TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Software Science

IT40LT

Olga Andrejeva 112996 IAPB

PERSONALIZED CONTEXT-AWARE USER IDENTIFICATION SYSTEM FOR STATIONARY DEVICES USING FACE RECOGNITION, BASED ON RASPBERRY PI

Bachelor's thesis

Supervisor: Deniss Kumlander
Doctor's degree
Senior researcher

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

IT40LT

Olga Andrejeva 112996 IAPB

PERSONALISEERITUD KONTEKSTI-TEADLIK KASUTAJA IDENTIFITSEERIMISE SÜSTEEM STATSIONAARSETELE SEADMETELE KASUTADES NÄOIDENTIFITSEERIMISE TEHNOLOOGIAT NING RASPBERRY PI-D

bakalaurusetöö

Juhendaja: Deniss Kumlander
Doktorikraad
Vanemteadur

Tallinn 2017

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Olga Andrejeva

21.05.2017

Abstract

The aim of this work is to reduce a human-computer interaction complexity by hiding identification process from the end user. Author sees the key to the problem solution in the pervasive computing technologies and context-awareness.

Today the pervasive computing is a perspective field of IT. The computer is not only a separate machine anymore, but a network of different smart devices, which are able to adapt to the situation and user's needs by acquiring an information from surrounding environment, that is - being context-aware. Such devices tend to be as much invisible as possible, becoming the part of our everyday life.

A thorough research of the related studies was conducted in this paper, devoting special attention to the existing practices and models of handling and representing a contextual information along with face recognition possibilities. Provided survey allowed to compare different approaches and derive appropriate methods and technologies for the solution of the stated problem.

A personalized context-aware user identification system for stationary devices was designed based on the research and the system prototype was built on the Raspberry Pi. Provided solution ensures the availability and compactness of the system, proposed architecture allows to separate system components across different devices. Implemented system helps to hide an identification process from the end user through context information handling skills.

This thesis is written in English and is 74 pages long, including 4 chapters, 19 figures, 2 tables and 10 code samples.

Annotatsioon

Personaliseeritud konteksti-teadlik kasutaja identifitseerimise süsteem statsionaarsetele seadmetele kasutades näoidentifitseerimise tehnoloogiat ning Raspberry Pi-d

Antud lõputöö eesmärk on vähendada inimese-masina suhtluse keerukust peites identifitseerimise protsessi lõppkasutaja eest. Autor näeb probleemi lahendust ulatuslikus IT tehnoloogiate kasutamises ning konteksti-teadlikkuses.

Tänapäeval on ulatuslikud IT tehnoloogiad perspektiivne valdkond IT maailmas. Arvuti ei ole enam ainult eraldi masin, vaid terve võrk, mis ühendab erinevad targad seadmed. Need seadmed on omakorda võimelised kohanema vastavalt olukorrale ja kasutaja vajadustele, omandades informatsiooni ümbritsevast keskkonnast, olles seeläbi konteksti-teadlikud. Selliseid seadmed püütakse teha võimalikult märkamatuteks ja nad on muutumas meie igapäevase elu osaks.

Antud töö raames on läbi viidud põhjalik uuring, mille käigus on pööratud erilist tähelepanu olemasolevatele praktikatele ning mudelitele, mis on seotud konteksti-teadliku informatsiooni töötlemise ja esitlemisega ning näotuvastuse tehnoloogiate kasutamise võimalustega. Läbi viidud uuring võimaldas võrrelda erinevaid lähenemisi ning tuletada sobilikud meetodid ja tehnoloogiad püstitatud probleemide lahendamiseks.

Personaliseeritud konteksti-teadlik kasutaja identifitseerimise süsteem statsionaarsetele seadmetele on projekteeritud tuginedes läbiviidud uuringule ja süsteemi prototüüp on ehitatud kasutades Raspberry Pi-d. Pakutud lahendus tagab süsteemi kättesaadavuse ja kompaktsuse, pakutud arhitektuur võimaldab aga jagada süsteemi komponente erinevate seadmete vahel. Loodud süsteem aitab peita lõppkasutaja eest keerukaid identifitseerimise protsessi detaile ja kasutab tulemuse saavutamiseks kontekstipõhise informatsiooni käsitlemise oskusi.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 74 leheküljel, 4 peatükki, 19 joonist, 2 tabelit, 10 koodinäidet.

List of abbreviations and terms

AmI	*Ambient Intelligence*, a term to describe a surrounding environment saturated by electronic devices which react and response to human presence.
ARM	*Advanced (initially – Acorn) RISC Machine*, a family of licensed 32-bit and 64-bit architecture of microprocessor cores developed by ARM Limited company.
BSD license	*Berkeley Software Distribution license*, a licence agreement, initially applied to the distribution of the UNIX-like BSD operating systems.
CPU	*Central processing unit*, an electronic circuitry which performs computations of the machine instructions (i.e. code) of a computer program, located within a computer.
DBMS	*Database Management System*, system software for creating and managing databases.
DPI	*Dots per inch*, the measure of the picture resolution (clarity) represented by number of dots that can fit the one inch long line.
HCI	*Human-computer interaction*, a field of researches covering the studies of how people interact with computers and related developing processes, which aim to make computers "user-friendly".
IR	*Infrared (radiation)*, invisible electromagnetic radiation, which wavelengths are longer than wavelengths of visible light.
IT	*Information Technology*, the study or use of systems for handling (retrieving, storing, sending, processing) data or information, subset of information and communication technology.
JDBC	*Java Database Connectivity*, java application programming interface defining the ways of accessing the database.
LAN	*Local area network*, a computer network restricted by limited area, which manages interconnections locally.
LBP	*Local Binary Pattern*, visual descriptor type used for classification in computer vision by comparing central pixel to its neighbours represented in binary form as 8-bit binary code describing pixels neighbourhood.

LBPH	*Local Binary Pattern Histogram*, graphical representation of the distribution of numerical data used to compute 256-dimensional LBP feature vector.
OODBMS	*Object Oriented Database Management System*, system software for creating and managing object oriented databases.
ORDBMS	*Object Relational Database Management System*, system software for creating and managing object relational databases.
OS	*Operating system*, a software system designed for managing hardware and software resources and responsible for user-computer interaction handling.
PCA	*Principal Component Analysis*, the one of the main methods of decreasing a number of data dimensions with minimum data loss.
PL/pgSQL	*Procedural Language/PostGres Structured Query Language*, procedural extension of the SQL language used for the PostgreSQL database system.
RDF	*Resource Description Framework*, a model developed by World Wide Web Consortium for data (especially metadata) representation.
SQL	*Structured Query Language*, a standard language for accessing and manipulating relational databases.
UI	*User Interface*, an interface that provides an exchange of information between humans and machines.
XML	*eXtensible Markup Language*, a markup language used for defining a set of rules encoded in the document that can be read by computer as well as by human.

Table of contents

List of figures

List of tables

List of code samples

1 Introduction

Today a lot of scientists see the future of computer technology in many different devices, which are capable of interacting with each other, understanding what is going on in the surrounding environment and reacting on time to the occurring changes, means being context-aware. Hence, context and context awareness as the key components in pervasive computing [9] are perspective IT fields.

The aim of the current work is to try to minimize amount of interactions between human and computer by designing an abstract personalized stationary context-aware system. A conducted research of existing related studies in the given field should help to pick out the most appropriate practices, models and technologies relying on the obtained information. Since we are dealing with the model of a system which should provide personalized data output for each of its end users respectively, a face recognition is used as an identification method.

In order to demonstrate an efficiency of the proposed model a simple personalized context-aware user identification system was designed on the developed model basis and working prototype was developed. For the sake of simplicity the prototype is considered to be a home system: that means that it is designed to handle finite number of users, which allows to avoid problems related to system overload, but, at the same time, gives an opportunity to test the system in a real life conditions.

1.1 A problem statement

Human-computer interaction is the one of the general topics of the pervasive computing field. A whole interaction process should be as simple and intuitive as possible and each user should be threatened by the system differently, depending on the situation in general.

Because of the need of pervasive technologies to stay invisible as much as possible, a problem of minimizing human-computer interaction arises. Thus HCI comes along with a need of context interpretation mechanisms, which could help to solve some tasks using acquired context information and without a need to disturb the user by requiring from him some additional input or actions.

The aim of this work is to try to reduce the amount of actions that user needs to take to get a personalized response from the abstract stationary system. A two prior steps of a user-system interaction are being involved:

1. **User identification**

   User identification process should be straightforward, performed insensibly and without querying input data directly from the user. At the same time it should be precise enough to minimize the percent of mistakes. Thus, because of its tricky nature, the user recognition problem is considered to be the general problem of this work.

2. **Personalized data output depending on identified user**

   There is a grooving tendency to create personalized systems and personalization problem is taking one of the general places in the pervasive computing field. Personalization should not be performed only by using user settings, but also using gathered and analysed context information, related to the active user. The way of gathering, processing and storing such kind of information is the second general problem of this work.

So the main question is: how to design a stationary system which could detect users with the minimum effort and provide them personalized data output?

1.2 Methodology

Research & analyse

During the research process a literature review was conducted and the outcome was presented to describe topics related to this paper. Based on the gathered information and acquired results, the most appropriate methods and best practices were selected to provide an effective and actual solution of the problems which are defined in the scope of this work.

System design & prototype

Gathered information was used to design a context-aware stationary system. The system corresponds to pervasive computing principles, provides a straightforward and robust user identification mechanism and acquires, analyses and stores related context information to provide personalized data output to the user.

User identification problem is resolved using the face recognition mechanism.

The prototype is built to demonstrate implementation of the provided system design.

1.3 Overview of the following chapters

Chapter 2 of this work covers the theoretical background and the outcome of the conducted research of the related studies and literature, which is essential for finding the best solution of the stated problem. During the sections 1 and 2 of the chapter such topics as pervasive computing, context, context-awareness and context information are reviewed. The 3rd section of the chapter is focused on the contextual data, and particularly on such topics as its processing and representation.

Chapter 3 includes an information about personalized context-aware user identification system, which is intended to reduce the complexity of human-computer interaction process by using the outcome of the 2nd chapter. Sections 1 – 3 of the chapter cover a system structure and design, including requirements and used technologies. Section 3 is focused on the face recognition problem solution, explores different ways of implementation and provides derived results and conclusions. Section 4 describes technical solutions used for system construction. Sections 5 provides information about

required environment preparation, and Section 6 covers related system restrictions and possible improvements.

2 Theoretical background

2.1 Introduction into pervasive computing

Pervasive computing is a very popular field of researches, discussions and disputes in the IT field nowadays [45]. Scientists (Weiser [43], Friedemann Mattern [24] and Peter Sturn [23], Saha, D., & Mukherjee [35] and many others) see the future of computer technology no as multiple single devices, but as a unite network of multiple computers, which are interacting and cooperating with each other and are capable of adapting to the surrounding environment: they will be able not only to define own location or, for example, situation around them, but also act correspondingly. Unlike a virtual reality, which tears off a human of what is happening around, ubiquitous computers are expected to exist with people imperceptibly and not to be an obstacle to personal interactions and real world perception. Billions of tiny mobile processors will be built into many various devices from our surroundings, complementing familiar world. *"Machines that fit the human environment, instead of forcing humans to enter theirs, will make using a computer as refreshing as taking a walk in the woods."* [43].

2.1.1 Pervasive computing vs ubiquitous computing

Today there are two terms, which have very similar meaning and are being used as complementary definitions a lot [35]: these are pervasive and ubiquitous computing. "Ubiquitous computing" or „ubicomp" was firstly introduced by Weiser in 1991 [43]. *"Ubiquitous computing" in this context does not just mean computers that can be carried to the beach, jungle or airport"* – emphasized in the research. In the opinion of Weiser such computers should be connected by ubiquitous network, should know their location, be invisible, on the background, pose no barrier to personal interactions and provide private data safety.

Weiser thought that computers should not create virtual reality, but be an addition to our real world, enriching, enhancing it and cooperating with it. Most of these computers will stay invisible for humans while cooperating with the environment. That is being called "embodied virtuality". "Pervasive computing" term has the same origin. But if in the case of ubicomp the need to provide unite computing environment for the end user is

emphasized, regardless of user's location, then pervasive computing in its turn more environment context oriented and expects dynamic interaction with it [45].

The differences between these two terms were also noticed by Professor Friedemann Mattern (Swiss Federal Institute of Technology in Zurich), who mentioned in his paper [24] that Weiser's vision of the "ubiquitous computing" was "in a more academic and idealistic sense as an unobtrusive, human-centric technology vision that will not be realized for many years". He noticed, that the term "pervasive computing" has a "slightly different slant": despite that "pervasive computing" is also related to pervasive and omnipresent processing of information, its main goal – use these kinds of information processing in the electronic commerce and Web-based business processes - is different because it is concentrated on the nearest future.

Thereby, although there are several conceptual differences between terms pervasive and ubiquitous computing [22], yet in fact both are very close to each other. Because there is no need to strictly stick to the details of any of these terms to resolve problems stated in this work, then hereinafter terms ubicomp and pervasive computing will be used as the synonyms in the scope of the current work.

2.1.2 Required technology

Weiser in his early research made an assumption that there are three main parts of technology requirements for ubiquitous computing [43]:

1. **Cheap, low-power computers that include equally convenient displays.**

   In the far 1991 Weiser predicted (based on that time trends), that this requirement will easily be met, and he was right. Computers nowadays are not rare anymore: their number and diversity is constantly increasing, and prices become more and more affordable. As an example: microcomputer Raspberry Pi, which will be used later in the work, has a price less than 35$ (information as of December 2016, latest model - Raspberry Pi 3 Model B) [33].

2. **A network that ties them all together.**

   The last century scientists were facing such network problems as coverage, connection speed and single-threading [38], when ubiquitous computing only

began its evolution. Of course, a lot has changed since then, and the network problem is not as urgent anymore.

The access to the networks (both wired and wireless) with high data transmission rates became progressively cheaper (as predicted by Weiser in 1991) and now is available and widespread in the modern developed countries. Network coverage area will only grow in the future, the progression, mentioned above, will remain.

However there are still present such problems as limited coverage of wireless networks, restricted number of network connections, need to explicitly recognize the concept of devices moving in physical space and so on. New models and protocols are required.

3. **Software systems implementing ubiquitous applications.**

   Although software implementation is moving towards ubiquitous computing ideas, unfortunately there are still a lot of fixed base systems, which are bound to fixed settings and requirements and are not ready to adopt to changing environment (like location, actions and so on). This statement is applicable, for example, for modern operating systems (Windows, Linux, etc.). However mobile devices are one step forward in that direction: for example, we have smart watches and mobile phones that already have a lot of opportunities to adapt to changing environment, user needs and cooperate with each other.

   Nevertheless the field of research for that topic is still enormously huge because of the number of difficulties related to implementing ubiquitous applications.

2.2 What is context?

The global explanation would be that context is the environment in which some entity exists with some level of abstraction applied to that environment. But how to define that term according to the reality of pervasive computing? The definition must be precise, clear and rich enough, to provide a base for developing models for context-aware systems. Researchers all over the world try to give such definition more than for two decades, but still there is no “panacea” invented.

Notion of context

As suggested by Paul Dourish [8], the notion of context has a dual origin:

1. Technical notion offers ways of a) conceptualising human action and the relationship between that action and b) computational systems to support it;

2. Social science notion draws analytic attention to certain aspects of social settings.

In this thesis when we talk about context it is a context in ubiquitous computing if other is not said.

Notions explained above lead us to different definitions of the context.

Definition of the context

Since the context became an essential part of the pervasive computing and up to this day there is no unified definition of context [3]. Each year researches from different IT fields develop new definitions, which are the most suitable from their point of view and reflect the knowledge and experience of their authors and realities of the time in which they exist.

Some main context definitions are given below in chronological order (from 1994 to 2007).

1994: Bill N. Schilit and Marvin M. Theimer [38] regard context as the location and characteristics of objects - devices and people who use them - within some region and change of those objects over time. Later in 1994 Bill Schilit, Norman Adams and Roy Want refer to context as including three important aspects: location of an object, related objects and nearby resources [37].

1997: By definition of the Brown, Bowey and Chen [4] the aspects of the user's context are location, temperature, season, ambient people, time of day, season of the year, etc.

1998: Ryan and Morse [29] define context in their work as time, user's identity, location and environment.

2001: Anind K. Dey analyses in his work [7] definitions of "context" since the 1994 and proposes his own vision of that term, which is not "too specific" as many other definitions and because of that is applicable to any type of pervasive system: "*Context is any*

information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves".
This definition was then widely used by other researches and most widely referenced in their works ([40], [9], [13] and others).

2002: Chen and Kotz state, that *"Context is the set of environmental states and settings that either determines an application's behaviour or in which an application event occurs and is interesting to the user"* [6].

2004: Very interesting view of context was introduced by Ling Feng [12]: he assumes, that context should be viewed as an *n*-dimensional space, which is built up of *n* contextual attributes. Each context dimension should be represented by one contextual attribute, which describes one perspective of context.

2007: In the [9] research the context is defined as operational term, which is being considered as entity interpretation dependent at a particular moment and location rather than inherent characteristics of the entity.

As you can see from listed above, although there are differences between definitions given, they are still have similar and quite certain foundation:

1. Context is an information.
2. Context must have a subject.
3. Context must characterize a subject and corresponding subject changes.
4. Context must characterize situation and corresponding situation changes.

Today we can claim that these are main context characteristics. They form the basis which will be used in this work when designing and building context-aware system.

Context-awareness

"Context-awareness" means that system is capable to determine, analyse and use related context (the context, in which that system runs). Such system should adapt dynamically

to the changing environment over time [37]. Bill N. Schilit was the first to introduce "context-aware" term in his work in 1994 [38].

Kostas assumes, that system is context-aware if context is being used to provide relevant information and/or services to the user [40]. Relevancy in this case depends on the user's task.

2.2.1 Context structure

Context components

According to the section 2.2 a two main components of a context can be distinguished:

1. **Entity**

   A central object, relatively to which a notion of a context is being considered. Each entity has its own abstraction level.

2. **Related situational information**

   That can be any information, which is related to the environment that is surrounding the entity: e.g. physical location (regarding to the other objects or certain reference point), state of the surrounding objects, time, humidity etc.

   From the variety of all these information types four main types can be distinguished: **time, location, an action** and **the object, that performs the action**. Usually just these types characterize the context in the majority of cases.

Context types based on context properties

Researches in their works [15], [12], [13] consider and characterize context differently, but one way or another there is tendency while working with some specific problem to bound it to some certain context type. Context type in its turn can be handled as characteristic of the some context property. Based on researches three main context properties can be distinguished: **changeability**, **centrism**, **sensibility**. Each property has two context types correspondingly. Since context information can be viewed from the side of different properties, hence it can belong to many context types at the same time

(one for each property). For example, context can be dynamic, sensed and user-centric simultaneously.

Main context types and properties are schematically represented below (see Figure 1).

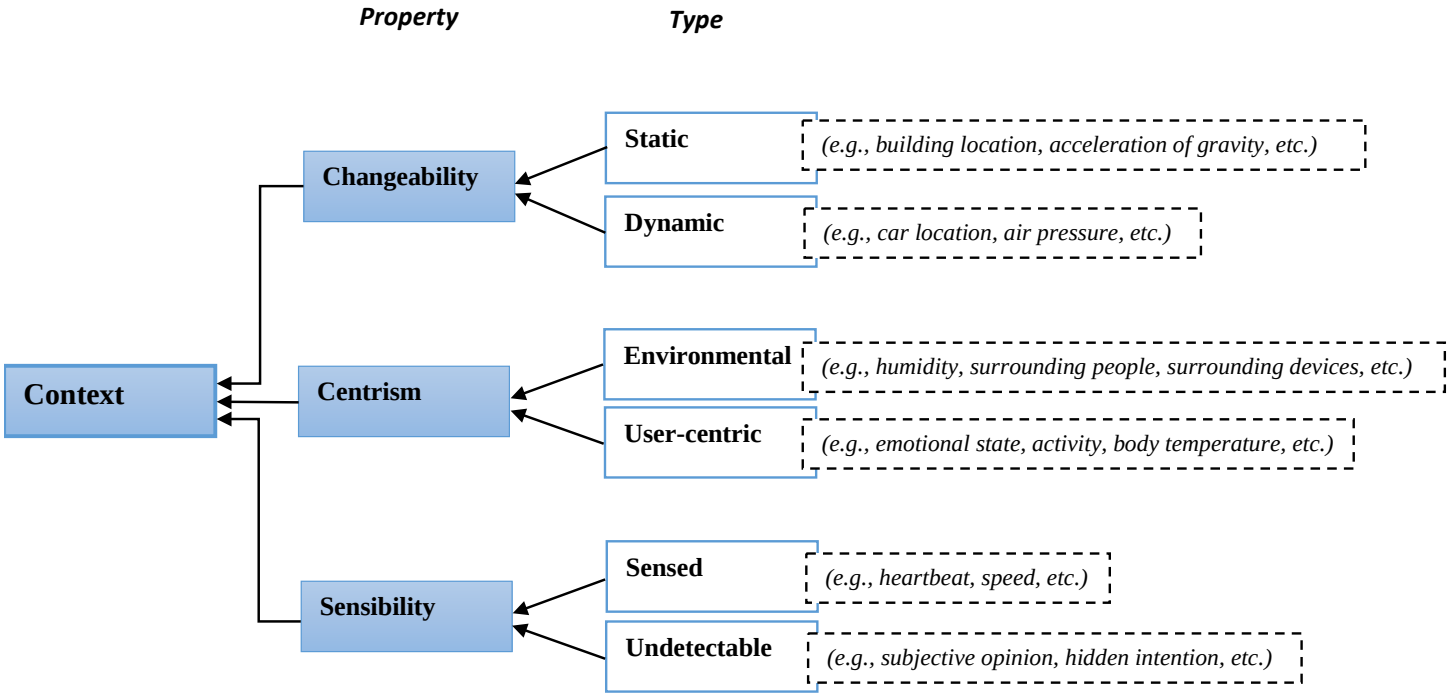


Figure 1. Schematic representation of a context types and properties. Context properties represented in solid blue boxes, types of context properties represented in white boxes with blue border. In dashed boxes some examples are provided accordingly to the each property type.

Changeability

Static context is represented by information that is barely changed (like person's name, sex or favourite colour) or does not change at all (like person's mother and father). Such type of information should be acquired directly from the user [15], which is the most reasonable, easy and cheap way of its acquisition.

Dynamic context is represented by continuously changing information (like weather, time and illumination). Such type of information should be acquired continuously, not from user directly, but using sensors of different types [15], from visible and hidden from the end user as well. Besides, this information should be constantly processed (i.e. needs

to be saved and interpreted appropriately in the scope of the specific system, and in some cases excess of such information should be periodically deleted, depending on acquisition frequency, information type and the system itself).

Centrism

User-centric context is all user-related context information. It could be acquired from sensors, user input, service providers etc. As proposed in [12], such type of context could be, for example, user psychological state, background (interest, working area), emotional state and dynamic behaviour.

Environmental context is all environment-related information. In the [12] environmental context is represented not only by information of physical environment, but also categorized as social and computational environment. In the work mentioned above context information of a given type is proposed to be acquired from service providers, user's activity and also from external sensors.

Sensibility

Sensed context information is being acquired through sensors from real world. Philip Gray and Daniel Salber give definition of sensed context in their work [13]: *„Properties that characterize a phenomenon, are sensed and that are potentially relevant to the tasks supported by an application and/or the means by which those tasks are performed"*.

In other words it is such information, which can be acquired from the surrounding world without using familiar external input devices, without engaging user of a system directly. Sensed context is obligatorily derivable from sensors.

Undetectable context can't be acquired from surrounding environment. Information of this type, though, can also be related to the context (personal opinion, for example), so it should be used in some systems along with other information types. This type of data is usually being received directly from user through some input device. For example, keyboard or touch screen.

2.3 Handling contextual data

2.3.1 Context data models

Overview of existing models

In order to give to the context-aware applications an opportunity to interact with context information, it should be represented in some specific way, using specific pattern, understandable to the initial system.

Initially different models were created by researches depending on properties and needs of some specific system. So designed and implemented in an ad-hoc manner, such models have limited field of use. But later more general models started to gain popularity [15].

Chen and Kotz in their "A Survey of Context-Aware Mobile Computing Research" distinguished five main data structures of existing context-aware applications, which are used for exchanging and expressing context information: key-value pairs, tagged encoding, object-oriented model, logic-based model and layered model [6]. Nine years later proposed models were analysed by Christian Hoareau and Ichiro Satoh [15], and their specifications were adjusted according to the realities of that time. Also two new models were introduced additionally to the existing ones: ontology and situation logic models.

1. Key-value model

In case of the key-value data representation, the key is represented by environmental variable and value is represented by actual context data of the key variable. This simple and at the same time powerful model also allows pattern-matching queries.

Schilit [37] used key-value model in order to manage location information. But because of the lack of expressiveness more flexible models have been used instead. Nevertheless, key-value pair model is appropriate for large-scale distributed systems (like cloud computing).

2. Markup model

Markup or «tagged encoding» model consists of tags and corresponding fields, i.e. content. Tags also can contain attributes, and fields can contain other tags and related

content, recursively. Markup model has a hierarchical data structure and can be represented by XML, for example.

But in the [15] is pointed out, that because of the static nature of XML and RDF notations, they are not suitable for maintaining dynamically evolving context information, which changes accordingly to the real world shifts.

3. Object-oriented model

This model is similar with object-oriented programming paradigm. It has two central elements: object is considered as the first central element of the model, and relationships between objects as the second one. Contextual information is represented and stored as object states. States can be queried and modified using functions, provided by the object itself.

Object-oriented model provides flexible structuring capabilities, such as inheritance, reusability and encapsulation.

4. Logic-based model

This model represents context information as a set of facts, expressions and rules in a rule-based system. New rules can be defined and data also can be queried.

Rule-based programming is well-suited for developing context-aware applications because of intuitiveness. There is an implementation of context-aware system built using logic-based model in Prolog.

5. Layered model

Chen and Kotz pointed out, that there also exists project that represents context data in a layered structure [6]. First layer is a contextual data acquired from sensors, raw and time-stamped. The second layer operates with data from the first layer, processes it and outputs corresponding value or values, if more than one are defined. Such processing is sensor-dependent, but more than one processing can be performed on one particular sensor. Third layer is the last one. It is responsible for deriving context from all received values across all sensors.

In layered model context is represented by a set of vectors. Vectors are two-dimensional and each consists of value and probability number. The final, highest application layer operates with gained vectors using primitive functions.

6. Ontology model

Ontology-based models have vocabulary for situational predicates. That allows share data representation across different systems or even over the Web. Also is possible to enrich sensor predicates with vocabularies and additional semantics, which means that context attributes specified by ontology can be represented by such predicates.

Regardless of all advantages of ontology-based model, it also has disadvantages of data representation.

7. Situation Logic model

Situation logic model came from rule-based programming principles. It represents entire situations by integration between LogicCAP as the programming layer and context infrastructures as providing sensed contextual information. So it is more than rule-based triggers for context-aware actions.

Summary

These approaches to modelling context in ubiquitous computing were accordingly reviewed in a context modelling survey [41]. In the research some optimal requirements were derived on context model, considering the optimal usage scenario in a ubiquitous computing environment. These requirements are: distributed composition, partial validation, richness and quality of information, incompleteness and ambiguity, level of formality and applicability to existing environments. Accordingly to the provided surveys [15], [41] of all referenced models ontology based model, object-oriented model and markup model meet most proposed requirements.

2.3.2 Three steps of context processing

As suggested in the [15], context-aware systems must do three main things in general: acquire context information, process it and then provide a number of services to the user, according to the data obtained during three steps mentioned before.

Let call those three steps "**steps of context processing**". They strictly follow each other, which can be represented schematically in the following way (see Figure 2):

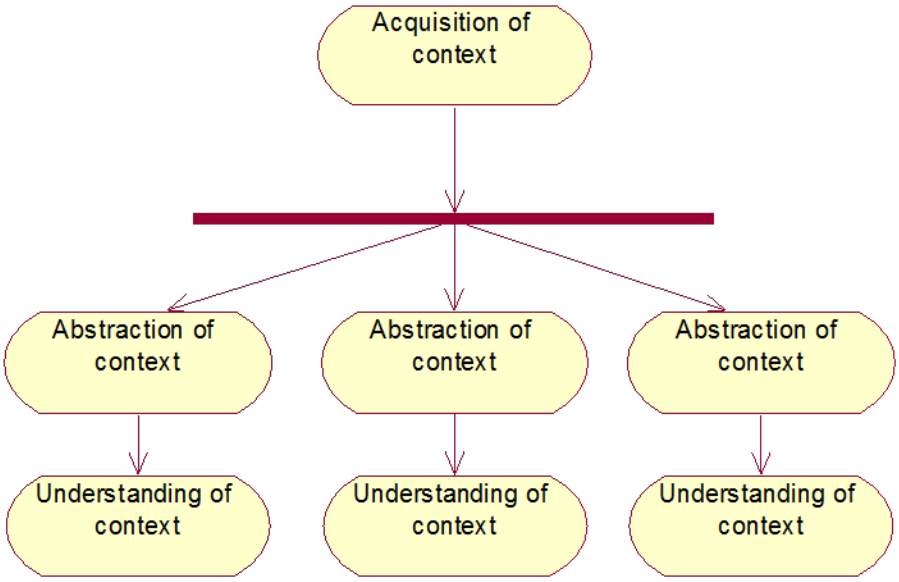


Figure 2. Activity diagram showing three steps of context processing. Sequence of 2nd and 3rd steps may be performed multiple number of times in parallel.

Speaking of context regarding context processing steps the context information is being meant here.

Each step of context processing is overviewed in details below.

1. Acquisition of context

Context information is extracted from surrounding environment with the help of various sensors, which can be completely different, be fully independent from the system itself and also remote, connected to the context-aware system by network. In addition to reading data from the sensors, information can be also acquired directly from a user of the system. Thus two main types of context information acquisition can be distinguished:

1. **Independent sensors** work most often with dynamic context (see Section 2.2.1), regardless if it is environmental or user-centric context. Data can be captured in a real time (i.e. temperature sensor, glucose sensor, IR sensor) as well as through specific time intervals or in case of occurrence of some triggering event (motion sensor, for example).

2. **External input devices** also can be involved in the context information acquisition process. In this case the information comes directly from the user. Usually such devices are targeted to gather static user-centric information and help to obtain such data that is impossible (or almost impossible) to acquire from the surrounding environment. For example, data that is related to the how the system is being used by the client and analyse of which allows to bring out preferences of specific user. Further obtained preferences may be complemented by the data from external sensors and so more complete picture of what is going on can be derived.

Both ways of acquisition a context information are complementary.

2. Abstraction of context

Context abstraction means bringing it to some general, common form and getting rid of excess details. Given step is needed, because depending on the specific situations one or another sensor may have an influence on the content of an acquired information.

Same information can be represented by different abstractions, which can or cannot be connected between each other. Considering the fact that each system may analyse and understand context differently, in its own way, obtained information should be converted to the most universal form eventually.

3. Understanding of context

At the given step obtained context information is being transformed to the form (data structure, algorithm etc.) that system can understand and work with. Final form of acquired data depends on the aims of the specific context-aware system and on the services, to which that data must be presented afterwards in the scope of the given system.

Same data may be represented and/or understood differently in the different systems (e.g. represented by different data structures), but these differences should not influence on the obtained information: data may (and must) be analysed and queried by the systems, but sources must not be changed at that step of context processing. Otherwise interpretation may be wrong, full of errors, and results of identical queries, performed on the same data subset, may differ with time (that should not happen).

2.3.3 Database model for storing context

Today there is a lot of different database management systems, but not all of them are suitable for dealing with context. A recent studies show that in case of context-aware systems, using object relational database along with ontology is a most optimal way for storing context information [42].

Object relational database is based on two other models:

1. **Relational model** allows complex queries while handles simple data types. RDBMS offers physical and logical data independence, i.e. physical structure of data can be changed without disturbing the application and conceptual changes are invulnerable to the applications program. High level query language is separated from the low level implementation details.

2. **Object-oriented model** process complex data types, but queries are also complex and hard to resolve. That said OODBMS has a great possibilities for modelling real world objects.

Due to the fact that **ORDBMS** came from combining those two models, it unites both declarative and modelling power, which is highly important when dealing with context information. It allows fast complex data handling along with fast complex queries processing, what makes it most complete than relational or object oriented models.

Also object-relational database proves to be the most effective way for handling spatial data, i.e. data representing the objects size, location, etc. [16].

2.3.4 Context-aware query processing

Context-aware query is defined by researches as a query, which result aside from database of evaluation depends also on context under which the query is issued [12] at the time of

query submission [40]. Altogether that means that the result of the same query may differ: query may produce different output being called under different context. On the other hand, execution results of two non-context aware queries issued on the same database in the same state are identical.

So context-aware query is different from traditional non-context aware query, and processing of such queries must differ as well. The idea of context-aware query is that it should be viewed as parameterized query with two parameters – **database** and **context** – opposite to traditional context-aware query that depends only on the database [13].

Context preferences

Each user of a system has his own context preferences. Preferences may be marked by user himself (e.g. user can estimate which cinema he prefers), or may be computed by analysing statistical data acquired by the system (e.g. that user searches for cinema tickets most often between 8 and 9 AM and between 6 and 7 PM). A numerical score between 0 and 1 is provided for representing those preferences, as suggested in [40], where 1 indicates extreme interest and 0 – no interest at all. According to the [40] research, preferences can be divided into two types:

1. **Basic preference** contains single context parameter, set of non-context parameters and degree of interest parameter:

   1 x [context parameter] + n x [non-context parameters] + 1 x [degree of interest]

   Basic preferences are stored in the system in hypercubes or cubes, accordingly to the OLAP paradigm.

2. **Aggregate preference** contains combination of context parameters, non-context parameters and degree of interest parameter:

   n x [context parameter] + n x [non-context parameters] + 1 x [degree of interest]

   Storing aggregate preferences in the system is usually not time and space efficient. So only the last computed values of the aggregate preferences could be stored instead.

In case of non-context aware query preferences are ignored.

Steps of context-aware query processing

According to the [12] there are three steps of context-aware query processing:

1. **Query pre-processing**

   First of all, the query is being refined: various restrictions and refinements related to the context information are being applied to the query depending on which information type it contains. After all refinements are made, specific values are being applied to the query parameters (i.e. context binding) and only after that complete query may be executed in the database.

2. **Query execution**

   Query processing in DBMS.

3. **Query post-processing**

   Obtained query result is being sorted according to the user related context information, being adjusted according to the system requirements and only then result is being delivered to the end user.

3 Personalized context-aware user identification system

In this section the user identification system, which allows to personalize the set of provided services to the specific user based on user preferences, is proposed.

The main goals of the system are a) user identification, b) acquiring information about user's preferences and c) providing different compatible services according to the gained information. While gathering information system should query a possible minimum of data from the user and do it as rarely as possible (ideally only once in each case), acquiring most of the needed information from the environment by analysing existing data.

The principle of minimal interaction that is mentioned above is related equally to the all main system tasks:

1. **User identification.** For this task a face recognition mechanism is being used, which allows the system to understand who is interacting with it, without requesting any extra actions from the user (like login or password input, etc.).

2. **Acquisition of information.** The main information is acquired from sensors (camera, motion sensor) and from analysing user's actions (like "User A mostly uses service B from five to nine PM"). System may ask the user to provide some data input or to react at the one or another system event somehow, but ideally that should not happen more than one time for each specific situation (an exception is when a user interact with the system intentionally, including, for example, change of personal data, manual setup and so on).

3. **Personalized output**. A results presentation (providing services, for example) should be performed mostly imperceptibly for the user and user's personal preferences should be also considered. System should act expectedly, accordingly to the stated requirements, user should not feel frustrated during interaction with the system.

Currently the system is intended for a stationary use, but in the future could be extended and complemented with opportunities of dynamical usage.

3.1 Requirements

3.1.1 Functional requirements

General

1) System must support both physical and logical data independence principles. Which means that changes in the physical structure of data and conceptual changes must not disturb the application [3].

Idle time

1) System should display impersonalized main screen which should contain general context information (like date, time and weather).

2) Motion detector should be working continuously until a face is detected.

   a) When the face is detected by the system in the received video input, motion detection stops and user identification process starts.

   b) When there is no face detected by the system in the received video input during 10 seconds and motion detection is not working, then face detection stops and motion detection should start working again. *(Interval of 10 seconds is considered to be an optimal, because this is sufficient amount of time to eliminate the possibility that user is still there and for some reason just stands still in front of the device.)*

User identification

1) System should start user identification process on motion detected.

   a) On motion detected system should launch video capturing.

   b) System should define people's face(s) in the captured video.

   c) When face(s) is (are) found, system should try to recognize the user.

      i) If user recognition was successful:

(1) If the person used the system before and related application user is registered in the database, a personalized system screen should appear.

(2) If this is the first time when the person interacts with the system and/or related application user is not registered in the database, the registration screen should appear. *(This happens when the dataset of user pictures was added to the file system, so the face recognition outcome is positive, but corresponding user has not been added to the application database yet.)*

ii) If related dataset of person's pictures is absent in the file system, then such user is not recognized and the system will not response.

Personalized screen (Identified user's point of view)

1) Personalized screen should appear instead of a general screen to the user whose face recognition process was successful.

   a) Personalized greeting should appear.

      i) Last time when user was recognized should be shown.

   b) Personalized module selection should appear.

      i) Mostly selected modules should appear first and be distinguished.

      ii) Mostly preferred modules should appear first and be distinguished.

      iii) Mostly viewed modules depending on the current context information should appear first and be distinguished. Should depend on:

         (1) Time of the day.

         (2) Day of the week.

2) User should be able to select module display criteria.

3) User should be able to set personal preference for each proposed module.

Logout

1) If logged in user's face is not recognized longer than 10 seconds then system performs logout.

2) After logout system should return to idle state.

Registration of a new user

1) When registration process is launched, registration screen should appear. Registration screen should contain:

   a) First name input.

   b) Last name input.

   c) Accept / Decline buttons.

2) When "Accept" button is pressed a new user is being added to the system.

3) When ""Decline" button is pressed system should return to idle state.

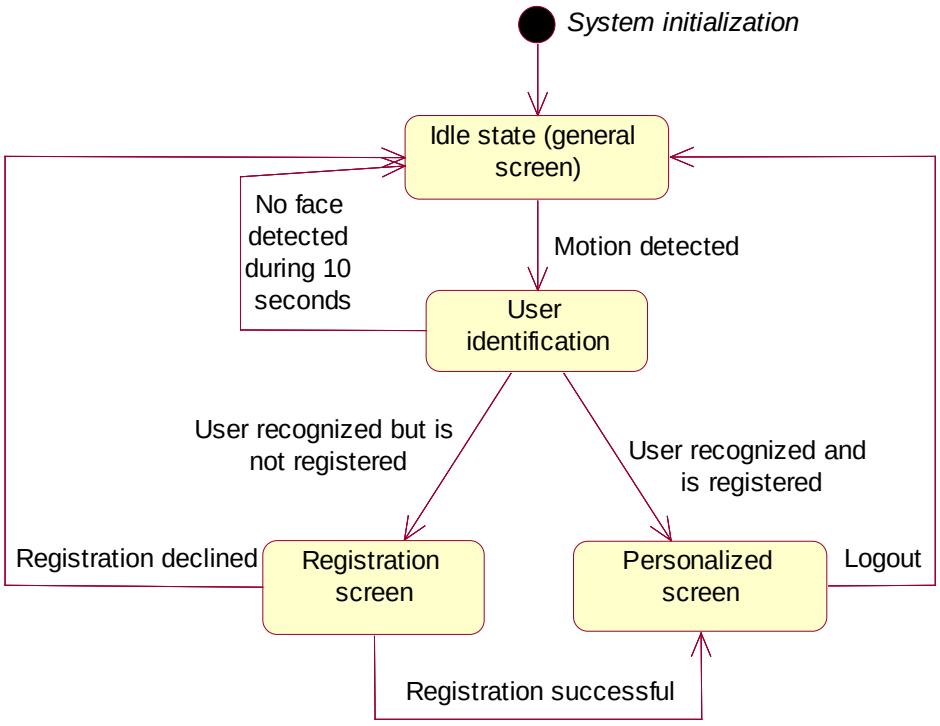


Figure 3. System statechart diagram – main lifecycle.

3.1.2 Non-functional requirements

1) Weather data on general screen should be updated each five minutes.

2) User's face is considered recognized if prediction confidence is a below a confidence threshold of 150 (see Section 3.3.2 for more related info on confidence threshold).

3) System should be able to handle one user at a time.

    a) If there are two or more people interacting with the idle system at the same time, then first identified user is considered to be the one using the system.

4) System should be able to handle up to 50 registered users.

5) Video capturing during face recognition process should be performed with maximum delay of 3 seconds.

6) System response should not take more than 3 seconds.

3.2 Design

This section provides a description of system structure: its layers, components and interaction between them.

3.2.1 General architecture

Software layers

Personalized user identification system contains three software layers: application layer, service layer and data layer (see Figure 4).

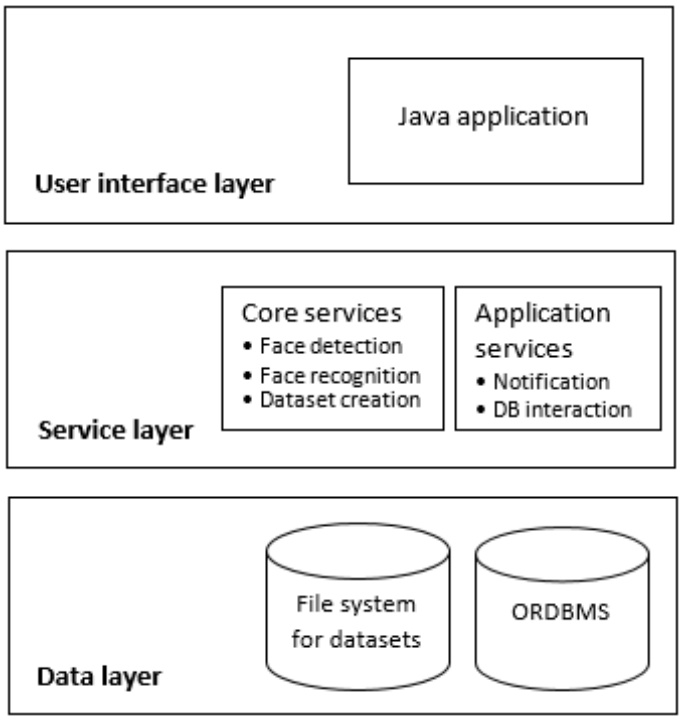


Figure 4. System software layers.

Data layer is responsible for data storage and includes 1) database, which is used by both services and application, and 2) datasets stored in the file system, which are represented by set of files (pictures) and folders containing them and used by core services. Each dataset contains number of pictures related to the specific user.

Service layer includes two main parts: the first par consist of the core services, which are responsible for the face detection and recognition process along with creating new datasets. Those services are running continuously, providing required functionality during all uptime. The second part is represented by services related to the user application, e.g. services responsible for handling database notifications.

User interface layer consists of a full-screen application which is running independently and responds correspondingly to the notifications from the service layer.

Communication between system components is represented as the component diagram at the Figure 5, where names of the main technologies are also provided.

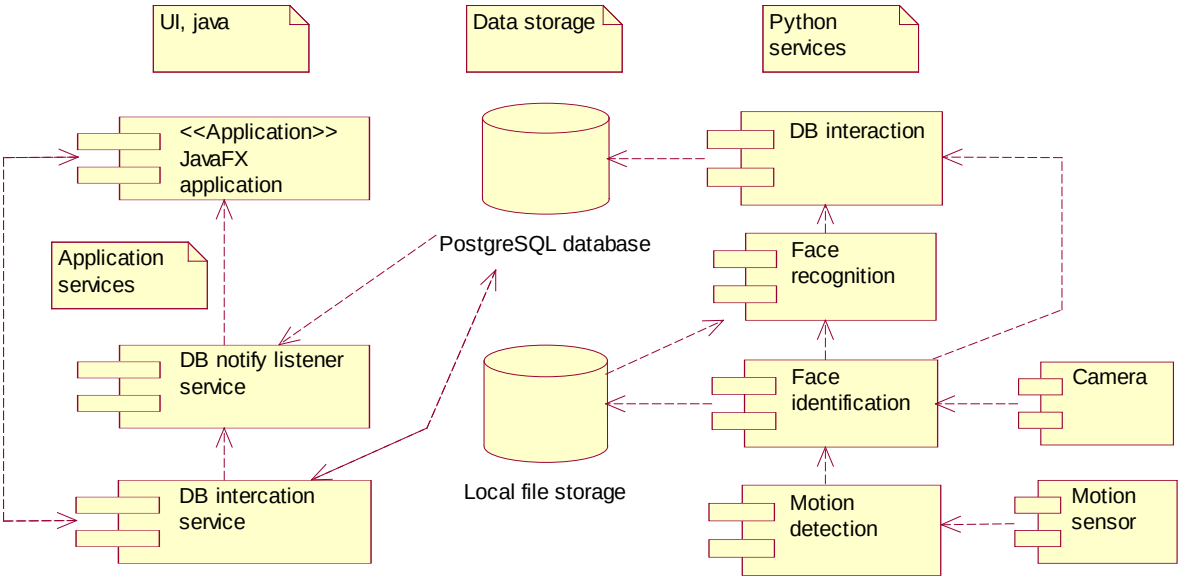


Figure 5. System component diagram shows system general composition which displays all physical system parts, including sensors and communication between them (represented by dashed arrows).

3.2.2 Core services

Core services is a collective name for face detection service, face recognition service and dataset creation service. All core services are written in Python and involved in the loop, which is regulated by core python script. Figure 6 shows the main activity diagram of the core service running in the normal mode.

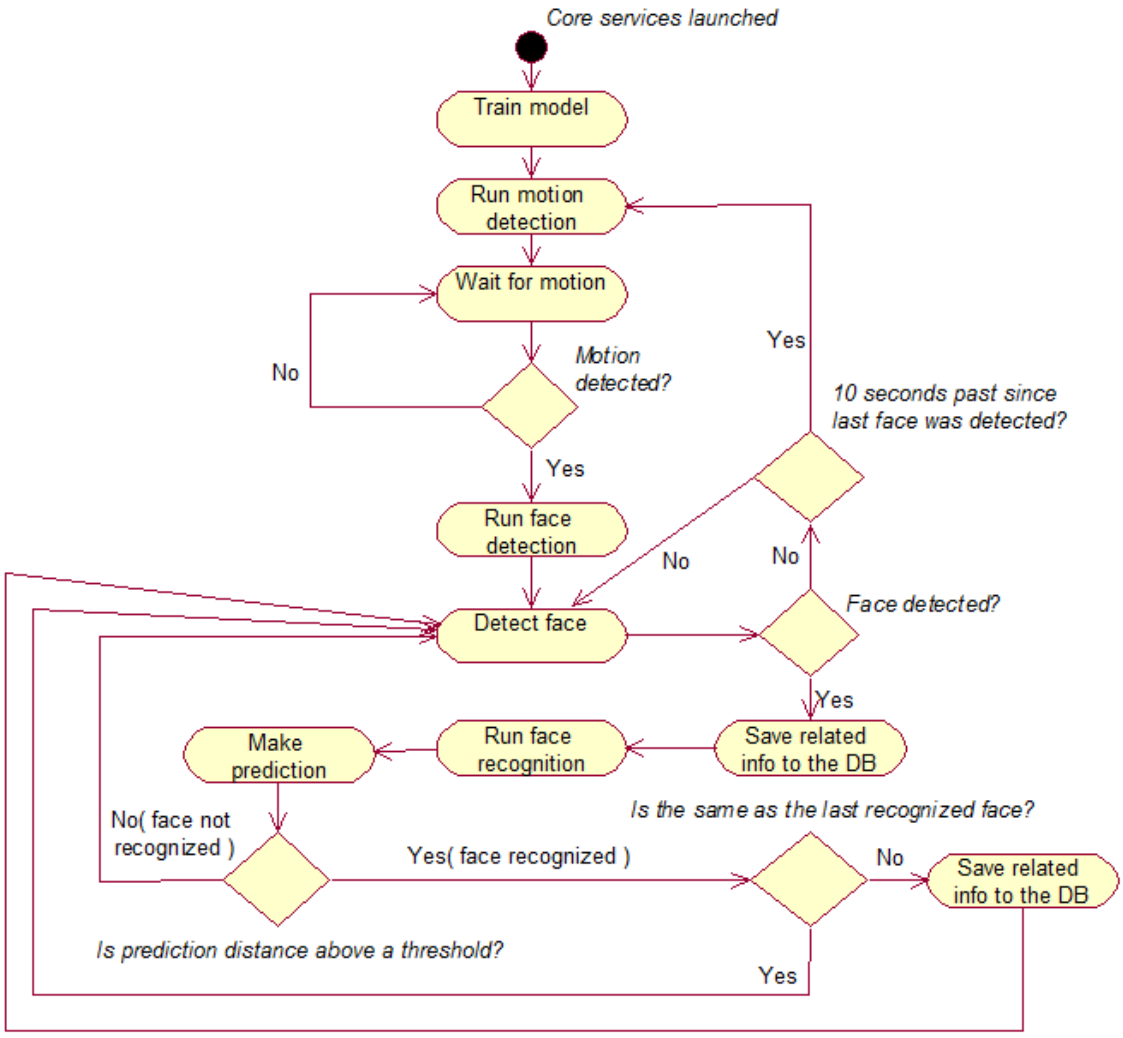


Figure 6. Activity diagram of the core services. Services are responsible for user identification using face recognition technology.

Core services provide three running modes:

1) **Normal mode**, when all features are working and outcome is visualized (i.e. captured video input is displayed in the separate window frame by frame, where detected and recognized faces are distinguished; also cropped grayscale picture of the detected face is displayed in the secondary window).

2) **Silent mode**, when all features are working without visualisation, except dataset saving feature, which is not available in the silent mode.

3) **Initialization mode**, when works only face detection and dataset saving along with visualization (with no face recognition and database connection).

3.2.3 User interface layer (Prototype)

Given section reviews the prototype of the user interface, which is considered to be the secondary part of this work.

User Interface is provided by a full-screen Java application, which should run constantly in the destination system. Application response depends on the underlying services, which receive and process notifications from the database and thus can provide a corresponding reaction to the obtained information.

Impersonalized screen

This is a systems main screen, which is displayed until some user is recognized. It contains no personal information, only some general data (e.g. time, weather, location, etc.), which does not depend on the specific user. Prototype of the impersonalized screen is shown on the Figure 7.

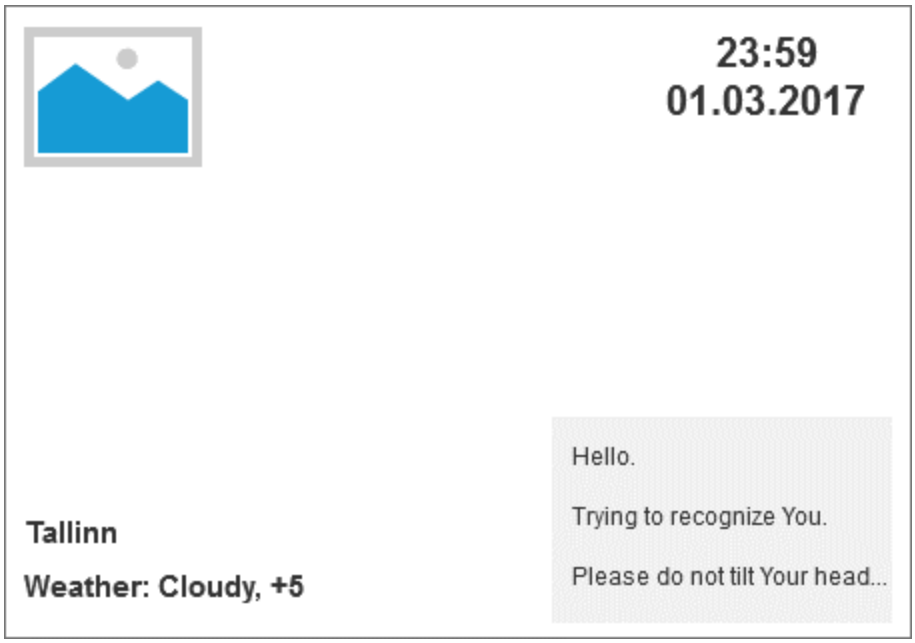


Figure 7. Prototype of the impersonalized system screen. System shows this screen when no user is recognized. Only the main information and system response is provided.

Registration screen

Registration screen appears when the user is recognized (i.e. user dataset is present), but has not been added to the application yet (see Figure 8). The screen contains the introduction form. After all needed data is provided, user may accept the registration or

decline it. If registration is successful, personalized user screen will be shown. If the registration is declined, then impersonalized screen appears.

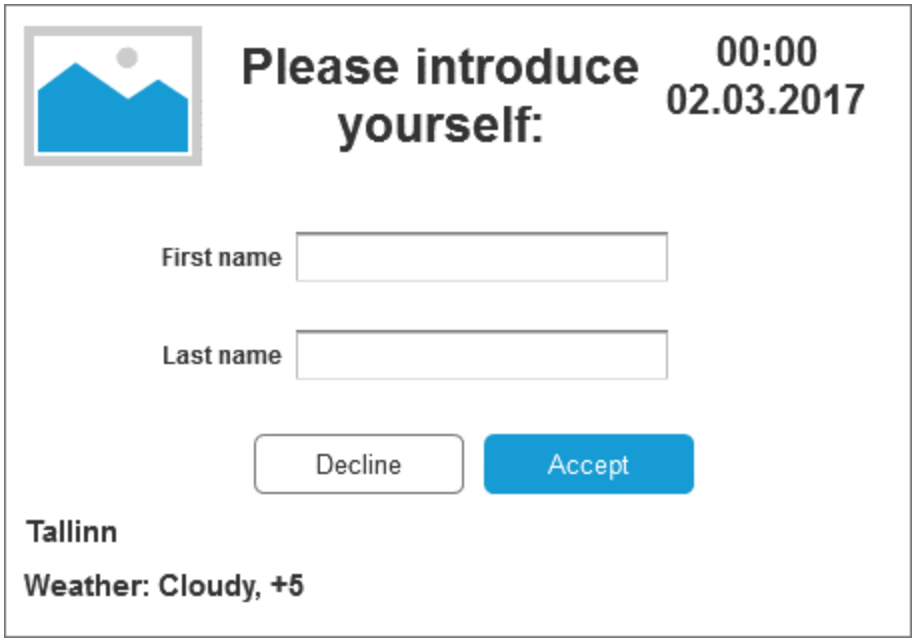


Figure 8. Prototype of the registration system screen. System shows this screen when user is recognized and tries to use the system for the first time. Registration form is provided along with the main information.

Personalized screen

Personalized screen appears when the application receives a notification from the service, that specific user was successfully recognized. This screen contains personalized information (like username and preferred system components) depending on the user, along with general information (like date, time, etc.). Preferred system components are displayed first and/or distinguished. Prototype of the personalized screen is shown on the Figure 9.

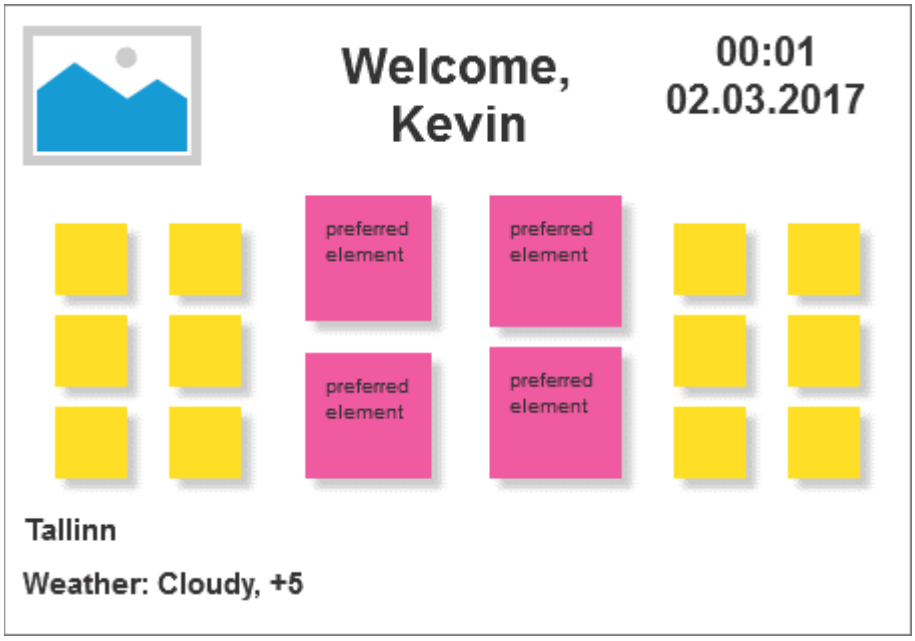


Figure 9. Prototype of the personalized system screen. System shows this screen when specific user is recognized. Personalized user-dependent information is provided along with general information. Preferred system components (pink boxes) are displayed first and/or distinguished.

3.2.4 Data layer

Accordingly to the data research provided in the section 2.3.1 and relying on system requirements, an object-oriented model is used for context data representation. Each software layer is involved in context data processing steps, which are reviewed in the section 2.3.2. The Figure 10 shows how context data processing steps are implemented in the scope of the given system.

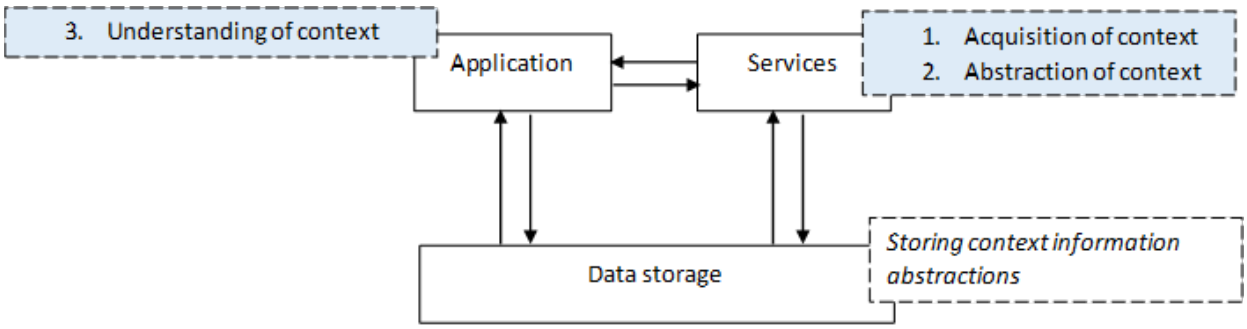


Figure 10. Schematic representation of a context data processing steps (solid blue boxes) in the scope of the given system (system components are represented by white boxes). Data exchange between components is designated by arrows. Each data processing step is numerated correspondingly (see Section 2.3.2).

Datasets

Datasets are used during face detection and recognition processes by core services (for more information about face recognition and datasets see Section 3.3 of this chapter). All datasets are located in the one specific folder, called “database”. Each folder in the database is considered to be a dataset. All pictures in the dataset are represented by individual files with determined resolution and of determined format and related to the one specific user. Structure of storing datasets is shown in the Figure 11.

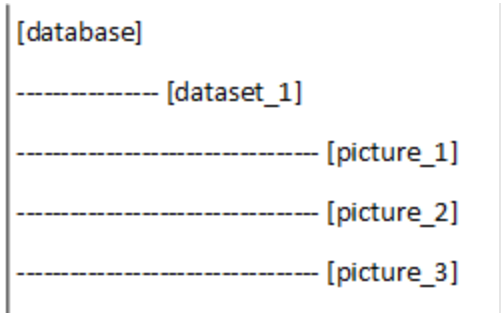


Figure 11. Folder structure of datasets storage. Placeholders for real folder and file names are included in square brackets.

Dataset names are used as identifiers and must be unique. Picture names are also unique and generated automatically by core features if face detection is launched in initialization mode.

Database

Context objects are represented by tables in the database. All data handling operations are provided as database functions, which are used by service layer. This approach ensures logical data independence.

Database consists of two schemas: “core” and “app”.

Core schema is related to the core services and contains context information, acquired from them. It includes information about detected and recognized faces and existing datasets. Class diagram of the core schema is provided on the Figure 12.

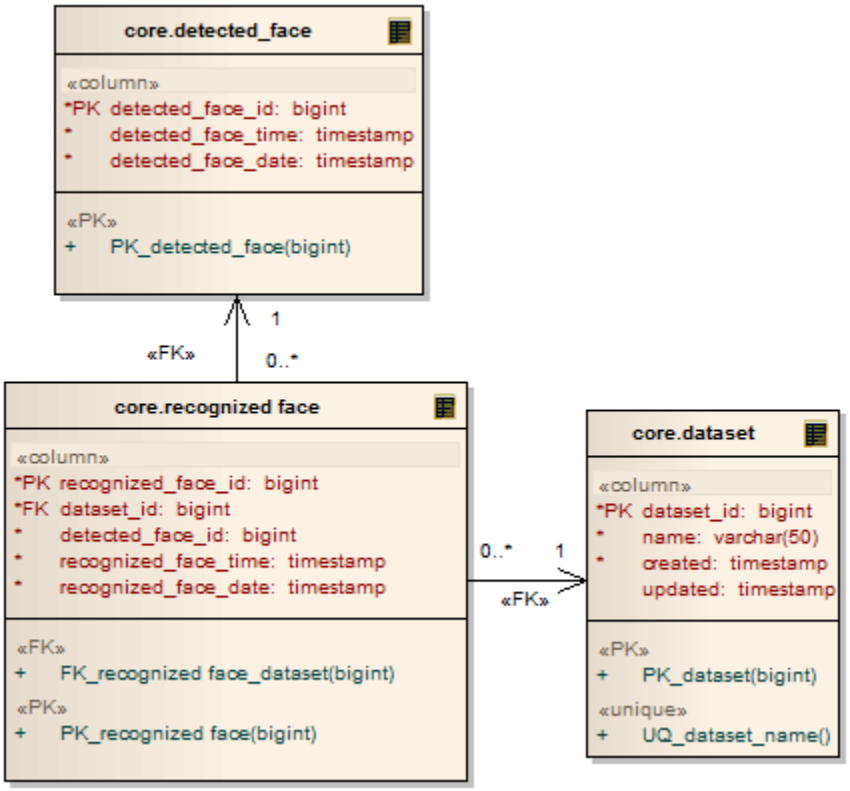


Figure 12. Class diagram of the core schema.

App schema includes all tables, related to the application itself. Each application user is related to the specific dataset in the core schema. In order to ensure a personalization of a system, "fact_" tables are provided: they include preference score of specific component related to the application user. Class diagram of the app schema is provided on the Figure 13.

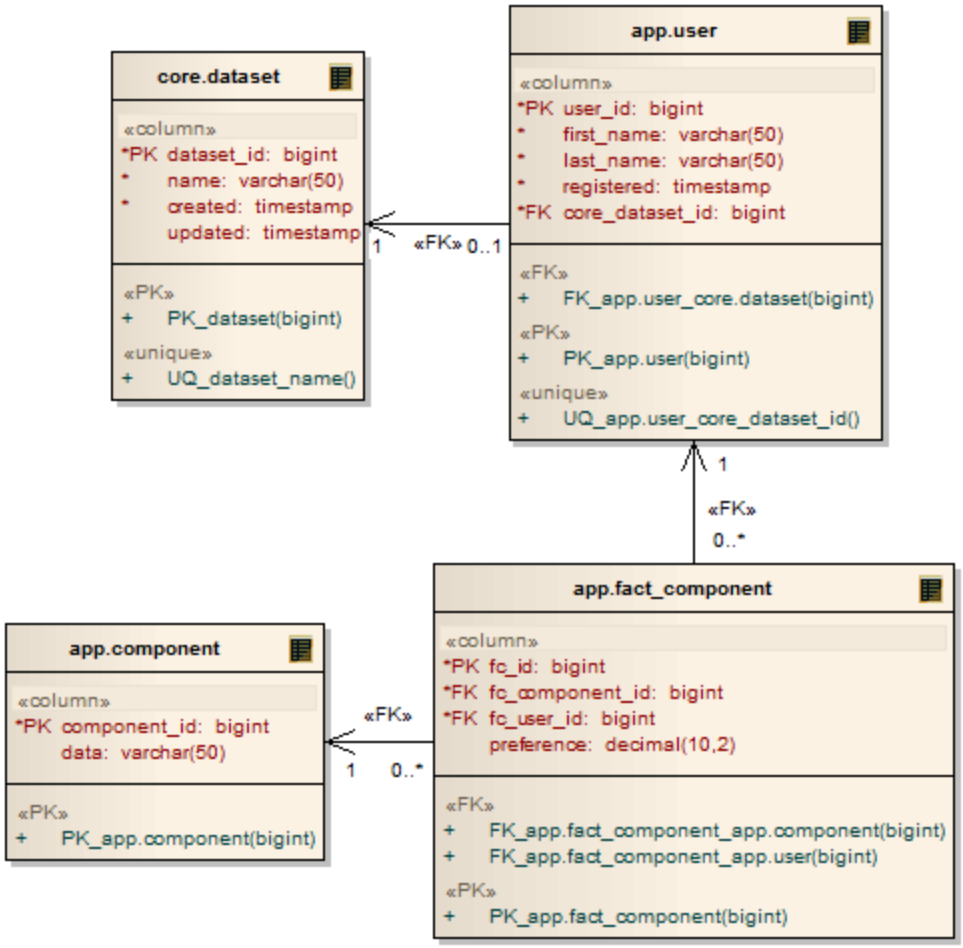


Figure 13. Class diagram of the app schema.

3.2.5 Used technologies

Table 1. Used technologies list separated by system components.

System part	Technology	Version / Model
DBMS	PostgreSQL [31]	9.4.10
Application	Java	1.8.0
	JavaFX [19]	8.0.31
Face recognition and motion detection services	OpenCV [28], [27]	2.4.9.1
	Python [32]	2.7.9
Hardware	Raspberry Pi [33]	3 Model B
	The Raspberry Pi Camera Module	v2
	Passive infrared motion sensor (PIR)	HC-SR501

DBMS

Corresponding to the data provided in the section 2.3.3 of this work, the most appropriate database for building a context-aware system is an ORDBMS.

For the system implementation PostgreSQL [31] is being used, because it supports an object-oriented extensions and a wide variety of server operating systems, is open-source, supports both Java and Python programming languages, sticks with SQL standards, is efficient for huge join operations and has a rich choice of data types.

There is also a research exists provided by M. Sarwat, J. L. Avery and M. F. Mokbel, where the prototype of context-aware system RECATHON was built on the PostgreSQL. RECATHON efficiency and performance in the real time were experimentally proved in the paper [36].

Database server is installed locally, i.e. on the same physical device along with other system parts. Considering a small amount of the system components, a problem with a lack of the resources does not arise, database is being used only by the given system and, thus, at the current stage of work there is no need to relocate the database on the separate machine.

However, a remote access is opened for the database administration, which allows to manage it without physically disturbing the running system (which may be difficult, if there is no input devices connected, e.g. mouse and/or keyboard).

Application

User interface is written in Java, graphical part is implemented using JavaFX library.

Face recognition and motion detection services

For face identification and recognition problem solution released under a BSD license open source OpenCV library and related modules are being used [28], [27]. Related code is written in Python [32].

Hardware

Correspondingly to the section 2.1.2 a fast computational device is needed to provide continuous system work. Such computer should be able to deal with high system load and resource consumption, which occur during context-aware computations and real time face detection. At the same time, because the system is intended in the first place for the everyday use by ordinary people, a hardware price is also important: it should be affordable for the statistical average family and total system price should be noticeable lower than the price of an ordinary personal computer.

All prototype functionality is running on the tiny Raspberry Pi 3 Model B computer [33]. It has a 1.2GHz 64-bit quad-core ARMv8 CPU which provides required system performance and 802.11n Wireless LAN, which provides possibility to use high speed wireless network connection.

The Raspberry Pi Camera Module is used as the video capturing device. It has a Sony IMX219 8-megapixel sensor, which provides great performance in a low-light

environment (which is critical for the face recognition task) and has a great sensitivity [17]. It can also capture 1080p video at 30 frames per second and 720p video at 60 frames per second, which makes captured video an appropriate input for real-time face recognition.

PIR sensor is responsible for motion detection.

Network

Proposed system is mostly network-dependent. The system design provided in this work allows to separate system components physically, providing communication between components via network.

3.3 Face recognition using OpenCV

Face recognition is a process of identifying and verifying face on the given graphical input. While face recognition is a trivial task for a human brain, it is a quite challenging for a computer to cope with. This section covers overview of three popular face recognition algorithms used in OpenCV - Fisherfaces, Local Binary Patterns Histograms (LBPH) and Eigenfaces (such related characteristics as performance and prediction accuracy are also reviewed below) - and provides an explanation of methods choice.

Alternative library

Along with OpenCV library OpenFace - another open source computer vision library - was also considered. OpenFace provides an implementation of face recognition based on deep neutral networks, which is based on Google article "FaceNet: A Unified Embedding for Face Recognition and Clustering" written by Florian Schroff, Dmitry Kalenichenko and James Philbin [39]. The library itself is written in python and relies on the OpenCV by using its pre-trained models for face recognition and some transformation methods.

OpenFace is powerful library, but requests more free resources than OpenCV. In the scope of the given work there is no need to use OpenFace features instead of OpenCV, because the last one provides all needed to satisfy system requirements and also not as exigent as OpenFace.

3.3.1 Face detection

Before we can start bothering about face recognition, we need to detect the face at first. The OpenCV uses cascade classifiers to perform a face detection task. The cascade classifiers training is beyond the scope of this work, so pre-trained open source xml files of cascade classifiers provided by OpenCV are being used instead.

Haar Cascade Classifier vs LBP Cascade Classifier

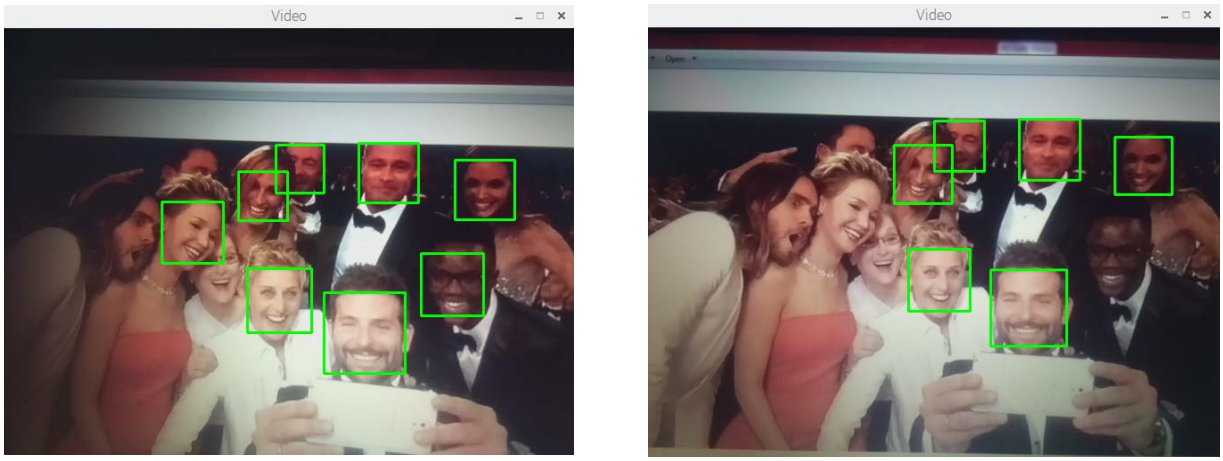
Cascade classifiers are being used to detect if a specific region of an image most likely contains the object of an interest or not. They consist of different levels, each of which is being applied subsequently to the given image region until some stage states that the region does not contain an object or until all levels are passed. Classifiers are being trained using the sets positive and negative samples of a target object (a face in the given case) [5].

Local Binary Pattern (LBP) operator assigns a label to every pixel of an image by thresholding the 3x3 neighbourhood of each pixel with the value of a central pixel and considering the result as a binary number [1]. The operator is robust to illumination changes and has a great performance provided by the simplicity of performed computations.

Haar features or Haar-like features are combinations of adjacent rectangles (dark and white), based on the Haar wavelets [14] and adopted for visual recognition tasks. There may be many Haar-like features per image, depending on situation [21]. Each feature is a single value determined by subtracting the average dark-region pixel value from the average light-region pixel value. The feature is considered to be present if obtained difference is above the threshold [20].

LBP Cascade Classifier is making all calculations in integers so it is faster than Haar Cascade Classifier, which performs calculations in floats. On the other hand, Haar is considered to be more accurate. But LBP can achieve same (or even better) results if trained properly (which also takes less time comparing to Haar training).

The performance is very important for user real-time identification task. So the small test was made to determine the appropriate method that should be used in a current work: same picture was shown from a separate screen to the Raspberry Pi Camera, processed by a small test program on the Raspberry Pi and then a screenshot of the returned output was taken (see Figure 14).



(a) (b)

Figure 14. Difference of face detection results using HAAR and LBB Cascade Classifiers with OpenCV. Pictures illustrate how face detection works using OpenCV cv2.CascadeClassifier.detectMultiScale method and LBP Cascade Classifier (a) / Haar Cascade Classifier (b) correspondingly. Detected faces are outlined by green squares. Due to the fact that initial picture was shown manually, final image scale and position may slightly differ.

The best result of using LBP Cascade Classifier is 8 recognized faces out of 10, 7 faces in average. The Haar Cascade Classifier recognized only 6 faces of 10 in the best case and 5 in average. Both methods have a small delay: 3 seconds for LBP and 9 for Haar (see Table 2).

Table 2. Results of the comparison of the Haar- and LBP Feature-based Cascade Classifiers in action.

	Test 1 X of 10 faces recognized	**Test 2** X of 10 faces recognized	**Test 3** X of 10 faces recognized	**Test 4** X of 10 faces recognized	**Total faces recognized**	**Average faces recognized**	**Delay** Average sec
Haar	6	5	5	5	**21**	**5,25**	**9**
LBP	8	7	8	6	**29**	**7,25**	**3**

Also there is a related research, where LBP and Haar features were compared based on detection hit rate and detection speed, which results show that Local Binary Pattern features are the most appropriate for the implementation of a real-time face detection systems [20]. Therefore the best method for cascade classifiers in the given conditions is LBP, thus it should be used in the proposed system.

3.3.2 Face recognition

Appropriate solution selection

There are three face recognition algorithms available in the OpenCV:

1. Fisherfaces

The idea of the algorithm is simple: same classes should cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation [11]. Fisherfaces is based on Linear Discriminant Analysis: image dimensions are being reduced based on specific classes.

2. Local Binary Patterns Histograms (LBPH)

Algorithm looks only at local features of an object, comparing each pixel of an image with its neighbourhood [11]. After constructing a histogram for each independent image region, compares images by comparing histograms of their corresponding regions [25].

3. Eigenfaces

The backbone of Eigenfaces is Principal Component Analysis (PCA). The image matrix is represented by a large number of dimensions, which are not all equally useful. So the main idea of PCA is to reduce high dimensionality by turning a set of possibly correlated variables into a smaller set of uncorrelated variables, retaining dimensions which account for most of the information.

In order to be able to recognize an object, each algorithm should be previously trained on a set of examples: different pictures of the object that should be recognized. Algorithms behave differently at the given stage: LBPH handles each image in the given set

independently, while Eigenfaces and Fisherfaces algorithms work with a set of the images as a whole, computing a mathematical description of its most dominant features.

Although LBPH can be a little bit slower comparing to other two algorithms, it shows good successful recognition results in different environments and also may work better in different light conditions [46], [44], [2]. At the same time it needs a good initial set of images to work efficiently.

Due to the fact that in this work we are dealing with a personalized, user-dependent system, it is very important to provide a maximum possible recognition accuracy, thereby Local Binary Patterns Histograms should be used as face recognition mechanism.

Model training and related pitfalls

Model training is an essential part of an object recognition process independently of model implementation. Algorithmic prediction of a face is dependent on the dataset of a number of frontal face pictures for each existing user. Trained model compares detected face with an existing data and provides a numerical output, which is called prediction confidence or distance, where a zero distance means a hundred percent of confidence. While a distance of a prediction grows, the prediction accuracy decreases. Too large distance number means, that face is not recognized (see Figure 15).

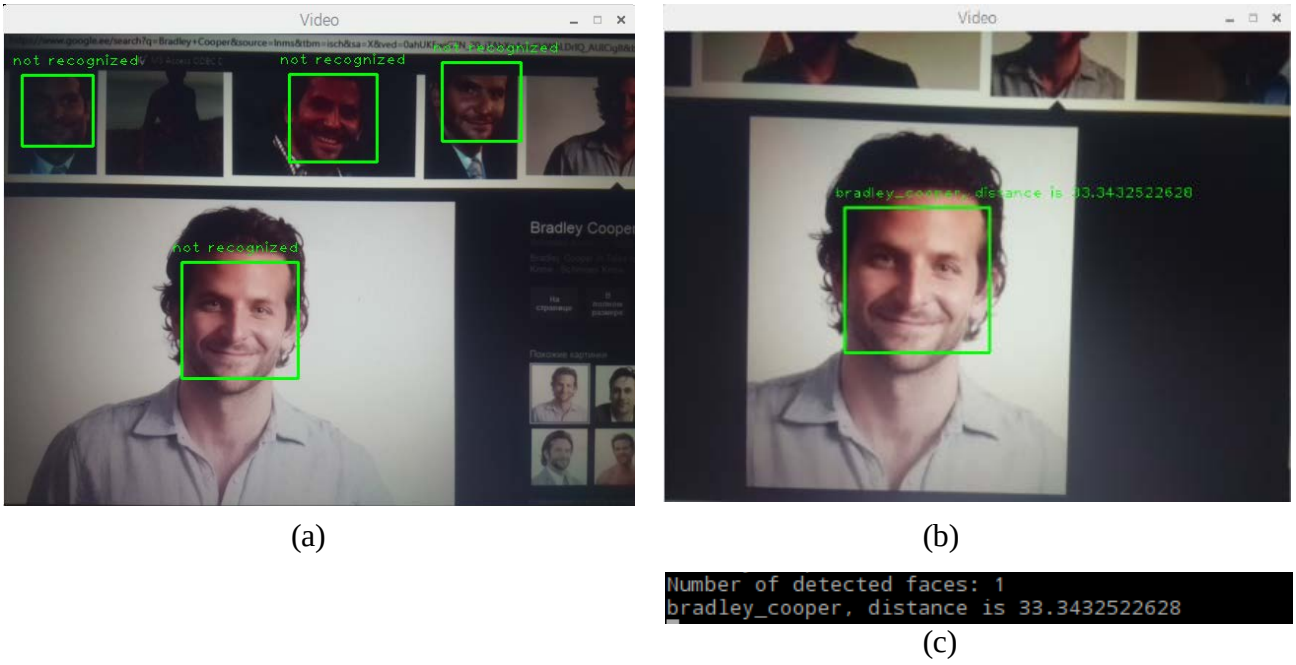


Figure 15. Face recognition results before and after model training. Bradley Cooper is not recognized by the system (a) until the model is trained (b). Face detection worked in both cases, but at the second time system was able to give a prediction with a small distance (c), which means a good confidence. Detected faces are outlined by green squares, prediction result is displayed above them. Scale and position of the pictures may differ due to the fact that initial picture was shown to the camera manually.

Each time when a new face is being added to the database it stays unused during recognition process until the model is trained over again, so that new files are being taken into account. Model can't be trained if it has access to the only one dataset, or no datasets at all.

For each system must be defined some point, defining prediction accuracy threshold (i.e. confidence threshold). If a threshold is too low, then a minimum (if any) faces will be recognized. On the contrary, very high threshold leads to the false positives: a lot of predictions may be wrong. Prediction results vary depending on datasets used: not only quantity of pictures in dataset is sufficient, but also size, quality, scale and position of a depicted face. The Figure 16 shows a face recognition results in case of insufficient

amount of training data: even with a high confidence threshold (500) there is no successful prediction.

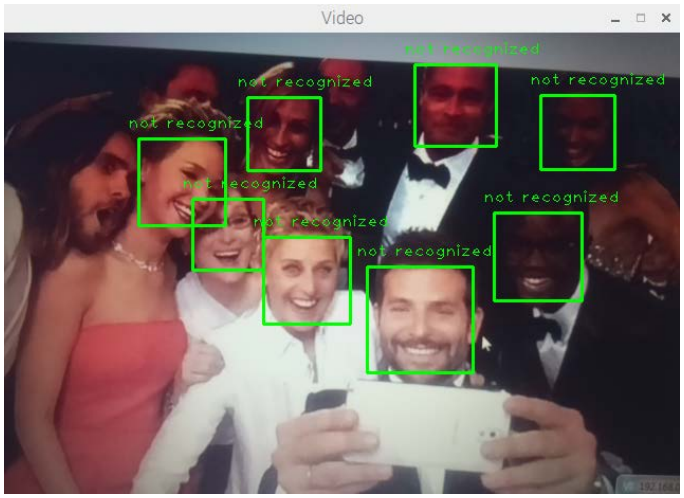


Figure 16. Face recognition with a large confidence threshold and insufficient amount of the datasets. Face recognition is not able to give a confident prediction with high confidence threshold when there is a lack of datasets for model training. Detected faces are outlined by green squares, prediction result is displayed above them.

However, if the model was trained using a lot of bad-quality data (see Figure 17 (a.1) and Figure 17 (a.2)), then defining a high prediction threshold (500) leads to high percent of errors (see Figure 18 (a)). Lowering a confidence threshold (120) may help in that case, but a confidence of predictions remain weak and a number of wrong predictions stay high (see Figure 18 (b)).

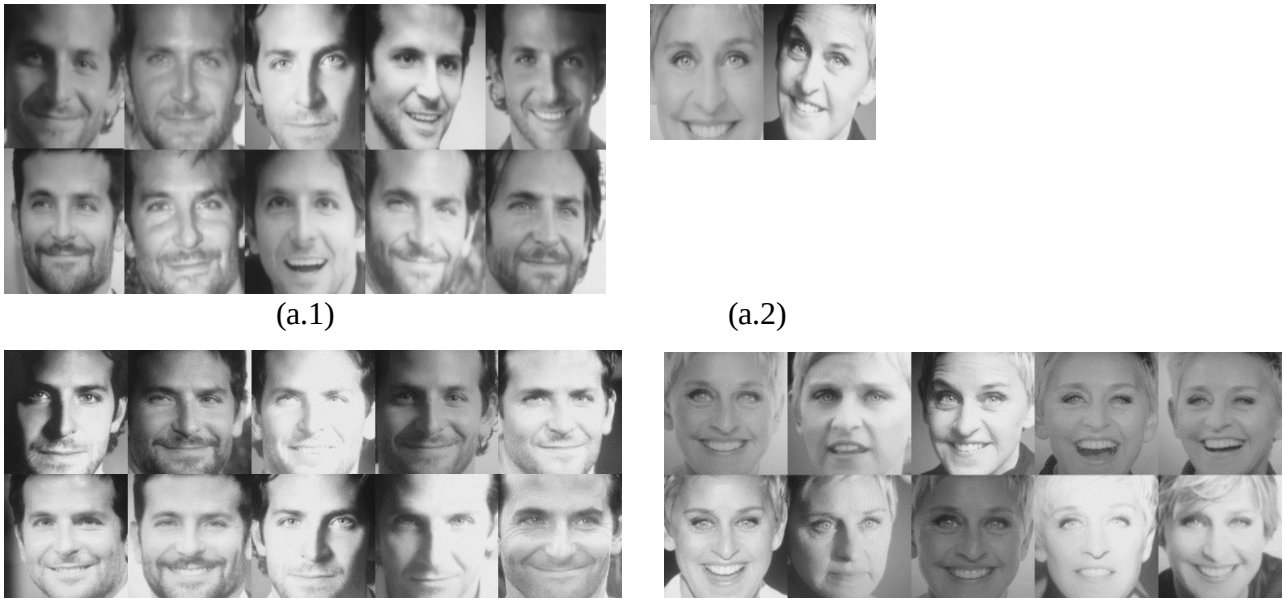


Figure 17. Difference between bad- and good-quality face recognition datasets. Pictures assembled of Bradley Cooper and Ellen DeGeneres datasets. Datasets (a.1) and (a.2) are considered to be of bad-quality: they contain narrow pictures (92x112 px) of faces, some of which are not fully front face and others are rotated, so eyes are not located at the same level, which makes face recognition more complicated; also dataset (a.2) does not include sufficient pictures in it. Datasets (b.1) and (b.2) are of a good quality: pictures are wide enough (200x200 px), heads on the photos are not rotated and eyes are on the same line.

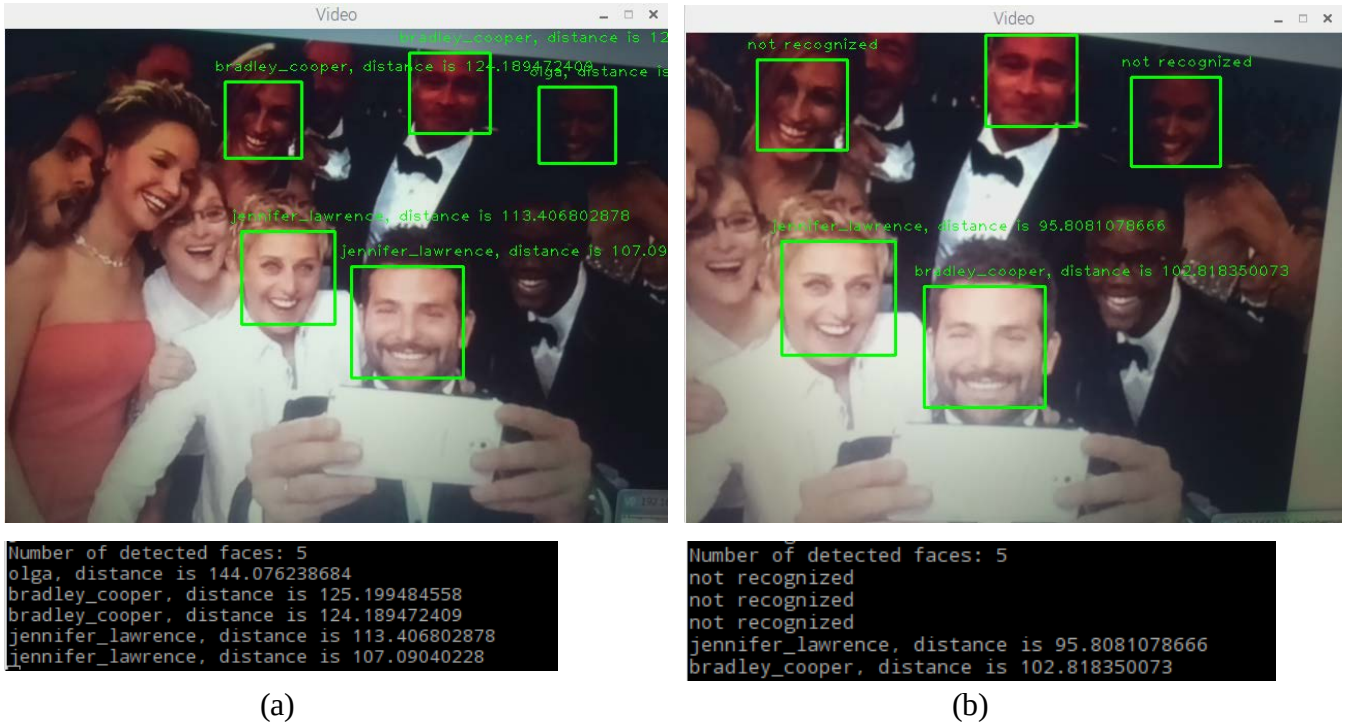


Figure 18. Face recognition with different confidence thresholds using a bad-quality datasets. Face recognition is giving wrong predictions with high confidence threshold if model is trained using bad-quality data (a). Lowered confidence threshold helps to remove some errors, but false predictions remain (Ellen DeGeneres is recognized as Jennifer Lawrence) (b). Detected faces are outlined by green squares, prediction result is displayed above them and on a corresponding screenshots of a terminal window.

In order to increase a confidence rate of predictions (i.e. to shorten the prediction distance) a model should be trained using good-quality datasets (see Figure 17 (b.1) and Figure 17 (b.2)). Properly trained model may also make some mistakes when an inflated confidence threshold (500) is set (see Figure 19 (a)), but gives mostly correct predictions with an appropriate confidence threshold (200) (see Figure 19 (b)).

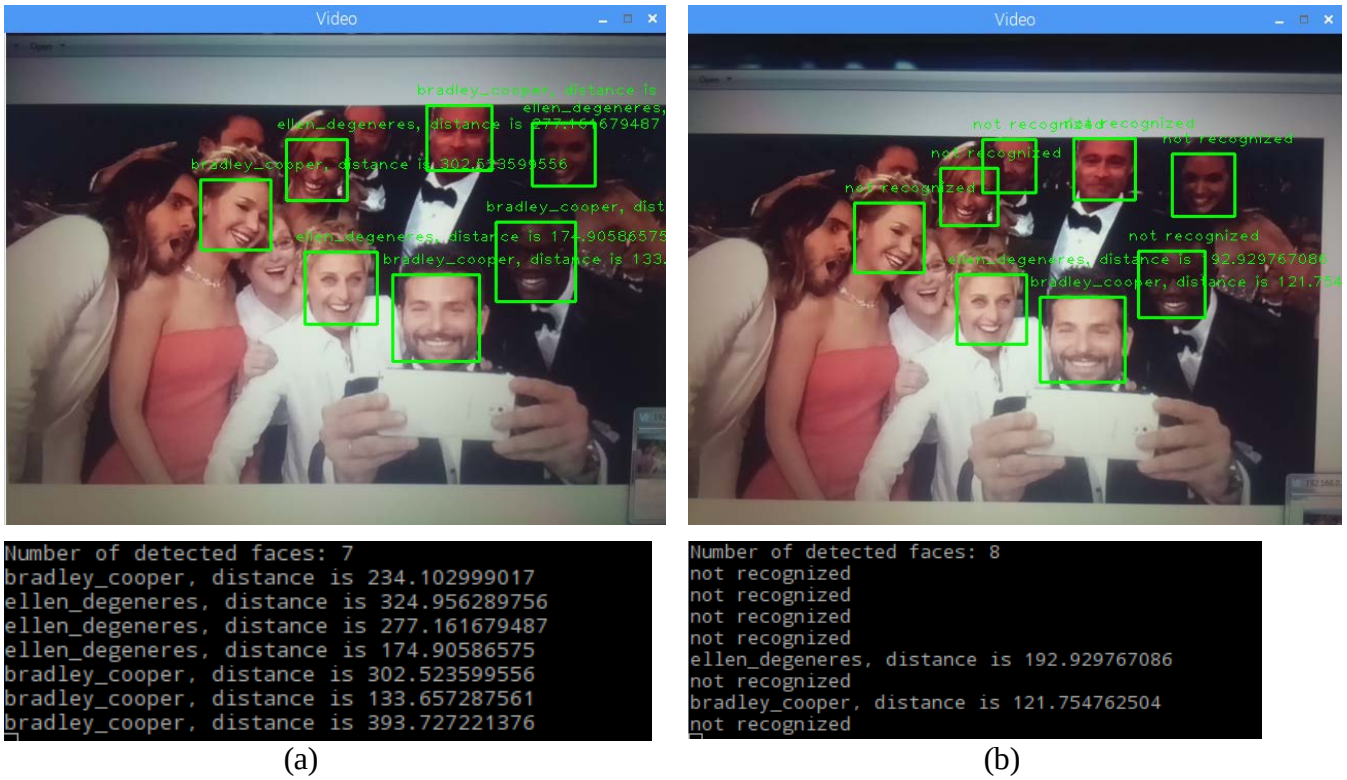


Figure 19. Face recognition with different confidence thresholds using a good-quality datasets. Face recognition is giving wrong predictions with a high confidence threshold if the model is trained using good-quality data (a). Lowered confidence threshold helps to remove false predictions (b). Detected faces are outlined by green squares, prediction result is displayed above them and on a corresponding screenshots of a terminal window.

3.4 Technical solutions

This section describes some general technical solutions of implementation tasks concerning all system layers.

3.4.1 Python services

Configuration

All general configuration of python services is located in the /config/config.properties file. The file is divided into sections, each section is responsible for single configurable feature and holds corresponding property names and values. Example of the property file content is provided at the Code sample 1.

```
[DatabaseOfFaces]
database.directory=database
database.face.width=200
. . .
[FaceRecognition]
recognition.threshold=150
[Database]
database.host=localhost
database.port=5431
database.name= . . .
```

Code 1. Property file for Python services. Content and structure sample. Sections names are in the square brackets, property names on the left side of equals sign and corresponding property values are on the right side respectively.

Property reading implemented using Python's ConfigParser module, which provides an implementation of configuration file parser. Each service gets an appropriate property value by property name, as represented at the Code sample 2.

```
import cv2, sys, ConfigParser, os, numpy
. . .
config = ConfigParser.RawConfigParser()
config.read('config/config.properties')
. . .
def saveFaceImage(username, grayscale_image):
    # Initialize properties from config file
    database = config.get('DatabaseOfFaces', 'database.directory')
    file_format = config.get('DatabaseOfFaces', 'database.face.format')
. . .
```

Code 2. Reading properties from config.properties file in Python.

Database connection

Psycopg - PostgreSQL adapter for the Python language – is used in order to establish the database connection. Connection is created while initializing database connection service, i.e. after importing the corresponding service file into the core service, which means that attempt to connect to the database occurs at the beginning, when core service is launched. A code responsible for database connection creation is represented at the Code sample 3.

```
import psycopg2, ConfigParser
. . .
# Getting database connection properties
db_host = "host='" + config.get('Database', 'database.host') + "'"
db_port = "port='" + config.get('Database', 'database.port') + "'"
. . .
# Trying to establish a connection
try:
    print "Connecting to the database..."
    conn_parameters = db_name + " " + db_user + " " + db_password + "
" + db_host + " " + db_port
    conn = psycopg2.connect(conn_parameters)
    print "Database connection established."
except:
    print "Failed to establish database connection."
. . .
```

Code 3. Python code responsible for establishing database connection using psycopg2 module.

All communication with database takes place via database functions. Example of function call with parameters is represented at the Code sample 4.

```
def registerFaceRecognition(dataset_name, detected_face_id):
    cur = conn.cursor()
    cur.callproc('core.recognized_face_insert', [dataset_name,
detected_face_id])
    row = cur.fetchone()
    cur.close()
    conn.commit()
    return row
```

Code 4. Python function responsible for calling parameterized DB function.

3.4.2 Application

Database connection and configuration

Separate configuration file is provided for the database connection properties. It contains database url, username and password in a plain text format and may be quickly changed if necessary.

The properties are initialized during the application start-up process and stored as the instance of the Properties class of the java.util package. Core service of the Java application among other is responsible for storing database properties and creating connections. The Code sample 5 demonstrates the related basic functionality of the CoreService class.

```
import java.sql.Connection;
import java.util.Properties;
import . . .

public class CoreService {
	. . .
	private Properties properties;

	public CoreService() {
		try {
			ititializeProperties();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
	. . .

	private Connection createConnection() throws SQLException {
		String databaseUrl = properties.getProperty("database.url");
		String databaseUser = properties.getProperty("database.user");
		String databasePassword =
properties.getProperty("database.password");

		return DriverManager.getConnection(databaseUrl, databaseUser,
databasePassword);
	}
. . .
}
```

Code 5. Basic functionality of the CoreService class. Implementation of the further use of the property file for creating a database connection is demonstrated.

PostgreSQL JDBC Driver is used to create database connections.

Notification listener

The context-awareness of the application is determined also by its fast response to the changing context. The application is aware of the occurring changes due to the database notifications (see Section 3.4.3 for more info on notifications).

Implemented listener class is responsible for database notifications processing: it determines possible notification channels (see Code sample 6), executes LISTEN command in the database (i.e. starts listening to the NOTIFY command of the given type see Code sample 7) and sends information about received notification to the user interface shell.

```
public enum Messages {
            USER_DETECTED("user_detected"),
            USER_RECOGNIZED("user_recognized"),
            NO_MOVEMENT("no_movement");

            . . .
```

Code 6. Enumerator for representing database notification channels.

```
public NotificationListener(Connection conn) throws SQLException {
       NotificationListener.pgconn = (PGConnection) conn;
       Statement stmt = conn.createStatement();
       stmt.execute("LISTEN " + Messages.USER_DETECTED.getValue());
       stmt.execute("LISTEN " + Messages.USER_RECOGNIZED.getValue());
       stmt.execute("LISTEN " + Messages.NO_MOVEMENT.getValue());
       stmt.close();
}
```

Code 7. LISTEN commands are executed in the constructor of the notification listener object.

Notification listener is running continuously as the different thread in parallel with UI thread. Code sample 8 demonstrates the life cycle of the notification listener.

```
public class NotificationListener extends Thread {

	. . .

	public void run() {
		while (true) {
			try {
				getNotifications();
				Thread.sleep(500);
			} catch (SQLException sqle) {
				sqle.printStackTrace();
			} catch (InterruptedException ie) {
				ie.printStackTrace();
			}
		}
	}
```

Code 8. Life cycle of the notification listener thread.

User interface

Application is represented by JavaFX Application class and consists of primary stage and corresponding scene. Each application screen is represented by different scene, which consists of border pane and its child elements. Switching between screens mean changing the currently displayed scene on the primary stage.

Entire work of the user interface graphical components is managed by user interface shell: a general class, which is responsible for initializing the general screen for the first time, switching between screens and handling received messages from the notification listener class: messages are being processed correspondingly and required actions are performed in the thread of JavaFX application. As an example, the Code sample 9 demonstrates how shell handles a notification about absence of movement.

```
private static Application application;
. . .
else if (notification.getName().equals(Messages.NO_MOVEMENT.getValue())) {
                    Platform.runLater(new Runnable() {
                   @Override public void run() {
                    application.removePersonalizedScreenIfExists();
                    application.showGeneralScreen();
. . .
```

Code 9. Example of UI shell notification handling.

3.4.3 Database

Logical data independence is ensured on the database level by implementing the set of functions, which are responsible for handling database operations requested by other system layers. Thus, the logic of the insert and update operations is defined inside the functions written in PL/pgSQL, not inside the application code.

Notifications

Database is also responsible for sending notifications about the changes in the core schema. Due to the fact that system handles dynamic context, it is common that related information is being frequently updated. Thereby database considered to be the best place to send notifications about data changes, as it is directly involved in the processing those changes.

The notifications are send using NOTIFY command provided in the PostgreSQL database. Database layer does not care about notification receiver at all and will continue to send notifications even if no one listens. Thus any interested client should execute LISTEN command in order to be able to receive notifications from the database.

There are three types of the notifications in the system: when user is detected, when user is recognized and when there is no movement present for ten seconds or longer. The first two notifications are related to the data changes, so are being executed right after insert operation in the same function, as shown at the Code sample 10. The notification about no present movement is implemented as stand-alone function, for the reason that it is not related to any data changes in the database.

```
BEGIN
. . .
    INSERT INTO core.recognized_face( . . . )
    VALUES( . . . )
    RETURNING detected_face_id
    INTO inserted_id;

    PERFORM pg_notify('user_recognized', _dataset_id::varchar(50));

    RETURN inserted_id;
END;
```

Code 10. Example of sending a notification from the database function responsible for a data insert. Code is written in PL/pgSQL, presented notification is being sent when user is recognized.

3.5 Environment setup

This section provides a description of software preparations that should be made to provide proper system functioning in the working environment. Presented instructions are related to the system exploitation and maintenance only, thereby environment setup required for system development and testing is beyond the scope of the given section.

Operating system

System is designed to work with Debian-based OS optimized for Raspberry Pi hardware – Raspbian [34]. Prototype is built and tested using Raspbian GNU/Linux Version 8 (jessie). All environment setup information provided below concerns this particular OS version. It is theoretically possible to run the system on the other Debian-based OS if all needed components (e.g. packages and JVM) are respectively installed.

Size of a swap file must be increased in order to ensure the performance required by the system features: for example, prototype is configured to use 1024mb.

Required packages

For build and compile support: build-essential, pkg-config, cmake

Required by OpenCV: libgtk2.0-dev (graphical user interface library), libavcodec-dev (codec library for video streaming), libavformat-dev (demuxer library), libswscale-dev (video scaling library), libtbb2 (for multi-core processor performance handling), libtbb-dev (for multi-core processor performance handling), libjpeg-dev (files for jpeg library), libpng-dev (files for png library), libtiff-dev (Tag Image File Format support), libjasper-dev (for image handling), libv4l-dev (provides abstraction for camera device)

Python: python2.7-dev, python-numpy (support of multidimensional arrays)

(Package management system for python - pip [30] - should be preinstalled.)

PostgreSQL: postgresql-9.4, psycopg2

Application: oracle-java8-jdk

Some of the packages listed above may be already pre-installed depending on the OS version. Provided package versions are also important. Changing a version may cause the incompatibility problems.

OpenCV

OpenCV 2.4 must be cloned from git, compiled and built according to the official instructions provided in the OpenCV documentation and available online [18]. The library version is very important, since using a newer library release may cause incompatibility problems.

Database

Database installation scripts should be executed on the destination machine in order to setup an initial empty database with the prepared structure. Core services and application are configured to address to localhost database located on the 5431 port by default, but these settings as well as username and password may be changed by providing preferred values in the corresponding .properties files (different for application and core features, see Sections 3.4.1 and 3.4.2 for more info). Configuration files are located in the "config" (core fetures) and "resources" (application) folders correspondingly.

Core services

Face recognition and face detection services are provided as Python files along with shell scripts included to run them in general modes. No additional setup is needed as long as all required packages are installed.

Application

User interface is implemented as executable jar file. No additional setup is needed as long as all required packages were installed.

Datasets preparation

This is important to have at least two existing datasets (i.e. two different sets consistent of the face pictures, which are belong to the two different people correspondingly) before running the system. Pictures in the datasets must be properly aligned, be of the same size, only one face must be represented at the each picture. The preferred minimal number of

the pictures in one set is 10, but it should be rather bigger than the mentioned minimum. More detailed explanation about importance and meaning of the datasets quality is provided in the section 3.3.2, refer to it for additional information on the datasets.

It is possible to use system features to prepare first datasets if needed. System must be launched in initialization mode, which means that face recognition, motion detection and a connection with the database will not work. In this mode face detection runs along with visualization process (i.e. the video stream processing will be represented as graphical output) and detected face may be saved if needed.

3.6 System restrictions and future work

Due to the limitations related to the topic and a wish to concentrate on the core system, some features and corresponding issues were postponed until next releases. The given section provides a brief overview of the topics that were not considered in this work and will be an object of the future works as well as an overview of the features and solutions which implementation is postponed until next releases.

Restrictions and future work

1. The privacy issues are beyond the scope of this work. A privacy problem requires an independent precise analyse, which could be a topic of a distinct work in the future.

2. Server centralization / decentralization issues are beyond the scope of this work. Some related information can be found in the [38] paper.

3. Handling out-of-context objects issue [26] is beyond the scope of this work.

4. Query performance could be improved using Bloom filters as suggested in [40].

5. Since the proposed system is closely related to the dynamic context processing, it requires mechanisms and strategy of information deletion [15]. See also Section 2.2.1 of the current work. Such mechanism design and implementation are out the scope of this thesis, but may be considered as future work.

6. Python and OpenCV versions are not up to date due to the occurred incompatibility issues with newer versions (3.6 and 3.2 correspondingly). Incompatibilities may be settled in the future and system may be improved to support newer Python and OpenCV versions.

7. The paper focuses mainly on the face recognition problem solution, design of the related services and on the general system architecture. Thus, at the given stage of the work the application part (including user interface) is at the sketch phase and can be gradually developed in the future.

Technical debt

1. Exception handling should be improved at the all layers. Additional analyse and strategy are required.

2. Saving faces must be improved since the position of face is essentially important in the face recognition process. Although the face alignment script provided in OpenCV documentation [10] was correspondingly worked up and adapted for the current system, it still needs an improvements and despite it is included to its core functionality, for now some of its parts still can't be used completely due to the existing bugs.

3. Logging mechanisms should be implemented at the all layers to provide robust system handling opportunities.

4. Known vulnerability of the system is the possibility to bypass the user identification mechanism and cheat the algorithm by showing the photo to the camera. Elimination of this gap is postponed to the next releases.

3.7 Comparison with existing systems

Given section contains an overview of the alike systems, which functionality and implementation resemble the system proposed in this paper. Although there are stand-alone realisations of face recognition and detection tasks where similar technologies were used, they usually do not include a personalized application part (and vice versa), so no exact match with the proposed solution was found during the survey of existing systems. Thus not the entire systems, but their main design and realisation principles were compared in the scope of this chapter, without addressing to the particular systems separately, but by deriving main statistics of the characteristics mentioned above.

Facial recognition projects and personalization systems are represented by a wide range of available implementations, both commercial and open source. In order to provide an accurate comparison, complex commercial systems were not considered in the scope of this chapter, and the main accent was made on the simple light-weight solutions with perceptible implementation similarities.

There is variety of simple face recognition projects written in OpenCV, but many of these projects are devoted to demonstrate the face recognition working principles without any further purpose. But if the face recognition is being used as user identification mechanism, the main used methods stay the same: the most popular (statistically) cascade classifier features used for face detection in these projects are Haar-features and the most popular algorithm for face recognition is Eigenfaces. These choices does not consider the performance and prediction accuracy issues, which are essential for our system: such systems perform real-time face detection with delay more than 3 seconds due to the Haar-features, and false-positive or false-negative prediction outcome may occur more often comparing to our system, because of the lower adaptive rates of used algorithm in different environments (see Section 3.3.2 of this work for further information on used face recognition technologies).

4 Summary

The aim of this work was to reduce a number of actions performed during human-computer interaction process, simplifying it and making it intuitively understandable by building a personalized context-aware user identification system for stationary devices to demonstrate possible approaches to solve that goal.

A thorough research of the related studies was conducted in this paper. Special attention was devoted to the existing practices and models of handling and representing a contextual information. Provided survey allowed to compare different approaches and derive appropriate methods and technologies for designing the solution of the stated problem.

The face recognition is considered to be the main problem in the scope of the modelled system. Different methods were tested in a real-life conditions and obtained conclusions on this topic were documented in this work. As a result, user identification system prototype for stationary systems was designed and built on a Raspberry Pi.

Provided solution ensures the availability and compactness of the system, proposed architecture allows to separate system components across different devices. Implemented system helps to hide an identification process from the end user through context-information handling skills.

Based on the foregoing author may state, that main aims of the given work were successfully fulfilled. Also conducted research provides a solid base for the future researches and improvements and may be a topic foundation for the master's thesis.

References

[1] Ahonen T., Hadid A., Pietikainen M. Face description with local binary patterns: Application to face recognition. – *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006, Volume 28, Issue 12, 2037 - 2041.

[2] Anonymous Author(s). Learning to Recognize Faces in Realistic Condition. – *The University Of British Columbia, Faculty of Science, Department of Computer Science,* 2013. [*Online*] http://www.cs.ubc.ca/~nando/540-2013/projects/p51.pdf (09.05.2017)

[3] Bauera, C., Dey, A.K. Considering context in the design of intelligent systems: Current practices and suggestions for improvement. – *Journal of Systems and Software*, 2016, Volume 112, 26–47.

[4] Brown, P.J., Bovey, J.D., Xian Chen. Context-aware applications: From the laboratory to the marketplace. – *IEEE Personal Communications*, 1997, Volume 4, Issue 5, 58-64.

[5] Cascade Classification OpenCV. – *OpenCV 2.4.13.2 documentation.* [*Online*] http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html (09.05.2017)

[6] Chen, G., Kotz, D. *A Survey of Context-Aware Mobile Computing Research*. – *CiteSeerX, Technical report, Dartmouth College,* 2000. [*Online*] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.4330 (20.12.2016)

[7] Dey, A.K. Understanding and Using Context. – *Personal and Ubiquitous Computing*, 2001, Volume 5, Issue 1, 4–7.

[8] Dourish, P. What we talk about when we talk about context. – *Personal and Ubiquitous Computing*, 2004, Volume 8, Issue 1, 19–30.

[9] Ejigu, D., Scuturici, M., Brunie, L. An Ontology-Based Approach to Context Modeling and Reasoning in Pervasive Computing. – *PERCOMW '07 Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, 2007, 14-19.

[10] Face Recognition with OpenCV, Aligning Face Images. – *OpenCV 2.4.13.2 documentation.* [*Online*] http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#aligning-face-images (13.05.2017)

[11] Face Recognition with OpenCV. – *OpenCV 2.4.13.2 documentation.* [*Online*] http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html (09.05.2017)

[12] Feng, L., Apers, P.M.G., Jonker, W. Towards Context-Aware Data Management for Ambient Intelligence. – *Database and Expert Systems Applications*, 2004, 422-431.

[13] Gray, P., Salber, D. Modelling and Using Sensed Context Information in the Design of Interactive Applications. – *Engineering for Human-Computer Interaction*, 2001, 317-335.

[14] Haar, A. For Orthogonal Function Systems Theory. – *German Mathematical Research Journal*, 1910.

[15] Hoareau, C., Ichiro, S. Modeling and Processing Information for Context-Aware Computing: A Survey. – *New Generation Computing*, 2009, Volume 27, Issue 3, 177–196.

[16] Huibing, W. Extending object-relational database to support spatio-temporal data. – *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2008, Volume XXVII, Part B2.

[17] IMX219PQ. – *Sony Semiconductor Solutions Corporation*. [*Online*] http://www.sony-semicon.co.jp/products_en/new_pro/april_2014/imx219_e.html (19.04.2017)

[18] Installation in Linux. – *OpenCV 2.4.13.2 documentation*. [*Online*] http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html (13.05.2017)

[19] JavaFX - The Rich Client Platform. [*Online*] http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html (05.05.2017)

[20] Kushsairy, K., Kamaruddin, M.K., Haidawati, N., Safie S.I., Zulkifli A.K.B. A Comparative Study between LBP and Haar-like features for Face Detection Using OpenCV. – *IEEE 2014 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*, 2014.

[21] Lienhart, R., Maydt, J. An Extended Set of Haar-like Features for Rapid Object Detection. – *IEEE, Proceedings. International Conference on Image Processing*, 2002.

[22] Lyytinen, K., Yoo, Y. Issues and challenges in ubiquitous computing. – *Communications of the ACM*, 2002, 45(12), 63–65.

[23] Mattern, F., Sturn, P. 2003. From Distributed Systems to Ubiquitous Computing-State of the Art. Trends and Prospects of Future Networked systems. – *Fachtagung_Kommunikation in Verteilten Systemen(KiVS), Leipzig, Springer-Verlag, Berlin*, 2003, 3-25.

[24] Mattern, F., Zürich E. Wireless Future: Ubiquitous Computing. – *Proceedings of Wireless Congress*, 2004.

[25] Maturana, D., Mery, D., Soto, A. Face Recognition with Local Binary Patterns, Spatial Pyramid Histograms and Naive Bayes Nearest Neighbor classification. – *International Conference of the Chilean Computer Science Society, SCCC,* 2009.

[26] Myung, J.C., Torralba, A., Willsky A.S. Context models and out-of-context objects. – *Pattern Recognition Letters*, 2012.

[27] Open Source Computer Vision Library. – *GitHub, Inc.* [*Online*] https://github.com/opencv/opencv (19.04.2017)

[28] Open Source Computer Vision Library. – *OpenCV team*. [*Online*] http://opencv.org/ (19.04.2017)

[29] Pascoe, J., Ryan, N.S., Morse, D.R. Human-Computer-Giraffe Interaction: HCI in the field. – *Workshop on Human Computer Interaction with Mobile Devices*, 1998, 182-196.

[30] pip documentation, Installation. [*Online*] https://pip.pypa.io/en/latest/installing/ (13.05.2017)

[31] PostgreSQL open source database. [*Online*] https://www.postgresql.org/ (05.05.2017)

[32] Python programming language. [*Online*] https://www.python.org/ (09.05.2017)

[33] Raspberry Pi. – *RASPBERRY PI FOUNDATION*. [*Online*] https://www.raspberrypi.org/ (19.04.2017)

[34] Raspbian. [*Online*] https://www.raspbian.org/ (13.05.2017)

[35] Saha, D., Mukherjee, A. Pervasive computing: a paradigm for the 21st century. – *IEEE Computer*, 2003, Volume 36, Issue 3, 25–31.

[36] Sarwat, M., Avery, J., Mokbel, M. RECATHON: A Middleware for Context-Aware Recommendation in Database Systems. – *IEEE International Conference on Mobile Data Management (MDM),* 2015, 54-63.

[37] Schilit, B.N., Adams, N., Want, R. Context-Aware Computing Applications. – *IEEE Proceedings of the Workshop on Mobile Computing Systems and Applications*, 1994, 85–90.

[38] Schilit, B.N., Theimer M.M. Disseminating Active Map Information to Mobile Hosts. – *IEEE Network*, 1994, Volume 8, Issue 5, 22–32.

[39] Schroff, F., Kalenichenko, D., Philbin, J. FaceNet: A Unified Embedding for Face Recognition and Clustering. – *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[40] Stefanidis, K., Pitoura, E., Vassiliadis, P. A Context-Aware Preference Database System. – *International Journal of Pervasive Computing and Communications*, 2005.

[41] Strang, T., Linnhoff-Popien, C. A Context Modeling Survey. – *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp,* 2004.

[42] Vanathi, B., Rhymend Uthariaraj V. Comparing and Choosing Appropriate Database for Storing Context in Context Aware System. – *Journal of Computer Science*, 2011, Volume 7, Issue 7, 1027.

[43] Weiser, M. The computer for the twenty-first century. – *Scientific American,* 1991, Vol 265, No 3, 94-104.

[44] Yi-Shi, L., Wai-Seng, N., Chun-We, L. A Comparison of Different Face Recognition Algorithms. – *University of North Carolina at Chapel Hill, Department of Computer Science, National Taiwan University*, 2009. [*Online*] http://cs.unc.edu/~chunwei/papers/2009pr_face_recog.pdf (10.04.2017)

[45] Zhao, R., Wang, J. Visualizing the research on pervasive and ubiquitous computing. – *Scientometrics*, 2011, Volume 86, Issue 3, 593–612.

[46] Özdil, A., Özbilen, M.M. A Survey on Comparison of Face Recognition Algorithms. – *IEEE 8th International Conference on Application of Information and Communication Technologies (AICT)*, 2014.

Appendix 1 – Online access to the project source code

Source code of the project was also uploaded to the Bitbucket and can be accessed online:

(Web) https://bitbucket.org/112996/loputoo_projekt_2017/src/

(Git) https://112996@bitbucket.org/112996/loputoo_projekt_2017.git