and K.C. Chua. Policy-based qos management architecture in an integrated umts and wlan environment. *IEEE Communications Magazine*, November 2003.

[39] N. Damianou, A. Bandara, Sloman M., and E. Lupu. A survey of policy specification approaches, April 2002.

[40] E.C. Lupu and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE TRANSACTiONS ON SOFTWARE ENGINEERING*, 25(6):852–869, November/December 1999.

[41] M.W.A. Steen and J. Derrick. Formalising odp enterprise policies. *Enterprise Distributed Object Computing Conference,Proceedings. Third International*, pages 84–93, 1999.

[42] M. Bearden, S. Harg, and W. Lee. Integrating goal specification in policy-based management. In *proceedings of Workshop on Policies for Distributed Systems and Networks*, January 2001.

[43] Wikipedia. Message exchange pattern, 2007, `http://en.wikipedia.org/wiki/Message_Exchange_Pattern`.

[44] J.L. Hellerstein, Y Diao, S. Parekh, and D.M. Tibury. *Feedback control of computing systems*. New York: John Wiley, 2004.

[45] M Marples and P. Kriens. The open services gateway initiative: an introductory overview. *IEEE Communications Magazine*, pages 110–114, December 2001.

[46] F. le Mouël, N. Ibrahim, Y. Royon, and S. Frénot. Semantic deployment of services in pervasive environments, May 2006. INRIA ARES CITI Lab. INSA Lyon.

[47] Computerbase.de. Osgi and system structure, 2007, `http://www.computerbase.de/lexikon/OSGi`.

[48] The OSGi Alliance. Osgi service platform core specification. `http://osgi.org/osgi_technology/download_specs.asp?section=2#Release4`, 2006 July. Release 4, Version 4.0.1, 2007-02-26.

[49] M. Offermans. Using apache felix: Osgi best practices. `http://cwiki.apache.org/FELIX/presentations.data/best-practices-apachecon-20060628.pdf`, June 2006. 2007-02-26.

[50] R.S. Hall. Jsr 277, 291 and osgi, oh my! - osgi and java modularity. `http://cwiki.apache.org/FELIX/presentations.data/osgi-apachecon-20060628.pdf`, June 2006. 2007-02-26.

[51] P. Campanelli. Status of open source osgi containers, `http://www.pierocampanelli.info/articles/2007/01/22/status-of-opensource-osgi-containers`,2007.