Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava 4

Emotion Log

Tímový projekt

Dokumentácia k inžinierskemu dielu

Bc. Michal Biroš
Bc. Tomáš Caban
Bc. Tomáš Kunka
Bc. Filip Staňo
Bc. Tomáš Lekeň
Bc. Milan Martinkovič
Bc. Bálint Szilva

Vedúci tímového projektu: Ing. Martin Labaj
Ak. rok: 2012/13

Obsah

1 Úvod

1.1 Účel dokumentu

Tento dokument predstavuje projektovú dokumentáciu k softvérovému produktu, ktorého cieľom je pozorovať správanie používateľa pri používaní počítača, analyzovať jeho emočný stav a navrhovať odporúčania na základe analyzovaného stavu. Tento projekt vzniká na predmete Tímový projekt pod vedením Ing. Martina Labaja.

Našim cieľom je obohatiť prebraté softvérové riešenie o sledovanie používateľa prostredníctvom kamery a mikrofónu a rozšíriť tak možnosti analyzovania jeho emočného stavu. Na základe rozpoznaného emočného stavu používateľa mu chceme odporúčať najvhodnejšie akcie v danom okamihu, napríklad prestávku alebo pohybovú aktivitu pre odstránenie nežiaducich stavov.

1.2 Motivácia

Práca na počítači je v dnešnej dobe každodennou realitou. Čoraz väčší dôraz sa kladie na dodržiavanie nesplniteľných termínov odovzdania prác, v dôsledku čoho môžu byť používatelia frustrovaní, unavený a robiť pri práci s počítačom čoraz viac chýb. V dôsledku týchto faktorov klesá produktivita práce, pričom používateľ si to vôbec nemusí uvedomovať. Preto je viac ako vhodné aplikovať systém, ktorý by bol schopný včas identifikovať zvyšujúce sa napätie, prípadne klesajúcu produktivitu práce používateľa a vhodným spôsobom sa pokúsiť o nápravu tohto stavu.

Náš systém je schopný rozoznať tieto riziká a vykonať zmenu (spustiť na pozadí obľúbenú hudbu, zmeniť farbu pozadia) alebo niečo odporučiť (prestávka, prechádzka). Vďaka tomuto bude používateľ pokojnejší a rýchlejšie a kvalitnejšie urobí svoju prácu. Zamestnávateľ tiež môže získať prehľad o produktivite svojich zamestnancov a kvalitnejší vytvorený produkt. V prípade softvérových firiem môže získavať napríklad aj informácie o vznikajúcich zdrojových kódoch z reakcií iných programátorov, ktorý tento kód prezerajú či upravujú. Tento systém však nájde uplatnenie nie len vo firmách pôsobiacich v oblasti informačných technológií, ale prakticky všade, pretože s počítačmi dnes už pracujú ľudia v najrôznejších odvetviach. Môžeme ho preto použiť pri rôznych činnostiach s počítačom, ktoré si vyžadujú určitú mieru sústredenia (od bežnej práce s textovým editorom až po implementovanie rozsiahlych informačných systémov).

1.3 Ciele projektu

Naším cieľom je vytvoriť softvérový systém, ktorý dokáže identifikovať nežiaduce zmeny v používateľovom emocionálnom stave. Môžeme menovať hlavne zlosť, frustráciu, či smútok, ale aj fyzické zmeny, ako je únava. Všetky tieto faktory môžu ovplyvniť produktivitu práce, a preto ich chceme včas rozoznať a vhodnými odporúčaniami obmedziť, alebo úplne odstrániť. Taktiež chceme na základe pozorovania používateľa a vonkajších vplyvov

ovplyvňujúcich jeho produktivitu alebo emocionálny stav vytvoriť model používateľa, na základe ktorého budeme schopný odporúčať najvhodnejšie akcie v danom okamihu.

Vďaka modelovaniu a uchovávaniu stavu používateľa môžeme toto riešenie použiť aj na rôzne iné súčasné problémy. Využitie môže nájsť napríklad aj pri získavaní spätnej väzby od používateľov k prehliadaným objektom(zdrojový kód, dokument, video), keď na základe reakcie viacerých používateľov o nich vieme zistiť rôzne informácie. Napríklad pri prezeraní videa používateľmi môžeme identifikovať, aké emócie v nich vyvoláva(pozitívne, negatívne). Riešenie chceme navrhnúť tak, aby bolo v budúcnosti možné jednoduché doplnenie ďalších aplikácií rozpoznaného stavu a tým rozšírenie možností nášho systému.

1.4 Výber implementačného jazyka, prostredia a technológií

Implementácia je rozdelená do 2 častí:

- serverová časť
- klientska časť

Táto architektúra bola zvolená preto, lebo každý používateľ si môže nainštalovať na svoj počítač klienta, ktorý komunikuje so serverom, ktorému odosiela zachytené dáta. Server následne modeluje používateľa a modely jednotlivých používateľov môže porovnávať.

1.4.1 Serverová časť

Jadro servera : Windows Server 2008
Databáza : SQL Server 2008 R2
Verziovanie a komunikácia s klientskou časťou: Team Foundation Server 2010 a IIS 7.0

Serverová časť obsahuje databázu, v ktorej sa uchovávajú spracovávané údaje, funkcionalitu pre vytváranie a spracovávanie modelov.

1.4.2 Klientska časť

Programovací jazyk – C #
Vývojové prostredie – Microsoft Visual Studio 2010

Klientská časť obsahuje logovač, ktorý zaznamenáva používateľa pri práci na počítači. Zachytené dáta následne odosiela na server, ktorý ich spracuje. Klient taktiež zobrazuje používateľovi jednotlivé odporúčania.

2 Prvý šprint – Audi

Hlavnou úlohou prvého šprintu bolo oboznámiť sa s problémovou oblasťou a s prebratým softvérovým riešením. Cieľom projektu je softvérový produkt, ktorý bude schopný na základe pozorovania používateľa kamerou modelovať jeho emocionálny stav a na jeho základe mu odporúčať rôzne akcie. V rámci šprintu sme identifikovali nasledovné príbehy:

- Používateľ je sledovaný pri práci na počítači
- Používateľ je modelovaný
- Modely sú rozpoznávané
- Používateľ dostane odporúčanie

2.1 Používateľ je sledovaný pri práci na počítači

Ako používateľ som sledovaný pri práci na počítači pomocou kamery a mikrofónu.

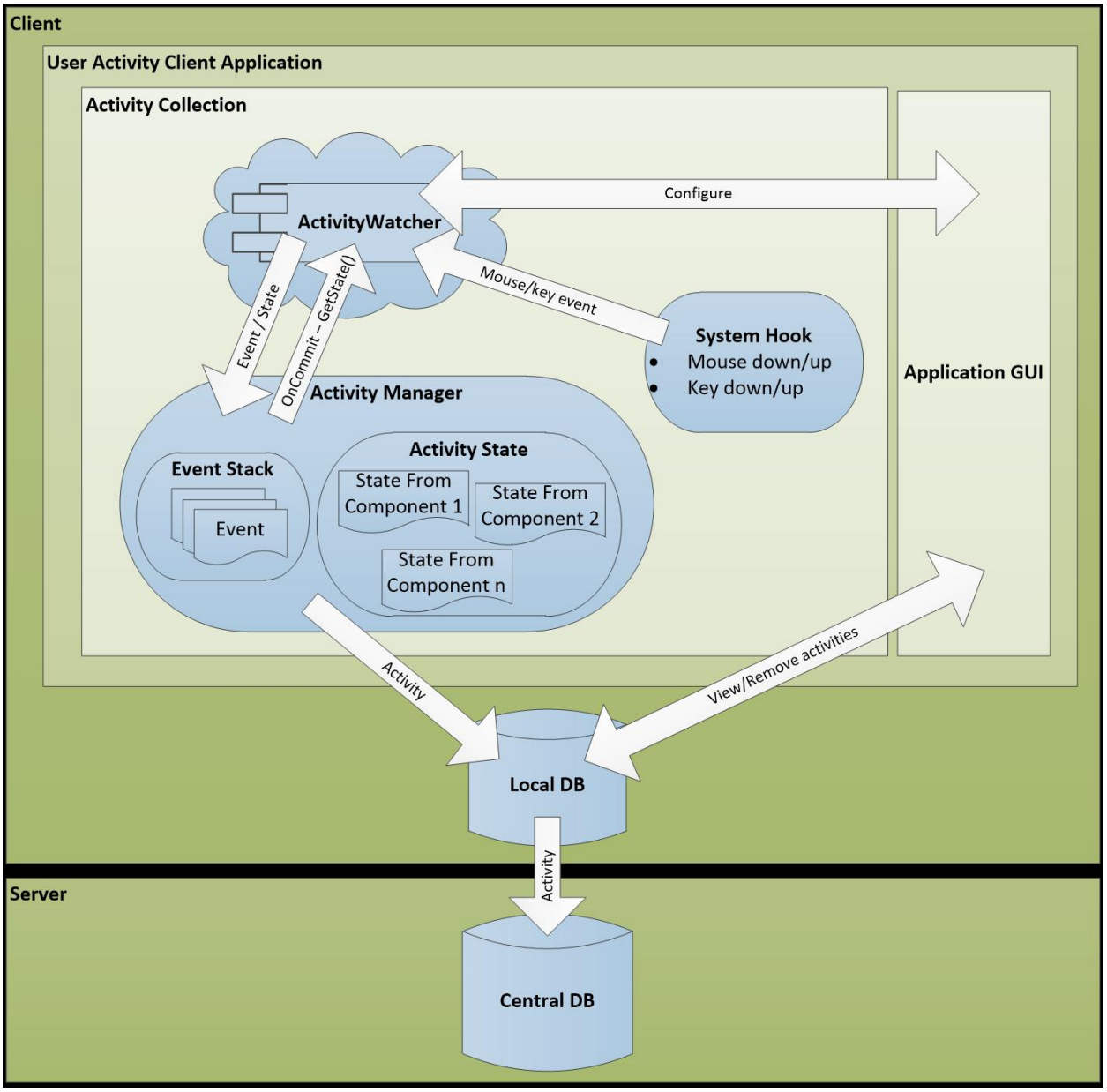
2.1.1 Analýza východzieho stavu

2.1.1.1 Architektúra systému

Aplikáciu budeme vytvárať v už existujúcom prostredí *logovača*, ktorý zaznamenáva aktivitu používateľa na počítači. Tieto údaje následne posiela do databázy na serveri.

Obrázok 1 znázorňuje architektúru tohto existujúceho systému. Tvoria ho dve časti – klientská a serverová časť. Klientskou časťou je aplikácia bežiaca na používateľovom počítači. Na obrázku vidíme, že na pozadí tejto aplikácie (Activity Collection) beží komponent Activity Watcher, ktorý prijíma údaje o akciách používateľa prostredníctvom *systémového háku* (System hook). Activity Watcher tieto údaje ďalej preposiela komponentu Activity Manager . V ňom sa z prijatých stavov a udalostí vytvárajú aktivity, ktoré sú ukladané do lokálnej databázy a potom ďalej na server, do centrálnej databázy.

Na obrázku je taktiež znázornené, že klientská aplikácia má grafické rozhranie, pomocou ktorého je možné konfigurovať nastavenia aplikácie. Pomocou tohto rozhrania sa dajú taktiež prezrieť alebo vymazať zaznamenané aktivity z lokálnej databázy.



Obrázok 1 Architektúra existujúceho system

2.1.1.2 Zdroje zbieraných údajov

Zdroje, z ktorých sú zbierané údaje:

- Operačný systém – vyťaženie procesora, obsadenie operačnej pamäte
- MS Visual Studio 2012 – operácie nad projektom, operácie nad dokumentom, aktuálne otvorené okno, pozíciu v kóde
- MS Office 2010 OneNote – activity v otvorených poznámkových blokov
- Spustené aplikácie a ich okná
- Klávesnica – stlačené klávesy
- Myš – pohyb myši po obrazovke
- Webový prehliadač (Firefox) – otvorené web stránky v jednotlivých kartách, operácie nad záložkami
- MS Lync 2010 – stav používateľa (online, offline, zaneprázdnenosť)

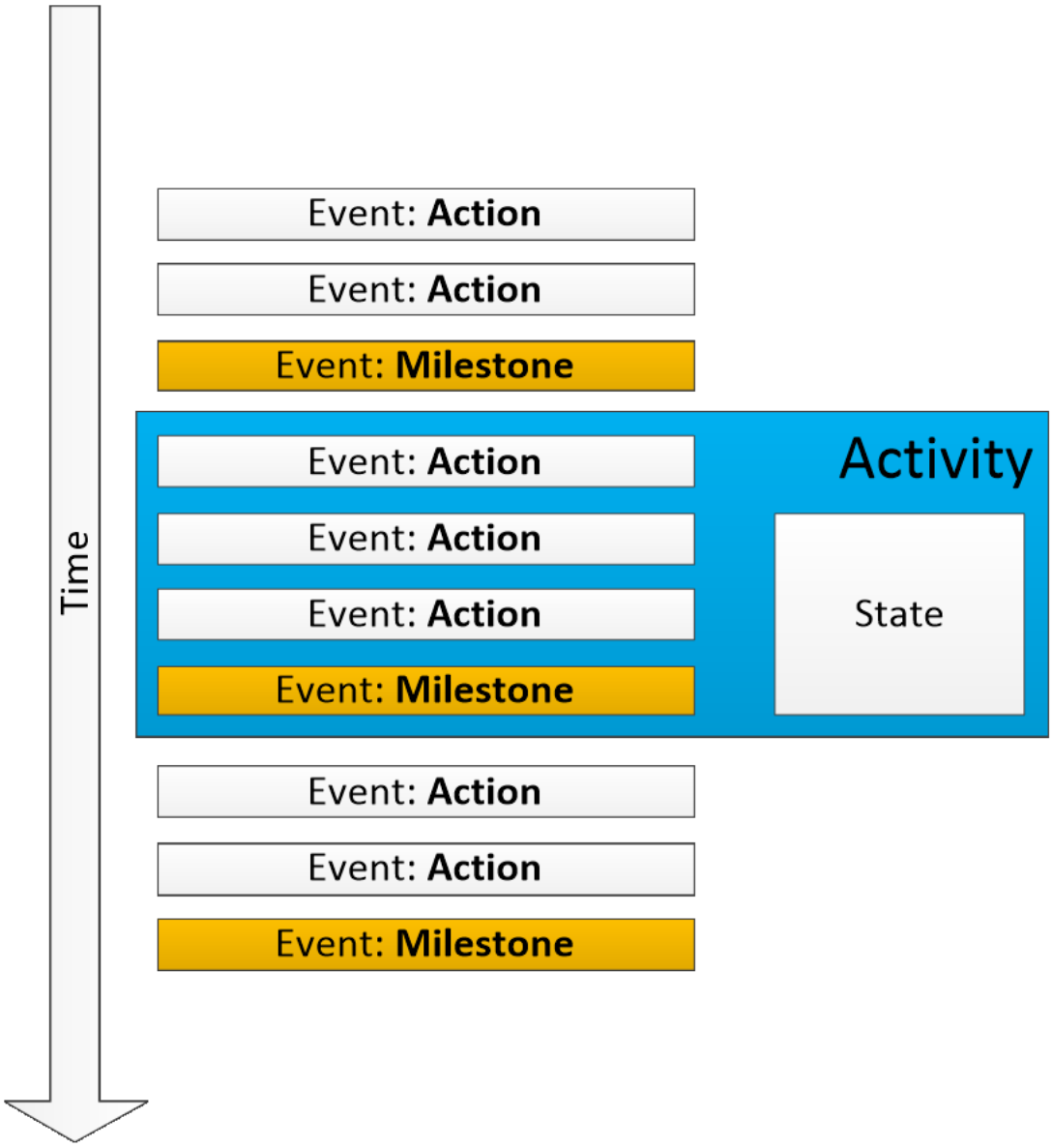
2.1.1.3 Aktivity

Údaje sú ukladané do databázy vo forme tzv. aktivít. Jedna aktivita obsahuje časovo usporiadané akcie a stavy akcií, ktoré nastali počas priebehu tejto aktivity.

```
<?xml version="1.0" encoding="utf-16"?>
<Activity              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"             StartTime="2012-11-
13T02:42:20.3489129Z"                 EndTime="2012-11-13T02:42:21.3989094Z"
Workstation="REXXAR8"      ActivityId="96ff3b03-004d-4ea4-983e-4acce11296ee"
xmlns="http://url/ActivityService">
  <Events>
      <ApplicationFocusLostDto       IsMilestone="true"       Time="2012-11-
13T02:42:15.4215973Z">
            <NewApplication       Pid="5440"       ApplicationName="firefox"
RunId="8f5c9dae-bfbf-482a-8e41-a85e7019f926" Hwnd="1573812" />
      </ApplicationFocusLostDto>
      ...
  </Events>
  <States>
      <RunningApplicationsListDto>
            <Items>
            ...
            </Items>
      </RunningApplicationsListDto>
      <KeyboardStateDto>
            <Data>
            ...
            </Data>
      </KeyboardStateDto>
      ...
  </States>
</Activity>
```

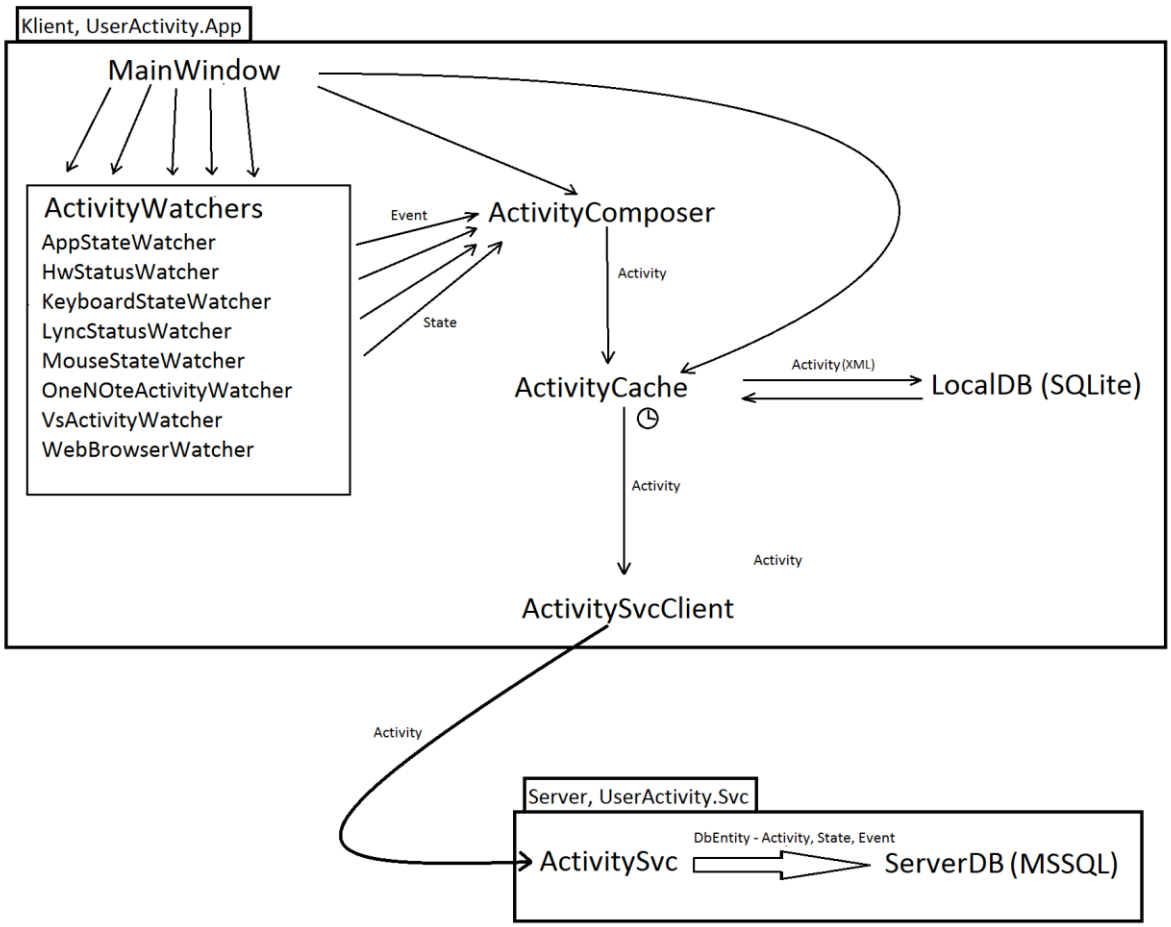
Zdrojový kód 1 Príklad aktivity zapísanej v XML

Obrázok 2 znázorňuje, ako sa vytvárajú aktivity v čase. Systém zaznamenáva akcie z jednotlivých *watcherov*. Ak je zaznamenaná akcia s atribútom *milestone*, aktivita je uzavretá a uložená do lokálnej databázy. Následne sa vytvorí nová prázdna aktivita.



Obrázok 2 Aktivita obsahuje časovo usporiadané akcie. Aktivita je ukončená práve jednou akciou, ktorá je míľnikom.

2.1.1.4 Priebeh zaznamenávania údajov



Obrázok 3 Schéma priebehu zaznamenávania aktivít

Obrázok 3 znázorňuje priebeh zaznamenávania aktivít od spustenia programu až po uloženie do databázy na serveri.

MainWindow – počiatočný komponent. Na začiatku inicializuje inštancie jednotlivých *ActivityWatchers, ActivityComposer, ActivityCache*.

ActivityWatchers – priebežne zaznamenávajú jednotlivé aspekty práce na počítači. Implementujú interfejs *IActivityWatcher* . Ak nejaký *ActivityWatcher* zachytí udalosť (*Event*), odošle ju do triedy ActivityComposer.

ActivityComposer – zaznamenáva jednotlivé *Event* z *ActivityWatcher.* Ak zaznamená Event s atribútom isMilestone = true, zavolá metódu OnActivityFinishing() v každom *ActivityWatcher*. Táto metóda vracia stav sledovania (*State).*
ActivityComposer vytvorí z týchto údajov *Activity*, ktorú pošle triede *ActivityCache*.

ActivityCache – zaznamenáva *Activity* a ukladá do lokálnej databázy vo forme XML. Prevod triedy *Activity* do XML je vykonávaný pomocou triedy *XmlSerializer.*
ActivityCache v pravidelnom intervale vyberá aktuálne aktivity z lokálnej databázy a preposiela ich na server pomocou triedy *ActivitySvcClient.*

ActivitySvcClient – zabezpečuje spojenie klientskej aplikácie so službou na serveri (*ActivitySvc*). Pomocou metódy CommitActivity

ActivitySvc – služba, ktorá beží na serveri a prijíma *Activity*, odosielané komponentom z klientskej aplikácie.
Z jednotlivých aktivít typu Activity vyberie udalosti (Event) a stavy (State), ktoré uloží do databázy na serveri. Pred uložením sú tieto entity prekonvertované na databázové entity pomocou konvertorov (*Convertors*).

2.1.2 Analýza softvéru na sledovanie pohľadu

Sledovanie pohľadu je pre nás dôležité najmä z hľadiska možnosti určenia miery sústredenia pozorovaného používateľa. Dôležitými atribútmi sú jednak primeraná presnosť, nízka náročnosť na použitú kameru ale aj možnosť exportu údajov o požívateľovej tvári, získaných jeho sledovaním.

2.1.2.1 ITU Gaze tracker

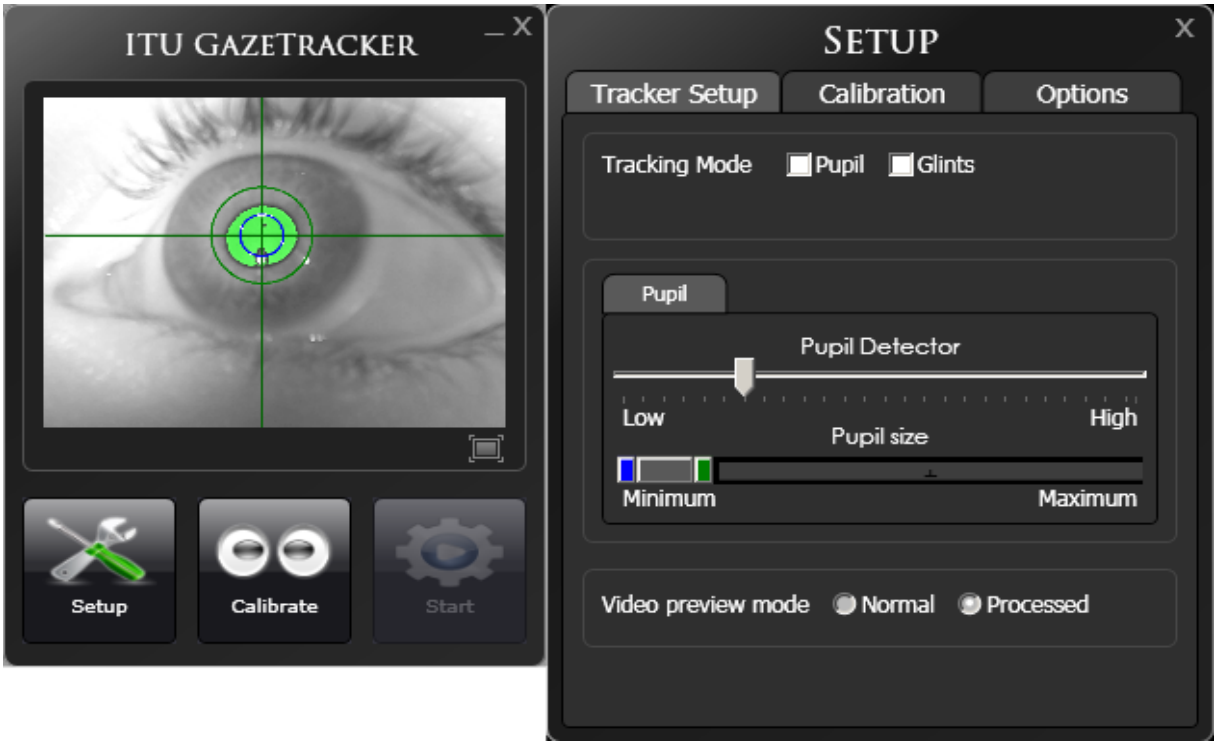
Jedná sa o nekomerčné riešenie. Spolupracuje s kamerami, ktoré obsahujú infračervené diódy ale aj s takými, ktoré ich nemajú.

Výhody:

- vhodná presnosť sledovania pohľadu
- možnosť exportovania údajov o sledovaní pohľadu

Nevýhody:

- nutná kalibrácia a to aj po miernej zmene polohy používateľa
- stand-alone riešenie bez otvoreného kódu
- potrebná konfigurácia kamery cez používateľské rozhranie
- pre maximálnu presnosť je potrebné použitie s kamerou, ktorá podporuje vysielanie infračerveného svetla



Obrázok 4 Ukážka používateľského rozhrania ITU Gaze tracker **Chyba! Nenašiel sa žiaden droj odkazov.**

2.1.2.2 Camera Mouse 2012

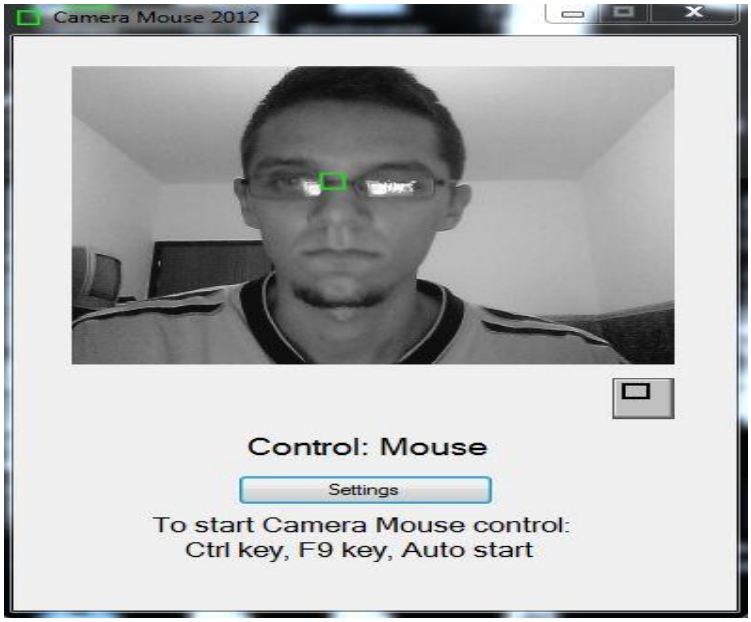
Taktiež nekomerčné riešenie, ktoré sa zameriava na ovládanie kurzora myši pomocou pohľadu.

Výhody:

- nie je potrebná zdĺhavá kalibrácia
- automatická detekcia pevného bodu na tvári, pomocou ktorého je zachytávané natočenie hlavy (zelený bod na obrázku predstavuje pevný bod, ktorý bol nájdený)
- vysoká presnosť

Nevýhody:

- je síce zadarmo ale zdrojový kód nie je k dispozícií
- žiadny export údajov



Obrázok 5 Ukážka Camera Mouse 2012

2.1.2.3 Track Eye

Výhody:

- open source
- bez kalibrácie

Nevýhody:

- nepresné
- pre zlepšenie presnosti je potrebná zložitá konfigurácia nastavení, po ktorej je presnosť na stále nízkej úrovni

2.1.2.4 Zhodnotenie analýzy

Analyzované boli aj ďalšie riešenia, tie však neuvádzame kvôli ich podstatnej miere negatívnych vlastností ako napríklad nestabilnosť, nepresnosť ale aj nízka kompatibilita s najpoužívanejšími platformami. Na základe analyzovaných riešení sme sa rozhodli funkcionalitu sledovania pohľadu nezahrnúť do nášho projektu. Hlavnými dôvodmi boli uzavretosť kódu daného softvéru, poprípade nízka úroveň riešení s otvoreným kódom. Ďalším dôležitým negatívnym faktorom bola pri väčšine riešení nutnosť modifikácie rôznych nastavení ale aj kalibrovanie sledovania. Tieto vlastnosti prispievali k nízkej úrovni používateľsky príjemného prostredia, ktoré by mohlo mať za následok frustráciu používateľa, čo je pravý opak toho o čo sa v našom projekte snažíme.
Na základe analýzy sme však napríklad identifikovali náležitosti, ktoré sú predpokladom pre zlepšenie presnosti a kvality sledovania používateľa. Jedná sa napríklad o vhodné

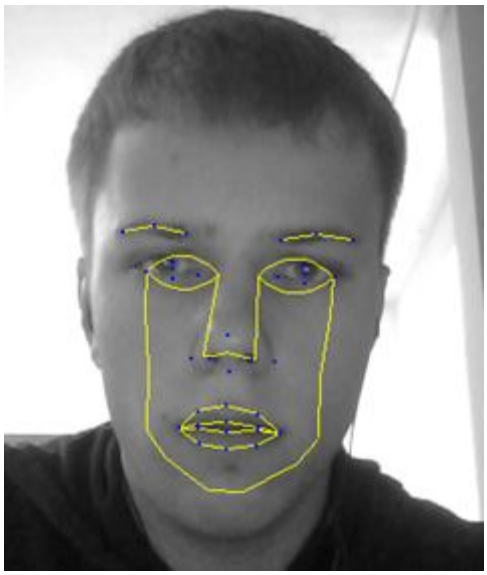
umiestnenie kamery. Lepšie výsledky boli dosahované pri externých kamerách, ktoré boli umiestnené na stole pod úrovňou tváre používateľa na rozdiel od vstavaných kamier v obrazovke. Ďalšími aspektmi sú vhodné osvetlenie či kvalita záznamu, ktorý kamera produkuje.

2.1.3 Analýza existujúcich riešení na rozpoznávanie emócií

2.1.3.1 FaceAPI

Rozoznáva základné črty tváre priamo z webkamery a v pomerne nízkom časovom intervale identifikuje jednotlivé časti tváre a vyznačí ich.
Vyznačuje:

- 3 body obočia (definuje tvar)
- 4 body nosa
- 5 bodov očí
- 14 bodov úst

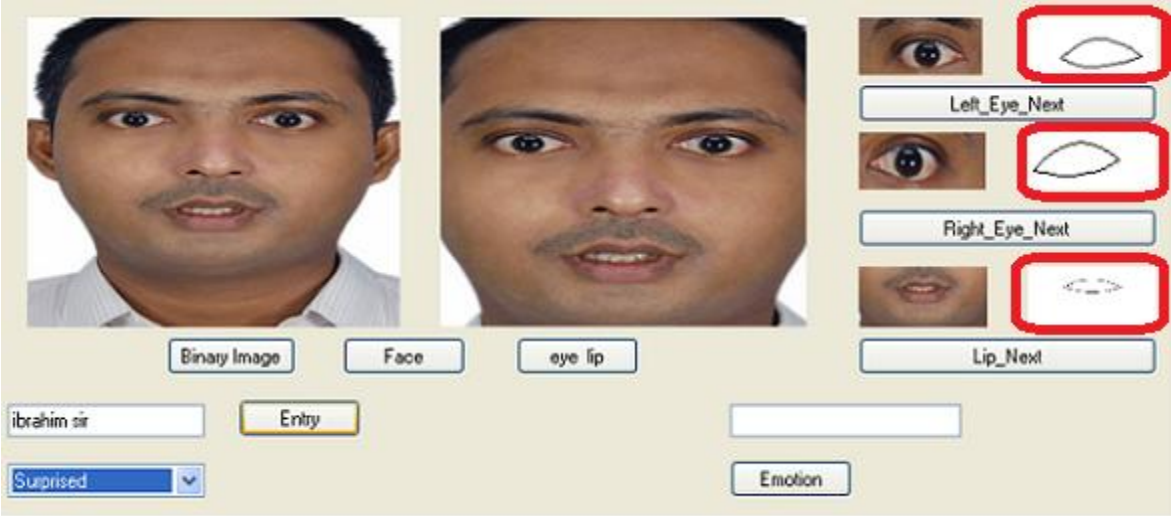


Obrázok 6 Spracovanie tváre pomocou FaceAPI

Nekomerčná licencia neobsahuje kompletné služby, vývojárska licencia obsahuje kvalitné výstupy aj plnohodnotné spracovanie tváre – treba o ňu požiadať. Optimálne je získať výstup súradníc pre jednotlivé časti tváre.

2.1.3.2 Human Emotion Detection

HED je projekt bangladéšskeho študenta, vyvíjaný pod CPOL licenciou. Rozoznáva emóciu podľa tvaru očí a úst. Pomocou prevodu obrazu tváre do binárneho prostredia segmentuje jednotlivé časti tváre(oči a pery), z ktorých následne odvádza emóciu. Je vyvíjaný v C# pomocou .NET frameworku.



Obrázok 7 GUI aplikácie Human Emotion Detection

Na Obrázok 7 je červenou farbou zvýraznený využiteľný výstup tvaru očí a úst.

2.1.3.3 Face.com

Tento projekt umožňuje rozpoznanie viacerých tvárí z fotografie a identifikáciu jednotlivých bodov a charakteristík nálady, či veku.
Vyznačuje:

- 1 bod nosa
- 2 body očí
- 3 body úst

Attributes:	
age_est:	45 (47%)
age_min:	39 (47%)
age_max:	53 (47%)
face:	true (99%)
gender:	male (91%)
glasses:	false (95%)
lips:	sealed (65%)
mood:	happy (63%)
smiling:	true (74%)
Rotations:	
roll:	-0.64°
yaw:	-1.4°
pitch:	-6.08°

Obrázok 8 Výstupy aplikácie Face.com

Pokúša sa o určenie:

- Veku
- Pohlavia
- Otvorenia pier
- Nálady
- Úsmevu
- Okuliarov

Ku všetkým spomínaným hodnotám pridelí pravdepodobnosť.
Verejné API na serveri bolo uzavreté v októbri 2012, avšak použiteľné knižnice sú na stiahnutie vo viacerých jazykoch (C#, JAVA, PHP, Python, RUBY, JS ...)

2.1.4 Analýza detekovateľných emočných stavov z tváre

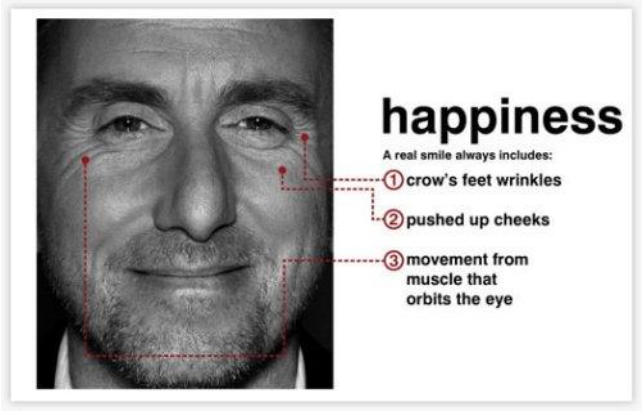
Existujúce rozpoznateľné emočné stavy:

- Radosť
- Smútok
- Hnev
- Strach
- Prekvapenie
- Znechutenie
- Pohŕdanie

2.1.4.1 Radosť

Špecifické znaky:

1. Roztiahnuté a vydvihnuté kútiky úst
2. Vydvihnuté líca
3. Zvraštené vonkajšie očné svaly

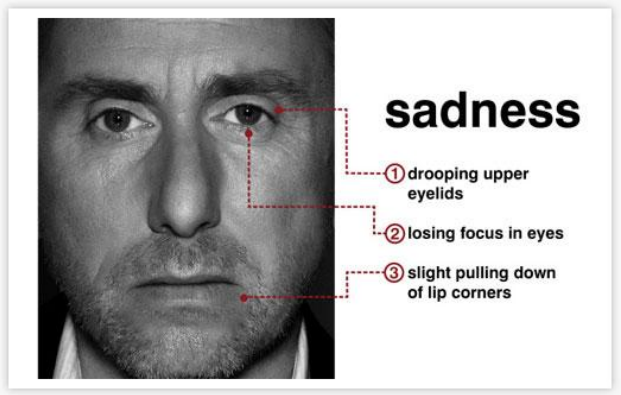


Obrázok 9 Radosť

2.1.4.2 Smútok

Špecifické znaky:

1. Mierne padnuté kútiky úst
2. Padnuté horné viečko

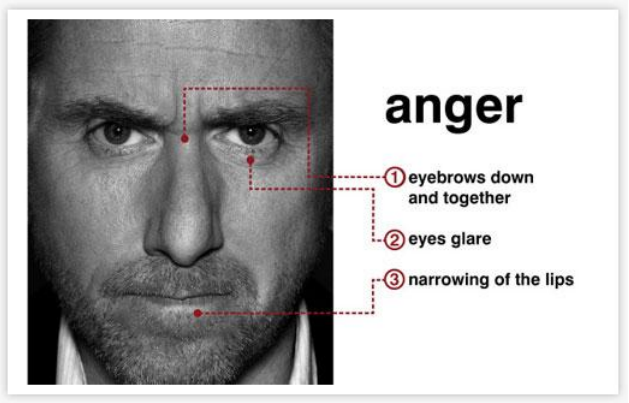


Obrázok 10 Smútok

2.1.4.3 Hnev

Špecifické znaky:

1. Zvraštené obočie – bližšie k očiam a k sebe navzájom
2. Maximálne otvorené spodné viečko
3. Vyrovnané a stiahnuté pery

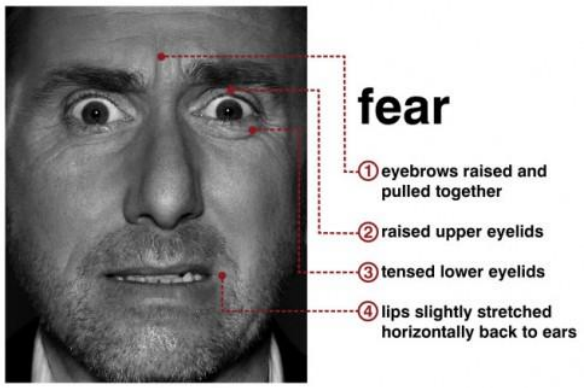


Obrázok 11 Hnev

2.1.4.4 Strach

Špecifické znaky:

1. Zdvihnuté obočie, bližšie k sebe navzájom
2. Maximálne otvorené horné viečko
3. Napnuté dolné viečko
4. Pootvorené ústa, natiahnuté pery

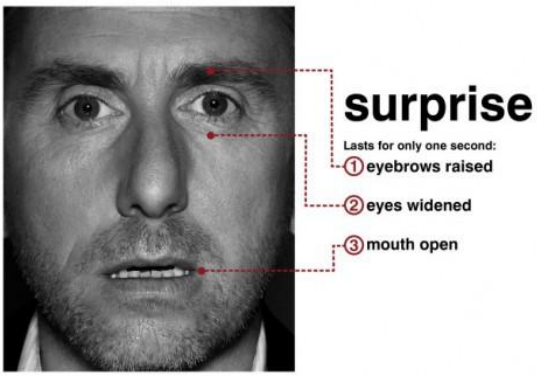


Obrázok 12 Strach

2.1.4.5 Prekvapenie

Špecifické znaky:

1. Zdvihnuté obočie
2. Pootvorené ústa
3. Maximálne otvorené oči

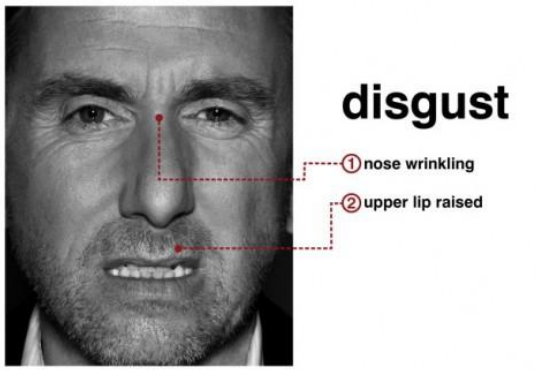


Obrázok 13 Prekvapenie

2.1.4.6 Znechutenie

Špecifické znaky:

1. Vydvihnutie hornej pery
2. Vrásky medzi očami

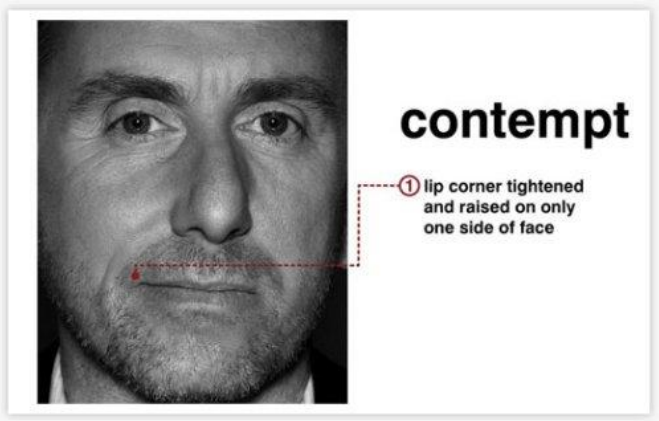


Obrázok 14 Znechutenie

2.1.4.7 Pohŕdanie

Špecifické znaky:

1. Vydvihnutá pera na jednej strane – polovičný úsmev



Obrázok 15 Pohŕdanie

2.1.5 Analýza možností využitia mikrofónu pre získavanie informácií o emóciách, identite používateľa a prítomnosti používateľa

2.1.5.1 Emócie

Získavanie emócií z hlasu používateľa je najmä predmetom výskumu. Existuje množstvo vedeckých článkov, ktoré sa zaoberajú touto problematikou. Nepodarilo sa nájsť žiadny komerčný nástroj, ktorý by umožňoval získavanie emócií z hlasu. Analyzovali sme dva akademické nástroje, ktoré sa snažia o získavanie emócií z hlasu.

2.1.5.2 OpenEAR

http://sourceforge.net/projects/openart/
Nástroj napísaný v jazyku C++. Umožňuje získavanie emócií priamo z mikrofónom zachyteného zvuku, alebo zo zvukových súborov. Je ho možné použiť ako samostatný nástroj (príkazový riadok) alebo ako knižnicu. Používa 6 modelov na vyhodnocovanie emócií, z ktorých väčšina pracuje na princípe, že majú niekoľko stavov (strach, hnev, šťastie, smútok, nechuť, ...) a vyjadrujú percentuálnu pravdepodobnosť pre jednotlivé stavy.
Na vyhodnocovanie emócií sa používajú vytvorené databázy, ale je možné vytvoriť aj vlastné a tak trénovať nástroj.
Jedná sa o open-source nástroj. Nevýhodou je veľmi slabá a nekonzistentná databáza. Na projekte sa nepracuje od roku 2009.

2.1.5.3 EmoVoice - Real-time emotion recognition from speech

http://www.informatik.uni-augsburg.de/de/lehrstuehle/hcm/projects/tools/emovoice/

Rovnako akademický nástroj napísaný v jazyku C++. Umožňuje získavanie emócií z hlasu používateľa. Nástroj posudzuje emócie pozitívna a negatívna a vracia percentuálnu pravdepodobnosť pre danú emóciu. Je ho možné použiť ako samostatný nástroj. Poskytuje aj grafické používateľské rozhranie, ktoré umožňuje aj trénovanie programu pre používateľa.
EmoVoice bol integrovaný do frameworku **Social Signal Interpration (SSI)**, ktorý poskytuje aj ďalšie funkcie, napr. rozoznávanie emócií z obrazu. SSI sa dá použiť aj ako C++ knižnica. SSI tiež poskytuje špeciálny XML editor s množstvom funkcií pre prácu s nástrojom.
Je to voľne dostupný nástroj. Jeho veľkou nevýhodou je rovnako ako pri OpenEAR nedostupná dokumentácia, k dispozícii je iba jeden návod.

2.1.5.4 Identita používateľa

Pri hľadaní možností využitia hlasu získavaného pomocou mikrofónu na identifikáciu používateľa je potrebné rozoznávať dve základné kategórie:

- Speech recognition – rozpoznanie reči
- Speaker recognition – rozpoznanie hovoriaceho

Na rozoznávanie rečí existuje množstvo nástrojov a knižníc, za všetky môžeme spomenúť PRAAT (http://www.fon.hum.uva.nl/praat/), naozaj rozsiahly projekt na analýzu reči. Niektoré algoritmy sú využívané aj už spomínaným EmoVoice nástrojom. Tieto nástroje či knižnice sú však pre náš projekt a pre naše potreby len veľmi málo využiteľné.
Táto oblasť je najmä cieľom výskumu a je veľmi komplexná. Existujú teoretické modely na rozoznávanie hovoriaceho či na základe slov, viet, ale aj hlasu. Nepodarilo sa však nájsť žiadny voľne dostupný nástroj, ktorý by sa o to aspoň snažil. Existuje niekoľko vedeckých článkov, kde aj implementujú takéto systémy, žiadny však nie je dostupný.

2.1.5.5 Prítomnosť používateľa

V našom projekte sa uvažuje aj o získavaní informácie o prítomnosti používateľa pri počítači akusticky – pomocou zvuku. Úlohou bolo analyzovať dostupné nástroje, ktoré by toto umožňovali.

2.1.5.6 Sonar Power Manager

Program, ktorý umožňuje zistiť prítomnosť používateľa na základe vysielania ultrazvuku (20 kHz). Program vypína obrazovku v prípade neprítomnosti. Program je voľne dostupný aj so zdrojovým kódom. Tento prístup nefunguje pre všetky počítače, niektoré nedokážu vysielať ultrazvuk. Na 3 testovaných počítačoch toto riešenie nefungovalo.

2.2 Modely sú rozpoznávané

2.2.1 Analýza možností strojového učenia

Strojové učenie môžeme deliť podľa viacerých kritérií. Jedno z možných delení je nasledovné:

- Induktívne – z konkrétnych príkladov generuje všeobecnejšiu znalosť
- Deduktívne – zo všeobecnej znalosti dedukuje menej všeobecnú, lepšie prispôsobenú na riešenie konkrétneho problému

Ďalšie možné delenie je podľa spôsobu získavania trénovacích príkladov:

- Online – získavajú trénovanie príklady postupne. Toto zodpovedá inkrementálnemu učeniu
- Offline – disponujú všetkými trénovacími príkladmi súčasne. Zodpovedá neinkrementálnemu učeniu

V našom prípade ide o offline učenie, keď potrebujeme najprv určitú štartovaciu množinu príkladov, na ktorých budeme môcť trénovať náš učiaci algoritmus. Postupne však budeme môcť použiť znaky online učenia, pretože by sme mali byť schopný učiť algoritmus aj z nových, postupne prichádzajúcich trénovacích príkladov.

Podľa stupňa kontroly rozlišujeme učenie na:

- Kontrolované – disponuje spätnou väzbou oúspešnosti učenia
- Nekontrolované – neexistuje spätná väzba

V našom prípade by bolo vhodné, ak by sme dostávali spätnú väzbu od používateľa, pretože by to pomohlo vylepšiť algoritmus rozpoznávania emočných stavov. Musíme však dbať na to, aby sme nezaťažovali používateľa zbytočným množstvom požiadaviek na spätnú väzbu.

Náš problém zodpovedá učeniu s učiteľom, teda priraďovaniu tried k určitým objektom. Na začiatku procesu potrebujeme trénovaciu sadu príkladov, na základe ktorých sa vytvorí klasifikátor, ktorý bude následne priraďovať vstupné dáta určeným triedam, v našom prípade emočným stavom.

Algoritmus učenia v našom prípade bude vhodné navrhnúť ako rozhodovací strom, ktorého listy budú predstavovať jednotlivé emočné stavy. Tento strom sa vytvorí na základe analýzy vzorov so známym stavom. Pri rozhodovaní sa bude postupovať od koreňa stromu, pričom v každom uzle sa testuje hodnota určitého znaku, až kým sa testovací príklad nedostane do niektorého z listov.

2.3 Používateľ dostane odporúčanie

2.3.1 Analýza odporúčaní pre jednotlivé emočné stavy

Cieľom analýzy je nájsť vhodné odporúčania aktivít, ktoré by mohli zmeniť náladu používateľa systém. Samozrejme ide o zmenu z negatívneho stavu do pozitívneho. Odporučené aktivity by mali byť vhodné a použiteľne v kancelárskom pracovnom prostredí. Identifikovali sme tieto základné negatívne emócie, s ktorými sa môžeme stretnúť v práci:

- frustrácia
- obavy/nervozita
- hnev
- niečo sa nám nepáči
- sklamanie/smútok

Môžeme rozoznávať aj pozitívne stavy, ale tie u používateľa meniť nebudeme.

2.3.1.1 Spôsoby zmeny nálady

2.3.1.1.1 Fyzická relaxácia

Základnou myšlienkou je zabezpečenie uvoľnenie napätia a stresu. Medzi svalmi a náladou bolo dokázané spojenie. Dokonca keď človek používa svalstvo na ústach – usmeje sa jeho nálada sa zmení. Toto je zaujímavé zistenie, keďže prirodzene chápeme opak – ak je človek šťastný usmeje sa.

- Dýchanie:
  - frekvencia dýchania sa drasticky mení pri rôznych emočných stavoch(keď sme nahnevaní dýchame rýchlejšie, pri strachu zadržiavame dych)
  - ak sa dýchanie naučíme ovládať, dokážeme ovládnuť svoje emócie
- Svalová relaxácia:
  1. posadiť sa pohodlne
  2. zavrieť oči
  3. začať relaxovať svaly – od nôh smerom nahor
  4. sústreďte svoju pozornosť na dýchanie
  5. zhlboka dýchať a počítať

2.3.1.1.2 Zmena zamerania

Ak človek vykonáva určitú činnosť v neustále dookola, znižuje sa jeho pozornosť a jeho nálada sa mení na negatívnu, stáva frustrovaný. Ak sa po tejto činnosti nedostaví žiadaný výsledok, nastáva u ľudí sklamanie a smútok. Preto je potrebné občas zmeniť zameranie, ktorému sa ľudia v práci venujú. Návrhy odporúčaní pre zmenu zamerania sú:

- Hra:
    - pri hraní hier sa zameranie veľmi rýchlo mení, myseľ človeka sa sústreďuje na dosiahnutie dobrého výsledku v hre
    - hra by mala byť dostatočne náročná a poskytovať dobrý pocit z víťazstva – mozog človeka odmení dobrým pocitom ak splní nejakú úlohu
- Socializovanie:
    - ľudia prirodzene žijú v skupinách, preto ak počas dňa človek pracuje sám je dobré ak svoje problémy preberie s niekým iným
    - socializovanie sa dá spojiť aj s hrou – je vhodné mať v kancelárskom prostredí umiestnenú nejakú hru(šípky, malý basketbalový kôš)

2.3.1.1.3 Zvýšenie hladiny hormónov

Hormóny sú zlúčeniny, ktoré v tele slúžia ako chemický poslovia medzi bunkami. Sú produkované v ľudskom tela a riadia priebeh a vzájomnú koordináciu reakcií v organizme. Medzi dôležité hormóny, ktorými je možné ovplyvniť náladu sú:

- Endorfin:
    - hormón šťastia – vyplavuje sa v mozgu a spôsobuje dobrú náladu, pocit šťastia a tlmí bolesť
    - vyplavuje pri svalovej námahe, športovej činnosti
    - **Ako zvýšiť hladinu:** prechádzka, pálivé jedlá, čokoláda
- Serotonin:
    - hormón, ktorého nedostatok spôsobuje depresie a agresivitu
    - jeho hladinu je možné ovplyvniť jedlom alebo drogami
    - **Ako zvýšiť hladinu:** papáje, banány, datle

2.3.1.1.4 Hudba

Hudba dokáže podľa výskumov taktiež zmeniť emocionálny stav. Vytvorí buď úplne nový emočný stav, alebo v poslucháčovi vyvolá emóciu zo spomienok(toto je typické pre hudbu z pohrebov, svadieb). Odporúčanie skladieb by mohlo byť problematické, keďže hudobný vkus je subjektívny. Môžeme sa však pokúsiť vytvoriť akési univerzálne hudobné skupiny. Určite by bolo vhodné prehrávanie motivačnej hudby, ak je používateľ frustrovaný, alebo upokojujúcej ak je nahnevaný.

2.3.1.1.5 Farba pozadia

Niekoľko starobylých kultúr, vrátane Egypta či Číny praktikovalo liečenie farbami. Dnes sa tiež používa v alternatívnej medicíne. Niektorí psychológovia sú voči terapii farbami skeptickí a považujú ju za preceňovanú. V našom prípade by sme sa mohli pokúsiť zmeniť náladu používateľa zmenou farby pozadia na počítači. Bolo však dokázané, že tento efekt je len dočasný. Vnímanie farieb je subjektívne, ale existujú farby, ktoré sú univerzálne:

- Červená, oranžová a žltá sú vnímané ako teplé farby a evokujú emócie tepla, komfortu až hnevu.
- Modrá, purpurová a zelená sú vnímané naopak ako chladné a evokujú pocity pokoja, smútku a ľahostajnosti.

2.4 Zhrnutie šprintu

Cieľom šprintu bolo oboznámiť sa s prebratým softvérovým riešením a analyzovať možné prístupy sledovania používateľa a zisťovania jeho emočného stavu pomocou kamery a mikrofónu. Taktiež sme analyzovali odporúčania pre emocionálne stavy. Z analyzovaných prístupov sme vybrali tie, ktoré pre sú pre potreby nášho projektu vhodné. V oblasti analýzy pomocou kamery je to projekt Human Emotion Detection a pre spracovanie pomocou mikrofónu sme vybrali projekt openEAR. V šprinte sme riešili aj niektoré detaily softvérového návrhu projektu, ako sú potrebné triedy, ktoré musia byť implementované.

3 Druhý šprint – Bentley

Hlavným cieľom druhého šprintu bolo aplikovať analyzované riešenia, konkrétne získavanie a spracovanie obrazu z kamery. V tejto oblasti sme projekt HED nahradili projektom Luxand, ktorý viac vyhovuje naším potrebám. Pracovali sme aj na možnosti využiť získavanie a spracovanie zvuku v projekte a taktiež na úlohách súvisiacich s podporným softvérom. V oblasti spracovania zvuku sme sa rozhodli nahradiť projekt openEAR, keďže tento projekt už nie je vyvíjaný a jeho dokumentácia je nekonzistentná. Namiesto neho sme sa rozhodli použiť knižnicu openSMILE, ktorú používa aj projekt openEAR. Dôležitou úlohou tohto šprintu bolo aj zostaviť dokumentáciu k riadeniu projektu a k inžinierskemu dielu. V rámci šprintu sme riešili nasledovné úlohy:

- Príprava údajov na odosielanie
- Reprezentácia údajov z kamery
- Skompilovanie programu na spracovanie zvuku
- Tvorba dokumentácie
- Konfigurácia TFS a Webu
- Spracovať obraz pomocou Luxand
- Získavať obraz z kamery

3.1 Reprezentácia údajov z kamery

Cieľom tejto úlohy bolo navrhnúť ako reprezentovať údaje o používateľovi získané pomocou kamery. Bolo potrebné navrhnúť aké dáta budú posielané na server, ako budú reprezentované jednotlivé časti tváre, prípadne emócie.

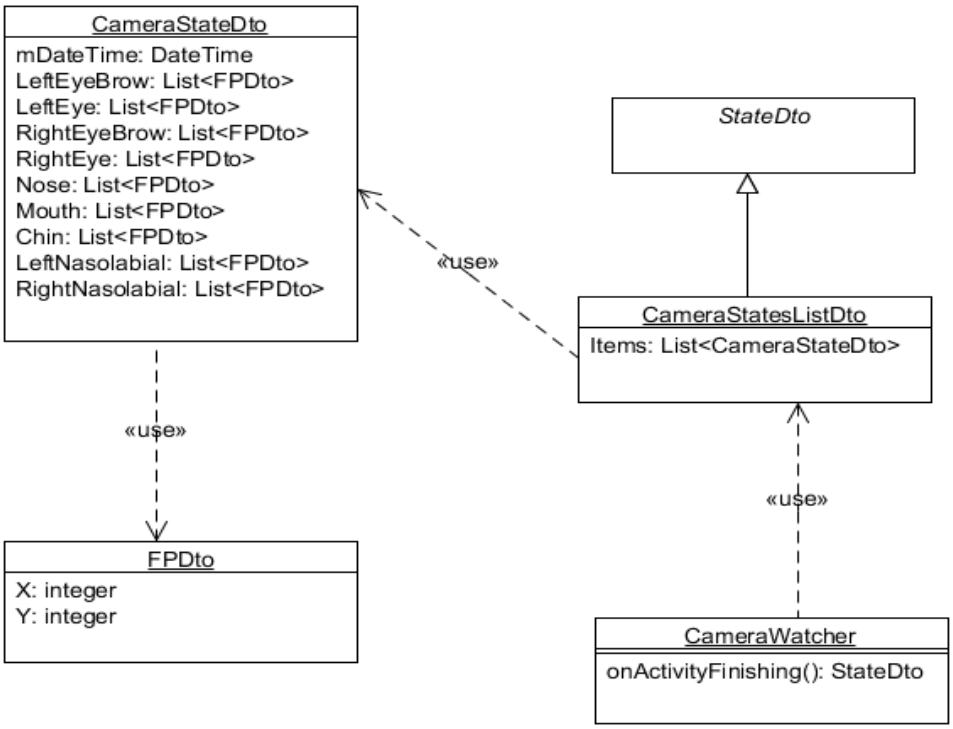
Výsledná podoba dát vyplynula z použitého nástroja Luxand SDK. Luxand FaceSDK rozoznáva 66 bodov tvare, ktoré reprezentuje dvoma číslami: x-ová súradnica a y-ová súradnica. Jednotlivé body boli rozdelené do zoznamov podľa toho akú časť tvare opisujú (ľavé oko, pravé oko, ústa, nos, brada, ľavé obočie, pravé obočie). Poradie bodov pre jednotlivé časti tvare sú podľa Tabuľka 1. Vzhľadom na to, že na rozoznávanie emócií používame, resp. plánujeme používať viacero metód, rozhodli sme sa využiť všetky body a tak sa na server posiela vždy všetkých 66 bodov.

3.2 Príprava údajov na odosielanie

Táto úloha zahŕňala implementáciu vytvárania XML z logovaných dát kamery podľa navrhnutej podoby reprezentácie dát, ich ukladanie do lokálnej SQLite databázy a odosielanie na server.

3.2.1 Návrh

Bolo potrebné navrhnúť a implementovať dátové triedy, ktoré predstavujú jednotlivé XML elementy. Každý koreňový XML element, ktorý obsahuje dáta a ktorý je vrátený metódou *onActivityFinishing*() pri skončení aktivity musí dediť od triedy `Svc.Interfaces.StateDto`.
V našom prípade sa počas jednej aktivity generuje niekoľko stavov, takže bolo potrebné vytvoriť ako hlavnú triedu zoznam všetkých stavov kamery počas aktivity. Hlavnou triedou je teda **CameraStatesListDto**, ktorá obsahuje zoznam objektov typu **CameraStateDto**. Jednotlivé body sú reprezentované pomocou triedy **FPDto**. Obrázok 16 zobrazuje UML diagram tried.



Obrázok 16 UML diagram tried

3.2.2 Implementácia

Samotná implementácia sa nachádza v súbore **CameraStates.cs** v projekte **UserActivity.Svc.Interfaces**. Pri implementácií bolo potrebné definovať že sa jedná o XML elementy a atribúty.

```
public class FPDto
    {
        [XmlAttribute]
        public int X { get; set; }
        [XmlAttribute]
        public int Y { get; set; }
```

```
        [XmlAttribute]
        public double rX { get; set; }
        [XmlAttribute]
        public double rY { get; set; }
    }

[Export(typeof(StateDto))]
    [XmlRoot(Namespace = "http://www.gratex.com/PerConIk/IActivitySvc")]
    public class CameraStatesListDto : StateDto
    {
        private List<CameraStateDto> mStates;

        public List<CameraStateDto> Items;

    }
```

Zdrojový kód 2 Implementácia triedy FPDto

Samotná inicializácia týchto tried sa vykonáva v triede **CameraWatcher**. Dáta získané pomocou Luxand SDK sú skonvertované na **CameraStateDto** pomocou metódy

```
private void createAllStates(FSDK.TPoint[] facialFeatures, FSDK.CImage image)
```

V tejto metóde sa skonvertujú body z poľa bodov *FSDK.TPoint* na jeden objekt CameraStateDto a pridá sa do zoznamu stavov pre danú aktivitu, ktorý vráti *onActivityFinishing*() metóda po skončení aktivity.
V **ActivityCache** sa zobrazujú dáta z kamery v nasledujúcom XML formáte:

```
<CameraStatesListDto>
      <CameraStateDto mDateTime="2012-11-11T23:06:08.7835375+01:00">
            <LeftEye>
                  <FPDto X="350" Y="166" />
                  <FPDto X="365" Y="168" />
                  <FPDto X="333" Y="169" />
                  <FPDto X="358" Y="171" />
                  <FPDto X="349" Y="173" />
                  <FPDto X="341" Y="170" />
                  <FPDto X="340" Y="164" />
                  <FPDto X="349" Y="162" />
                  <FPDto X="358" Y="161" />
                  <FPDto X="346" Y="167" />
                  <FPDto X="358" Y="167" />
            </LeftEye>
            <RightEye>
                  ....
            </RightEye>
            ...
      </CameraStateDto>
      <CameraStateDto mDateTime="2012-11-11T23:06:09.7835375+01:00">
            ...
      </CameraStateDto>
</CameraStatesListDto>
```

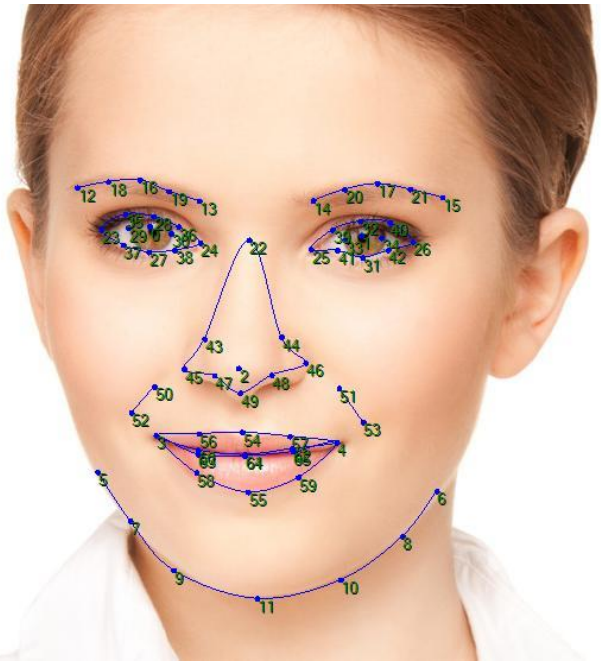
Zdrojový kód 3 XML formát zobrazovaných dát

3.3 Spracovať obraz pomocou Luxand

Tvár používateľa je sledovaná pomocou kamery a body potrebné pre výpočet emócie sú zaznamenané do systému.

3.3.1 Analýza

Pri spustení prebratého softvérového riešenia sa spustia sledovače(ActivityWatcher), ktoré sledujú aktivitu a zaznamenávajú údaje ako stlačené klávesy, otvorené programy a iné. Tieto údaje sú v systéme reprezentované ako rôzne stavy(State). Tie sú po nastaní určitého míľnika(milestone) odoslané v XML formáte.
Projekt Luxand je dostupný pre C# ako knižnica v podobe dll súboru. Tento projekt rozoznáva 66 bodov tváre, pričom zahŕňa oči, obočie, nos, ústa aj bradu. Poskytuje tak vhodnejšie dáta na ďalšie spracovanie.
V Tabuľka 1 sú uvedené všetky body, ktoré Luxand rozoznáva. K týmto bodom je možné pristupovať prostredníctvom ich názvov, uvedených v tabuľke.



Obrázok 17 Body tváre rozoznávané projektom Luxand [2]

Názov bodu tváre	**Hodnota**
FSDKP_LEFT_EYE	0
FSDKP_RIGHT_EYE	1
FSDKP_LEFT_EYE_INNER_CORNER	24
FSDKP_LEFT_EYE_OUTER_CORNER	23
FSDKP_LEFT_EYE_LOWER_LINE1	38
FSDKP_LEFT_EYE_LOWER_LINE2	27

FSDKP_LEFT_EYE_LOWER_LINE3	37
FSDKP_LEFT_EYE_UPPER_LINE1	35
FSDKP_LEFT_EYE_UPPER_LINE2	28
FSDKP_LEFT_EYE_UPPER_LINE3	36
FSDKP_LEFT_EYE_LEFT_IRIS_CORNER	29
FSDKP_LEFT_EYE_RIGHT_IRIS_CORNER	30
FSDKP_RIGHT_EYE_INNER_CORNER	25
FSDKP_RIGHT_EYE_OUTER_CORNER	26
FSDKP_RIGHT_EYE_LOWER_LINE1	41
FSDKP_RIGHT_EYE_LOWER_LINE2	31
FSDKP_RIGHT_EYE_LOWER_LINE3	42
FSDKP_RIGHT_EYE_UPPER_LINE1	40
FSDKP_RIGHT_EYE_UPPER_LINE2	32
FSDKP_RIGHT_EYE_UPPER_LINE3	39
FSDKP_RIGHT_EYE_LEFT_IRIS_CORNER	33
FSDKP_RIGHT_EYE_RIGHT_IRIS_CORNER	34
FSDKP_LEFT_EYEBROW_INNER_CORNER	13
FSDKP_LEFT_EYEBROW_MIDDLE	16
FSDKP_LEFT_EYEBROW_MIDDLE_LEFT	18
FSDKP_LEFT_EYEBROW_MIDDLE_RIGHT	19
FSDKP_LEFT_EYEBROW_OUTER_CORNER	12
FSDKP_RIGHT_EYEBROW_INNER_CORNER	14
FSDKP_RIGHT_EYEBROW_MIDDLE	17
FSDKP_ RIGHT_EYEBROW_MIDDLE_LEFT	20
FSDKP_ RIGHT_EYEBROW_MIDDLE_RIGHT	21
FSDKP_RIGHT_EYEBROW_OUTER_CORNER	15
FSDKP_NOSE_TIP	2
FSDKP_NOSE_BOTTOM	49
FSDKP_NOSE_BRIDGE	22
FSDKP_NOSE_LEFT_WING	43
FSDKP_NOSE_LEFT_WING_OUTER	45
FSDKP_NOSE_LEFT_WING_LOWER	47
FSDKP_NOSE_RIGHT_WING	44
FSDKP_NOSE_RIGHT_WING_OUTER	46
FSDKP_NOSE_RIGHT_WING_LOWER	48
FSDKP_MOUTH_RIGHT_CORNER	3
FSDKP_MOUTH_LEFT_CORNER	4
FSDKP_MOUTH_TOP	54
FSDKP_MOUTH_TOP_INNER	61
FSDKP_MOUTH_BOTTOM	55
FSDKP_MOUTH_BOTTOM_INNER	64
FSDKP_MOUTH_LEFT_TOP	56
FSDKP_MOUTH_LEFT_TOP_INNER	60
FSDKP_MOUTH_RIGHT_TOP	57
FSDKP_MOUTH_RIGHT_TOP_INNER	62
FSDKP_MOUTH_LEFT_BOTTOM	58
FSDKP_MOUTH_LEFT_BOTTOM_INNER	63
FSDKP_MOUTH_RIGHT_BOTTOM	59

FSDKP_MOUTH_RIGHT_BOTTOM_INNER	65
FSDKP_NASOLABIAL_FOLD_LEFT_UPPER	50
FSDKP_NASOLABIAL_FOLD_LEFT_LOWER	52
FSDKP_NASOLABIAL_FOLD_RIGHT_UPPER	51
FSDKP_NASOLABIAL_FOLD_RIGHT_LOWER	53
FSDKP_CHIN_BOTTOM	11
FSDKP_CHIN_LEFT	9
FSDKP_CHIN_RIGHT	10
FSDKP_FACE_CONTOUR1	7
FSDKP_FACE_CONTOUR2	5
FSDKP_FACE_CONTOUR12	6
FSDKP_FACE_CONTOUR13	8

Tabuľka 1 Zoznam rozoznávaných bodov

3.3.2 Návrh

Vzhľadom na prebraté riešenie, ktoré ideme rozširovať, je vhodné navrhnúť nový sledovač (ActivityWatcher), ktorý bude implementovať existujúce rozhranie. Výstupy z kamery je tiež nutné zaznamenávať do nejakej štruktúry. Tu je zasa potrebné navrhnúť nový stav(State), ktorý bude uchovávať 66 bodov získaných z Luxandu.

Obrázok 18 Volanie funkcie v moment nastatia míľnika

Zaznamenanie bodov tváre sme navrhli ako jedno pole 66 bodov, ktoré sa získajú práve v okamihu nastania míľnika(milestone – významná udalosť). Pod pojmom bod rozumieme súradnicu x a súradnicu y.

3.3.3 Implementácia

Pri implementácii bola najdôležitejšou metódou, metóda GetFacialFeatures(), ktorá využíva hlavne metódy z knižnice FaceSDK.

```
class CameraWatcher : IActivityWatcher
...
private FSDK.TPoint[] facialFeaturesTest;
...
public Svc.Interfaces.StateDto OnActivityFinishing()
//metóda volaná v momente nastatia míľnika
        {
            GetFacialFeatures();

            var cameraStateDto = new CameraStateDto()
            {
                facialFeatures = facialFeaturesTest,
            };
            return cameraStateDto;
        }
...
private void GetFacialFeatures()
        {

            ...

                if (FSDK.FSDKE_OK != FSDKCam.GrabFrame(cameraHandle, ref
imageHandle))
                // grab the current frame from the camera
                FSDK.CImage image = new FSDK.CImage(imageHandle);

                Image frameImage = image.ToCLRImage();
                Graphics gr = Graphics.FromImage(frameImage);

                FSDK.TFacePosition facePosition = image.DetectFace();

                // if a face is detected, we detect facial features
                if (facePosition.w != 0)
                {
                    FSDK.TPoint[] facialFeatures =
                    image.DetectFacialFeaturesInRegion(ref facePosition);

                    SmoothFacialFeatures(ref facialFeatures);

                    foreach (FSDK.TPoint point in facialFeatures)
                    gr.FillEllipse(Brushes.DarkBlue, point.x, point.y, 5, 5);
                    gr.DrawRectangle(Pens.LightGreen, facePosition.xc - 2 *
                    facePosition.w / 3, facePosition.yc - facePosition.w / 2,
                    4 * facePosition.w / 3, 4 * facePosition.w / 3);

                    facialFeaturesTest = facialFeatures;
                }
                else // if a face is disappeared, we reset smoothing
parameters
                    ResetSmoothing();
                GC.Collect(); // collect the garbage after the deletion
            }
...
```

Zdrojový kód 4 Implementácia triedy CameraWatcher

Implementovaná bola ešte trieda CameraStateDto(), ktorá implementuje rozhranie StateDto. Jej atribútom je pole `private FSDK.TPoint[] facialFeatures`, kde sa uchováva 66 spomínaných bodov z Luxandu.

3.3.4 Testovanie

ID	8	Názov	Používateľ je sledovaný kamerou a súradnice bodov častí tváre sú zapisované do XML	
Vstupné podmienky			Program je spustený, používateľ sedí pred kamerou a je dobre osvetlený.	
Výstupné podmienky			Body tváre používateľa sú v XML súbore.	
Krok	**Akcia**		**Očakávaná reakcia**	**Skutočná reakcia**
1	Používateľ zapne klientský program		Spustený program, ikona v spondej lište	rovnaká
2	Nechá program logovať		CameraWatcher je v zozname "Running Activity Watchers"	rovnaká
3	Do XML sa zapisujú pravidelne v intervaloch (milestone)súradnice 66 bodov tváre používateľa		Súradnice sú viditeľné v "Manage Logs"	rovnaká
4	XML súbory - aktivity v ActivityCache obsahujú <CameraStateDto> stavy, ktoré obsahujú 66 elementov <FPDto> rozdelených podľa častí tváre		Všetky očakávané XML atribúty sa v XML aj naozaj nachádzajú.	rovnaká

3.4 Získavať obraz z kamery

3.4.1 Analýza

Okrem postačujúcej kvality získaného obrazu musí byť zachytávanie obrazu aj časovo nenáročné aby samotné spracovanie obrazu mohlo prebiehať v reálnom čase.

Dostupné sú v tejto oblasti mnohé riešenia, kvalitných je z tejto množiny však len zlomok projektov. Najadekvátnejším spôsobom získavania fotiek používateľa sa ukázalo byť použitie knižnice EmguCV, ktorá zaobaľuje kód známej knižnice OpenCV do kódu C#.

3.4.2 Návrh

Zachytávanie obrazu, predstavuje iniciačný krok pre spracovanie obrazu tváre používateľa pomocou projektu Luxand. Spracovanie obrazu prebieha cyklicky, v zmysle opakujúcej akcie získania obrazu a jeho následného spracovania. Momentálne bude riešenie vyvíjané samostatne za účelom integrácie do riešenia pre spracovanie obrazu.

3.4.3 Implementácia

Riešenie je implementované v jazyku C#, pomocou spomínanej knižnice EmguCV. Majoritná funkcionalita zachytávania obrazu je zabezpečená pomocou knižničnej triedy *Capture*. Zachytenie obrazu je implementované za pomoci metódy spomínanej triedy, ktorej názov je

QueryFrame. Táto metóda vracia zachytený obrázok v knižničnom formáte *Image*. S týmto formátom sa dá ďalej pracovať napríklad formou upravovania vlastností obrázku akými sú jeho kvalita, poprípade farebná schéma, ktorú je možné pozmeniť z farebnej na čiernobielu a podobne. Potom ako sa obraz zachytí je uložený na disk, pričom jeho názov pozostáva z dátumu a času zachytenia.

3.4.4 Testovanie

Testovací scenár je pomerne jednoduchý. Spočíva v spustení cyklu, v ktorom prebieha zachytávanie obrazu. Očakávaným výsledkom testu je séria uložených obrázkov. Výsledok testu splnil očakávania. Rýchlosť zachytávania obrazu je približne 10 obrázkov za sekundu, čo je postačujúca rýchlosť pre následne spracovanie obrazu.

3.5 Zhrnutie šprintu

Cieľom šprintu bolo implementovať analyzované riešenia z predchádzajúceho šprintu. Počas šprintu sme však identifikovali problémy, kvôli ktorým sme sa rozhodli analyzované riešenia nahradiť. V oblasti spracovania obrazu išlo najmä o použitie projektu Luxand. Počas šprintu sme riešili taktiež úlohy týkajúce sa získavania a spracovania obrazu a venovali sme sa aj tvorbe dokumentácie. Na konci šprintu sme boli schopný identifikovať zo získavaného obrazu body dôležité pre rozpoznávanie emócií.

4 Tretí šprint – Chevrolet

Cieľom tretieho šprintu bolo navrhnúť a implementovať riešenia na rozpoznávanie emocionálneho stavu používateľa. V rámci šprintu sme riešili úlohy:

- Vypracovanie prihlášky na TP Cup
- Logovanie používateľa v intervaloch
- Skrátenie intervalu odosielania snímok
- Strojové učenie prostredníctvom LibSVM
- Zaobstarať a spracovať testovacie dáta
- Výpočet emočného stavu
- Výpočet neutrálneho stavu používateľa
- Preposielanie dát
- Inštalácia služby do IIS
- Konverzia a vkladanie prijatých dát

4.1 Logovanie používateľa v intervaloch

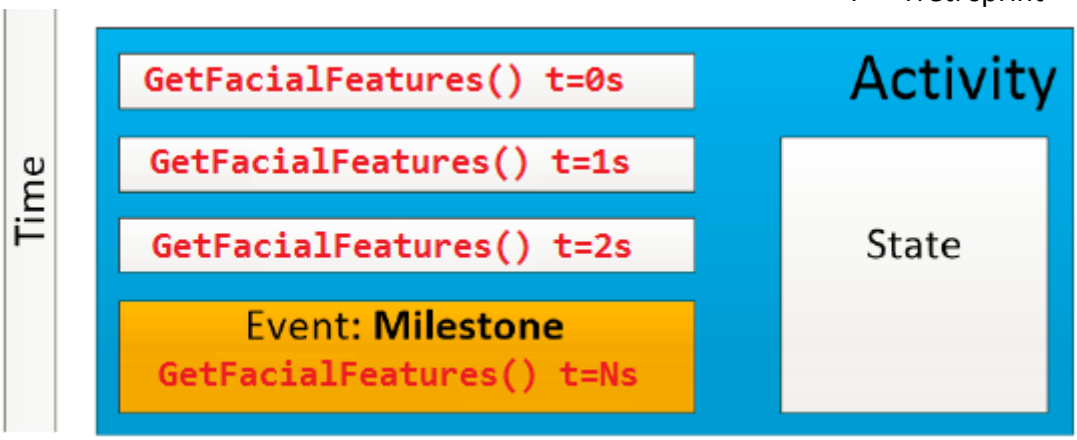
Rozšírenie úlohy Spracovať obraz pomocou Luxandu z predošlého šprintu o získavanie bodov každú sekundu.

4.1.1 Analýza

Štúdovaním prebratého riešenia sme zistili, že pri logovaní v intervaloch by sme sa mohli inšpirovať triedou HWStatusWatcher, ktorá rovnako v intervaloch zaznamenáva údaje o procesore.

4.1.2 Návrh

Navrhli sme logovanie používateľa každú sekundu. Získané body majú dostatočnú informačnú hodnotu na to, aby sme vedeli na strane servera určiť o akú emóciu ide. Musíme vytvoriť triedu, ktorá bude uchovávať nie len jedno pole bodov, ale viacero. Každej sérii 66 bodov, musí byť priradený časový údaj.



Obrázok 19 Volanie funkcie každú sekundu a aj v moment nastania míľnika

4.1.3 Implementácia

Pri implementácii bola rozšírená trieda CameraWatcher.

```
class CameraWatcher : IActivityWatcher
private const double CheckTimeInterval = 1000;
...

        public void Start(ActivityComposer activityComposer,
SystemInputListener
        systemInputListener)
        {
         ...
            if (null == mTimer)
            {
                mTimer = new Timer();
                mTimer.Stop();
                mTimer.Interval = CheckTimeInterval;
                mTimer.AutoReset = false;
                mTimer.Elapsed += new ElapsedEventHandler(mTimer_Elapsed);
                mTimer.Start();
            }
        }
...
private void mTimer_Elapsed(object sender, ElapsedEventArgs e)
        {
            try
            {
                lock (mSyncObj)
                {
                    if (null != mTimer)
                    {
                        GetFacialFeatures();
                    }
                }
            }
            catch (Exception ex)
            {
                AppTracer.Instance.WriteError(ex);
            }
```

```
            finally
            {
                mTimer.Start();
            }
        }
...
```

Zdrojový kód 5 Rozšírenie triedy CameraWatcher

Bola pridaná trieda CameraStatesLlisDto, ktorá obsahuje atribút `private List<CameraStateDto> mStates`, čo je zoznam stavov kamery(každý stav = 66 bodov).

4.1.4 Testovanie

ID	4	**Názov**	Tvár používateľa je každú sekundu zaznamenaná a zapísaná do XML
Vstupné podmienky			Program je spustený, používateľ sedí pred kamerou a je dobre osvetlený.
Výstupné podmienky			Body tváre používateľa sú v XML súbore.

Krok	**Akcia**	**Očakávaná reakcia**	**Skutočná reakcia**
1	Používateľ zapne klientský program	Spustený program, ikona v spodej lište	rovnaká
2	Nechá program logovať	CameraWatcher je v zozname "Running Activity Watchers"	rovnaká
3	Do XML sa zapisujú pravidelne v intervaloch 1 sekunda súradnice 66 bodov tváre používateľa	Súradnice sú viditeľné v "Manage Logs" spolu timestampom	rovnaká
4	XML súbory - aktivity v ActivityCache obsahujú <CameraStateDto> stavy, ktoré obsahujú 66 elementov <FPDto> rozdelených podľa častí tváre	Všetky očakávané XML atribúty sa v XML aj naozaj nachádzajú, každý <CameraStateDto> má svoj timestamp o sekundy väčší ako predošlý(okrem posledného-milestone)	rovnaká

4.2 Skrátenie intervalu odosielania snímok

Úlohou bolo skrátenie intervalu odosielania údajov na server z klientskej časti programu. Interval odosielania bol zmenený z 30 minút na 1 minútu. Ďalej bol znefunkčnený ovládací prvok nastavovania intervalu, aby ho používateľ nemohol zmeniť. Išlo o jednoduché upravenie existujúcich súborov SettingsDesigner.cs a SettingsDialog.xaml.

```
SettingsDesigner.cs
...
[global::System.Configuration.DefaultSettingValueAttribute("00:01:00")]
//("00:30:00")] Commit interval changed to 1 minute.
...
```

Zdrojový kód 6 Nastavenie intervalu logovania

SettingsDialog.xaml

```
...
<extToolkit:DoubleUpDown
Name="mCommitIntervalSpin"
Minimum="0.0" Style="{StaticResource SettingSecondColumnContentStyle}"
IsEnabled="False" />
...
```

Zdrojový kód 7 Znefunkčenie ovládacieho prvku nastavovania interval

4.2.1 Testovanie

ID	5	**Názov**	Používateľ nemôže zmeniť interval odosielania
Vstupné podmienky			-
Výstupné podmienky			-

Krok	**Akcia**	**Očakávaná reakcia**	**Skutočná reakcia**
1	Používateľ zapne klientský program	Spustený program, ikona v spondej lište	rovnaká
2	Zvolí možnosť settings	Otvorí sa dialógové okno settings	rovnaká
3	Commit Interval dialógové okno ukazuje 1 a nie je možné ho zmeniť	Ovládací prvok zmeny intervalu je neaktívny.	rovnaká

ID	6	**Názov**	Klientský program odosiela údaje na server každú minútu
Vstupné podmienky			-
Výstupné podmienky			-

Krok	**Akcia**	**Očakávaná reakcia**	**Skutočná reakcia**
1	Používateľ zapne klientský program	Spustený program, ikona v spondej lište	rovnaká
2	Zvolí možnosť settings	Otvorí sa dialógové okno settings	rovnaká
3	Otvorí logger programu - možnosť Show Log	Otvorí sa dialógové okno logov	rovnaká
	ActivityCache by mal meniť stav na Sending každých 60 sekúnd ("ActivityCache state has been changed to Sending")	ActivityCache mení stav na Sending každých 60 sekúnd	rovnaká

4.3 Strojové učenie prostredníctvom LibSVM

4.3.1 Analýza

Naším cieľom je na základe zistených súradníc dôležitých bodov tváre následne určiť aktuálny emocionálny stav používateľa. Okrem manuálneho spracúvania týchto dát a rozpoznávania určitých vlastností, ktoré sú typické pre danú emóciu je možné túto úlohu vykonať aj pomocou takzvaného strojového učenia.

Jedným z riešení, ktoré sa zameriavajú na strojové učenie je aj knižnica SVM(*Secure Vector Machine*). Táto knižnica implementuje algoritmus pre strojové učenie s rovnomenným názvom. Poskytuje funkcie pre učenie sa na tréningovej množine. Pomocou tohto učenia sa následne vytvorí model, ktorý je možné aplikovať na dáta, ku ktorým je potrebné priradiť určité vlastnosti, ktorých charakteristické črty boli v tomto modely naučené. Podporuje viacero obmien algoritmu ako napríklad polynomiálny, lineárny, sigmoidný a podobne.

4.3.2 Návrh

Pomocou metódy SVM sa bude strojové učenie realizovať tak, že sa najprv vytvorí model, pomocou ktorého sa následne budú určovať emocionálne stavy z novo zistených súradníc. Model bude vytvorený pomocou spracovania množiny príkladov, ktoré sú určené na učenie, teda obsahujú súradnice bodov tváre a ku každej sérií týchto súradníc je zadaná aj emócia, ktorá bola na analyzovanej snímke zobrazená. Dáta na trénovanie budú získané z externej databázy obrázkov, ktorá sa špecializuje na zachytenie emócií na tvári.

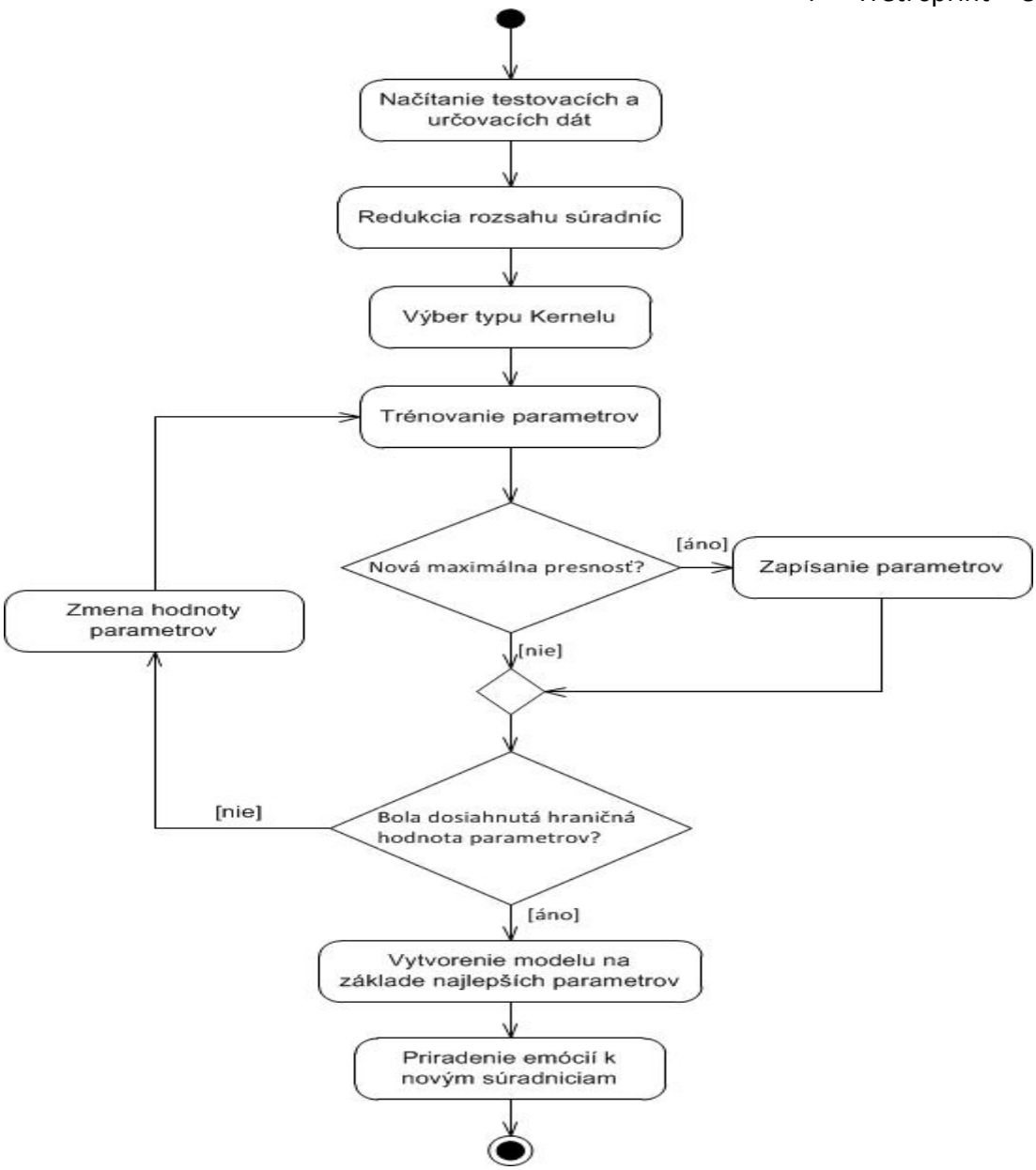
4.3.3 Implementácia

Implementácia je realizovaná pomocou knižnice lib SVM, konkrétne sa jedná o zaobalenie C++ kódu do jazyka C#. Dáta, ktoré sú určené na trénovanie ale aj na následné určovanie vlastností sú reprezentované triedou *Problem.* Pomocou jej metódy *read* sa v našom riešení načítavajú jednotlivé dáta, zvlášť trénovacie a zvlášť tie na určovanie. Formát údajov je pomerne jednoduchý, jedná sa o vektor. Prvá číslica vektoru predstavuje príslušnosť súradníc k emócií. Následne sú za touto číslicou uvádzané súradnice a to tak, že najprv sa zadá poradové číslo súradnice, dvojbodka a konkrétna hodnota danej súradnice. Nasledovná súradnica je uvedená rovnakým spôsobom ale od predošlej je oddelená medzerou.

Pre čo najlepší výsledok je potrebné aby dáta nemali od seba veľký rozptyl. Za týmto účelom je v knižnici implementovaná trieda *Scaling* a pomocou jej metódy *Scale* následne prepočítavame dáta do intervalu -1 a 1. Potom ako zúžime rozsah dát určíme typ *Kernelu*, pomocou ktorého bude prebiehať výpočet (už spomínané variácie algoritmu polynomiálneho či lineárneho typu). Typ *Kernelu* sa určuje pomocou metódy *kerneltype* triedy *Parameter*. Táto trieda zaobaľuje parametre, ktorých hodnota sa určí na základe spracovania údajov určených na trénovanie.

Samotné trénovanie je realizované pomocou triedy *ParamaterSelection*, konkrétne pomocou jej metódy *grid*. Parametrami tejto metódy sú jednak dáta na trénovanie, objekt pre parametre, a premenné pre C a gamma, čo sú parametre, ktorých hodnota sa pri trénovaní určuje. Trénovanie prebieha skúšaním rôznych hodnôt týchto parametrov a porovnávaní ich úspešnosti.

Potom ako sa zistia najlepšie hodnoty parametrov sa vytvorí natrénovaný model pomocou metódy *train*, ktorá prislúcha k triede *Training*. Pomocou tohto modelu je následne možné vykonávať určovanie emócií k vektorom súradníc. Táto funkcionalita je napĺňaná pomocou metódy *Predict*, ktorá prislúcha triede *Prediction*. Princíp algoritmu je znázornený na Obrázok 20.



Obrázok 20 Diagram aktivít popisujúci princíp učenia sa pomocou metódy SVM

4.3.4 Testovanie

Testovací scenár je založený na skutočnosti, že našim cieľom je v konečnom dôsledku čo najpresnejšie určenie emócie k danému vektoru súradníc. Preto sa testovanie realizovalo tak, že pomocou súradníc, ktoré boli získané spracovaním databázy obrázkov, bolo realizované vytvorenie modelu, ktorý vznikol naučením sa súradníc c a gamma. Následne bolo realizované určovanie emócií k súradniciam, ktoré sme taktiež získali spracovaním obrázkov z databázy, ktorá sa orientuje na problematiku vyjadrovania emócií pomocou tváre. Pri týchto súradniciach bola taktiež uvedené emócia, ktorú predstavujú aby bolo následne možné určiť presnosť odhadu na základe naučeného modelu.

Testované boli rôzne variácie algoritmu, ktoré knižnica poskytuje. Pomocou testovania sa dokázal predpoklad nutnosti zúženie rozptylu súradníc a taktiež bola identifikovaná najúspešnejšia variácia algoritmu pre našu problémovú doménu, ktorou bola polynomiálna metóda algoritmu SVM. Konkrétna úspešnosť jednotlivých metód je znázornená v Tabuľka 2.

Meranie	Metóda	Úspešnosť
1.	LIN	85 %
2.	LIN	100 %
3.	LIN	85 %
4.	POLY	100 %
5.	POLY	100 %
6.	POLY	100 %
7.	RBF	78 %
8.	RBF	78 %
9.	RBF	78 %
10.	SIGMOID	85 %
11.	SIGMOID	93 %
12.	SIGMOID	93 %

Tabuľka 2 Variácie SVM a ich úspešnosť

Na základe použitia polynomiálnej metódy môžeme prehlásiť, že výsledky testovania splnili očakávania.

4.4 Zaobstarať a spracovať testovacie dáta

Cieľom úlohy bolo nájsť zdroj relevantných dát, ktoré by sa mohli využiť ako trénovanie dáta pri strojovom učení. Najvhodnejší je dátový set obsahujúci emócie, ktoré chceme rozoznávať. Ako dáta na trénovanie sme sa rozhodli použiť dátový set z The Bosphorus Database6[1] . Jedná sa o databázu 105 ľudí. Každá osoba je nafotená v emocionálnych stavoch hnev, strach, šťastie, znechutenie, smútok a prekvapenie. Databáza obsahuje aj 3D modely a ďalšie fotky, ktoré boli pre nás nezaujímavé.

Fotky osôb sme spracovali pomocou projektu Luxand. Pre každú osobu a emóciu bolo výstupom 66 bodov, ktoré boli uložené do textového súboru v nasledujúcej podobe:

1. Anger
2. Disgust
3. Fear
4. Happiness
5. Sadness
6. Surprise

Formát dát: číslo emócie(1-6) 1:x 2:y 3:x 4:y...(až po 132 = pre všetkých 66 bodov z luxandu)

4.5 Výpočet emočného stavu

Po identifikácii potrebných rozmeroch tváre používateľa dokážeme s určitou presnosťou stanoviť jeho aktuálnu emóciu.

4.5.1 Analýza

Navrhnuté riešenie vychádza z analýzy detekovateľných emočných stavov, ktorej naštudovanie je predpokladom pre začatie ďalšej úlohy.

Dospeli sme k tomu, že voľným okom je možné detekovať 7 základných spontánnych emócií. Program Luxand však neumožňuje adekvátnu presnosť a z tohto dôvodu sme sa rozhodli pre vyradenie detekcie strachu a znechutenia. Výsledkom analýzy je aj zistenie, že ďalším detekovateľným stavom môže byť únava, ktorá bude podobná s prekvapením, avšak nebude sa meniť vzdialenosť obočia od oka.

Následne sme priradili k jednotlivým emóciam potrebné rozmery, ktoré su uvedené v Tabuľka 3.

Emócia	**Potrebné rozmery**
Radosť	• X – vzdialenosť kútikov pier od seba • Y – vzdialenosť dolného a horného stredného bodu pery • Y – vzdialenosť kútika od horného stredného bodu pery • Y – vzdialenosť kútika od dolného stredného bodu pery
Prekvapenie	• Y – vzdialenosť dolného a horného stredného bodu pery • Y – vzdialenosť vnútorného bodu obočia od vnútorného bodu oka
Hnev	• X – vzdialenosť kútikov pier od seba • Y – vzdialenosť dolného a horného stredného bodu pery • Y – vzdialenosť kútika od horného stredného bodu pery • Y – vzdialenosť kútika od dolného stredného bodu pery • X – vzdialenosť vnútorných bodov obočia od seba • Y – vzdialenosť vnútorného bodu obočia od vnútorného bodu oka
Smútok	• X – vzdialenosť kútikov pier od seba • Y – vzdialenosť kútika od horného stredného bodu pery • Y – vzdialenosť kútika od dolného stredného bodu pery
Únava	• Y – vzdialenosť dolného a horného stredného bodu pery
Pohŕdanie	• X – vzdialenosť kútikov pier od seba • Y – vzdialenosť dolného a horného stredného bodu pery • Y – vzdialenosť kútika od horného stredného bodu pery • Y – vzdialenosť kútika od dolného stredného bodu pery

Tabuľka 3 Rozmery pre jednotlivé emócie

Problematickou časťou môže byť aj rozdiel vo vzdialenosti používateľa od kamery, prípadne zmene kamery pri rovnakom používateľovi. Vhodným riešením bude koeficient založený na pevných bodoch tváre, teda takých, ktoré sa nehýbu pri žiadnej emócii.

4.5.2 Návrh riešenia

Následne sa pokúsime o stanovenie jednotlivých odchýlok na rozmeroch tváre používateľa a následne z nich vyvodíme emóciu.

4.5.2.1 Radosť

Zväčšenie rozmerov (oproti neutrálnemu stavu):

- X – vzdialenosť kútikov pier od seba
- Y – vzdialenosť dolného a horného stredného bodu pery
- Y – vzdialenosť kútika od horného stredného bodu pery
- Y – vzdialenosť kútika od dolného stredného bodu pery

Nárast vyššie spomenutých hodnôt opisuje úsmev na tvári. Čím väčšiu hodnotu dosahujú, tým výraznejší je úsmev.

4.5.2.2 Prekvapenie

Zväčšenie rozmerov (oproti neutrálnemu stavu):

- Y – vzdialenosť dolného a horného stredného bodu pery
- Y – vzdialenosť vnútorného bodu obočia od vnútorného bodu oka

Tieto hodnoty opisujú otvorené ústa a zdvihnuté obočie, čo opisuje prekvapenie.

4.5.2.3 Hnev

Zmenšenie rozmerov (oproti neutrálnemu stavu):

- X – vzdialenosť kútikov pier od seba
- Y – vzdialenosť dolného a horného stredného bodu pery
- Y – vzdialenosť kútika od horného stredného bodu pery
- Y – vzdialenosť kútika od dolného stredného bodu pery
- X – vzdialenosť vnútorných bodov obočia od seba
- Y – vzdialenosť vnútorného bodu obočia od vnútorného bodu oka

Hodnoty opisujú stiahnutie pier a obočia smerom k nosu.

4.5.2.4 Smútok

Zväčšenie rozmerov (oproti neutrálnemu stavu):

- X – vzdialenosť kútikov pier od seba
- Y – vzdialenosť kútika od horného stredného bodu pery

Zmenšenie rozmerov (oproti neutrálnemu stavu):

- Y – vzdialenosť kútika od dolného stredného bodu pery

4.5.2.5 Únava

Zväčšenie rozmerov (oproti neutrálnemu stavu):

- Y – vzdialenosť dolného a horného stredného bodu pery

Prihliada sa aj na to, aby Y-vzdialenosť vnútorného bodu obočia od vnútorného bodu oka bola nemenná, aby sme odlíšili prekvapenie od únavy.
Výraznosť jednotlivých emócií bude určovaná na základe rozdielov spomenutých hodnôt s tým, že na základe testovania bude stanovené dosiahnuteľné maximum týchto rozdielov.

4.5.2.6 Koeficient vzdialenosti

Koeficient vychádza z toho, že neutrálny stav používateľa pozná hodnotu vzdialenosti očí od seba.
Predpokladajme, že *eyeDist* je vzdialenosť očí od seba.

$$koeficient = \frac{eyeDist(actualState)}{eyeDist(neutralState)}$$

Pri získavaní emócie sú potrebné rozmery zakaždým prepočítavané do stavu vzdialenosti korešpondujúcej s neutrálnym stavom.

4.5.3 Implementácia

Úloha bola implementovaná v jazyku C# v prostredí Microsoft Visual Studio 2010.

4.6 Výpočet neutrálneho stavu používateľa

Úloha zahŕňa návrh potrebných výpočtov pre zisťovanie emócií zo snímok. Východiskom bude predspracovanie neutrálneho stavu používateľa do rozmerov potrebných na stanovenie emočného stavu.

4.6.1 Analýza

Navrhnuté riešenie vychádza z analýzy detekovateľných emočných stavov. Podstatnou časťou je identifikácia potrebných bodov tváre a následné určenie potrebných vzdialeností pre získanie emócie.

Nie všetky emócie detekovateľné voľným okom sú však zachytiteľné programom Luxand. Aj s týmto problémom je potrebné pri návrhu počítať.

4.6.2 Návrh

Výpočet neutrálneho stavu je operácia nevyhnutná pre získanie emócie používateľa. Počiatočným bodom splnenia úlohy je stanovenie potrebných rozmerov na stanovenie emócie.

Identifikované vzdialenosti bodov tváre potrebných pre získanie emócií sú uvedené v Tabuľka 4. Tieto body následne použijeme pre získanie emočného stavu používateľa.

Potrebné rozmery	**Použité Luxand body**
X - vzdialenosť očí od seba	23, 26
X - vzdialenosť vnútorných bodov obočia od seba	13, 14
X - vzdialenosť kútikov pier od seba	3, 4
Y – vzdialenosť dolného a horného stredného bodu pery	54, 55
Y – vzdialenosť kútika od horného stredného bodu pery	3, 4, 54
Y – vzdialenosť kútika od dolného stredného bodu pery	3, 4, 55
Y – vzdialenosť vnútorného bodu obočia od vnútorného bodu oka	19, 24

Tabuľka 4 Identifikované vzdialenosti bodov tváre

4.7 Preposielanie dát

4.7.1 Analýza

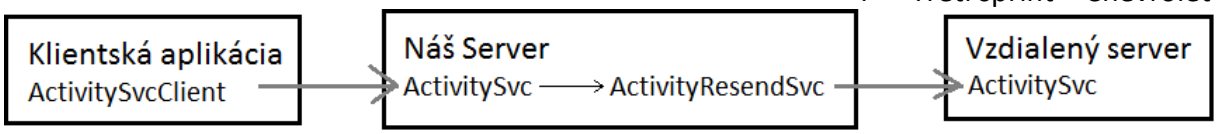
Pôvodný program - logovač, na ktorého základe vytvárame náš projekt, odosiela údaje na vzdialený server. Tento projekt sme zmenili tak, že zaznamenané dáta sa odosielajú na náš server. Tvorcovia pôvodného programu by však boli radi, aby sa údaje zaznamenané našim riešením posielali aj do ich databázy. Z toho dôvodu je potrebné naprogramovať preposielanie údajov na ich vzdialený server.

Naša aplikácia zaznamenáva a odosiela viac entít ako pôvodné riešenie, čo znamená, že údaje odosielané touto aplikáciou nie sú kompatibilné. Ak vzdialený server prijme entitu, ktorú nevie rozpoznať, uloženie údajov skončí chybou a nebude úspešné. Preto je potrebné tieto dáta, pred odoslaním na tento server, očistiť od entít, ktoré sme pridali do projektu.

4.7.2 Návrh

Pre odosielanie údajov na server je potrebné vytvoriť spojenie klientskej aplikácie so serverom. V tomto prípade bude webová služba klientom a služba na vzdialenom serveri je serverom.

Databáza, s ktorou pracuje naša aplikácia obsahuje viac entít ako pôvodná databáza na vzdialenom serveri. V prípade ak by sa na tento vzdialený server preposielali aj novovytvorené entity, nastala by chyba, ktorá by znamenala neúspešný prenos. Preto je potrebné tieto nové entity vyfiltrovať.



Obrázok 21 Schéma preposielania údajov

Obrázok 21 obsahuje schému preposielania údajov. Klientská aplikácia odošle zachytenú aktivitu na náš server, kde ju server prijme pomocou služby *ActivitySvc*. *ActivitySvc* uloží aktivitu do databázy. Následne z tejto aktivity odstráni entity, ktoré nie sú kompatibilné so službou na vzdialenom serveri a takto upravenú entitu odošle pomocou ActivityResendSvc na tento vzdialený server.

4.7.3 Implementácia

Na to, aby sme umožnili spojenie služby na našom serveri so službou na vzdialenom serveri, vytvorili sme novú triedu *ActivityResendSvc*, ktorej zdrojový kód vychádza z kódu triedy *ActivitySvcClient* nachádzajúcej sa v klientskej aplikácii. V metóde *BeforeSendRequest* sme zabezpečili, že do hlavičky správy, ktorou je preposielaná aktivita sa zaznamenajú informácie o používateľovi, ktorý odoslal pôvodnú správu.

V metóde *CommitActivity* triedy *ActivitySvc* sme upravili kód tak, aby z aktivity boli odstránené všetky entity, ktoré by sa nemali odoslať na vzdialený server.

```
// these variables will contain a list of events/states which should be removed from
activity before resending to remote server.
List<StateDto> statesToRemove = new List<StateDto>();
List<EventDto> eventsToRemove = new List<EventDto>();

// saving each state into database and making a list of states which shouldnt be
resent to another server
foreach (StateDto stateDto in activity.States)
{
    State state =
ConvertorManager.Instance.GetStateConvertor(stateDto.GetType()).CreateDbEntity(uac,
stateDto);
    act.States.Add(state);
    uac.States.AddObject(state);
    uac.SaveChanges();

    if (!OriginalDTObject.Default.IsOriginal(stateDto))
    {
        statesToRemove.Add(stateDto);
    }
}

// saving each event into database and making a list of events which shouldnt be
resent to another server
foreach (EventDto eventDto in activity.Events)
{
    Event evnt =
ConvertorManager.Instance.GetEventConvertor(eventDto.GetType()).CreateDbEntity(uac,
eventDto);
    act.Events.Add(evnt);
```

```
    uac.Events.AddObject(evnt);
    uac.SaveChanges();

    if (!OriginalDTObject.Default.IsOriginal(eventDto))
    {
         eventsToRemove.Add(eventDto);
    }
}

// removing uncompatible states and events from activity
foreach (StateDto notOriginalState in statesToRemove)
{
    activity.States.Remove(notOriginalState);
}
foreach (EventDto notOriginalEvent in eventsToRemove)
{
    activity.Events.Remove(notOriginalEvent);
}

// resend activity to remote server
ActivityResendSvc.Default.CommitActivity(activity);
```

Zdrojový kód 8 Odstránenie nekompatibilných entít a preposielanie

Pre jednoduchšie filtrovanie nových entít bola vytvorená trieda OriginalDTObject, ktorá obsahuje zoznam pôvodných entít typu StateDto a EventDto. Pomocou metódy IsOriginal je možné zistiť, či entita je pôvodná.

```
Class OriginalDTObject
        private static readonly Lazy<OriginalDTObject> mInstance = new
Lazy<OriginalDTObject>(() => new OriginalDTObject());
        public static OriginalDTObject Default { get { return mInstance.Value; } }

        private IEnumerable<string> stateWhiteList = new string[] {
            "KeyboardStateDto", "KeyboardStateBlobDto", "KeyboardGraphDto",
            "MouseStateDto", "MouseStateBlobDto", "MouseMoveDto", "MouseDragDropDto",
"MouseScrollDto", "MouseClickDto", "MousePosDto",
            "ApplicationStateDto", "RunningApplicationsListDto", "HwUsageDto" };
        private IEnumerable<string> eventWhiteList = new string[] {
            "IdeEventDto", "IdeCheckinDto", "IdeSlnPrjEventDto",
"IdeDocumentOperationDto", "IdeStateChangeDto", "IdeCodeElementEventDto",
"IdeProjectOperationDto", "IdeCodeOperationDto",
            "LyncStatusChangeDto",
            "OneNoteEventDto", "OneNoteNotebookDto", "OneNoteSectionGroupDto",
"OneNoteSectionDto", "OneNotePageDto", "OneNoteNavigateDto", "OneNoteViewChangeDto",
            "ApplicationRunDto", "ApplicationFocusLostDto",
            "WebEventDto", "WebNavigateDto", "WebBookmarkDto", "WebSaveDocumentDto",
"WebTabOperationDto",

        };


        /// <summary>
        /// Checks whether given state was included in original solution
        /// </summary>
        /// <param name="stateDto"></param>
        /// <returns>true if the state is original</returns>
        public bool IsOriginal(StateDto stateDto)
```

```
        {
            string state = stateDto.GetType().ToString();
            int position = state.LastIndexOf('.');
            if (position > -1)
                state = state.Substring(position + 1);
            return (stateWhiteList.Where(needleState =>
state.Contains(needleState)).Count() > 0 ? true : false);
        }

        /// <summary>
        /// Checks whether given event was included in original solution
        /// </summary>
        /// <param name="eventDto"></param>
        /// <returns>true if the event is original</returns>
        public bool IsOriginal(EventDto eventDto)
        {
            string eventstring = eventDto.GetType().ToString();
            int position = eventstring.LastIndexOf('.');
            if (position > -1)
                eventstring = eventstring.Substring(position + 1);
            return (eventWhiteList.Where(needleEvent =>
eventstring.Contains(needleEvent)).Count() > 0 ? true : false);
        }
```

Zdrojový kód 9 Trieda pre kontrolu compatibility entit

4.7.4 Testovanie

Testovanie prebieha neautomatickým spôsobom. Platí, že aktivita bola úspešne odoslaná na server vtedy, ak údaje obsiahnuté v tejto aktivite sú zaznamenané v databáze.

Pri testovaní však nie je vhodné porovnávať identifikátory aktivít, pretože ak už v databáze existuje aktivita s rovnakým identifikátorom, tak táto aktivita nie je uložená. Vhodnejším údajom sú údaje, ktoré sú do databázy ukladané menej často, a ktoré môžu obsahovať unikátne hodnoty.

Postup pri testovaní:

ID	1	**Názov**	Údaje zaznamenané klientskou aplikáciou sú prenesené na vzdialený server
Vstupné podmienky			Anonymizačný reťazec sa nenachádza v databáze v tabuľke Users na serveri Anonymizačný reťazec sa nenachádza v databáze v tabuľke Users na vzdialenom serveri Klientská aplikácia má prístup k internetu Webová služba na serveri je správne nakonfigurovaná
Výstupné podmienky			V tabuľke Users v databáze na serveri sa nachádza záznam obsahujúci zadaný anonymný reťazec V tabuľke Users v databáze na vzdialenom serveri sa nachádza záznam obsahujúci zadaný anonymný reťazec

Krok	**Akcia**	**Očakávaná reakcia**	**Skutočná reakcia**
1	Používateľ zapne klientský program	Spustený program, ikona v spodnej lište	rovnaká

ID	1	**Názov**	Údaje zaznamenané klientskou aplikáciou sú prenesené na vzdialený server	
2	Používateľ klikne na tlačidlo *Settings* v klientskej aplikácii		Otvorí sa dialógové okno *Settings*	rovnaká
3	Používateľ zadá unikátny anonymizačný reťazec dlhší ako 10 znakov do poľa *User Name for Anonymous Connection*		Zadaný reťazec sa nachádza v poli *User Name for Anonymous Connection*	rovnaká
4	Používateľ klikne na tlačidlo OK		Dialógové okno sa zatvorí a nastavenia sú uložené	Rovnaká
5	Používateľ nechá logovač zaznamenávať aktivity aspoň 5 minút		V dialógovom okne *Activity Cache Management* sa nachádzajú nové aktivity	Rovnaká
6	Používateľ klikne na tlačidlo *Show Log*		Otvorí sa dialógové okno *Show Log*. V tomto okne sa nachádzajú správy o činnosti logovača. Medzi správami sa nachádza aj správa '*ActivityCache state has been changed to Sending*'. Po tejto správe nie je zaznamenaná žiadna chyba (error)	Rovnaká
7	Administrátor servera vykoná v databáze UserActivity select nad tabuľkou Users, kde vyberie všetky záznamy		Používateľov unikátny anonymizačný reťazec sa nachádza medzi týmito záznamami	Rovnaká
8	Administrátor vzdialeného servera vykoná v databáze UserActivity select nad tabuľkou Users, kde vyberie všetky záznamy		Používateľov unikátny anonymizačný reťazec sa nachádza medzi týmito záznamami	Rovnaká

4.8 Inštalácia služby do IIS

Požiadavky na server:

- Windows Server 2008/2008 R2
- SQL 2008
- .NET 4.0
- V IIS zaregistrovaný ASP.Net 4.0
- V IIS zaregistrovaný WCF a WF

V službe IIS je potrebné vytvoriť novú aplikáciu (Application), ktorá bude využívať *ApplicationPool* podporujúci .Net Framework v4 a *Managed Pipeline* bude mať nastavený na *Integrated*.

Vytvorte si 2 databázy – UserActivity a Anonymization. Nahrajte do nich skripty priložené v *databaza.zip.* Pre každú z nich nastavte práva pre používateľa, ktorý pod ktorým beží aplikácia.

Obsah súboru *webova_sluzba.zip* je potrebné nahrať na server do zložky, ktorú má pridelená aplikácia v IIS.

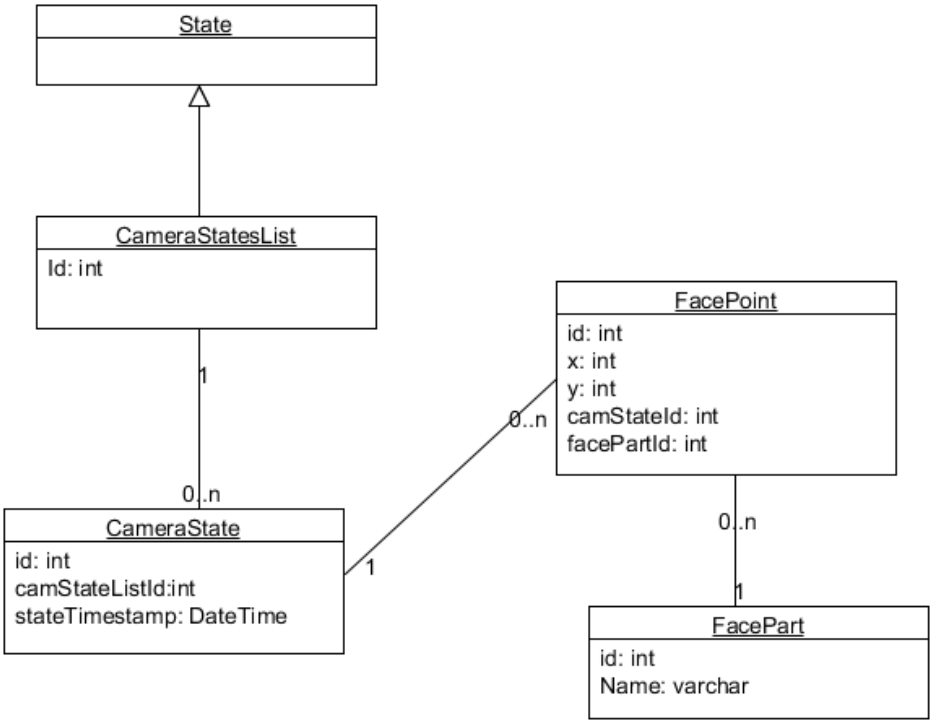
V súbore Web.config, pre *connectionString AnonymizationModelContainer* a *UserActivityContainer* nastavte prihlasovacie údaje do databázy.

4.9 Konverzia a vkladanie prijatých dát

Táto úloha zahŕňala implementáciu konverzie XML súborov na strane servera a ukladanie dát z kamery do databázy.

4.9.1 Analýza

Prvým krokom bolo navrhnutie databázového modelu pre dáta z kamery a vytvorenie tabuliek v databáze. Obrázok 22 zobrazuje UML model databázy.



Obrázok 22 UML model databázy

4.9.2 Implementácia

Na vytvorenie databázových tabuliek bol použitý nástroj Database designer, ktorý je súčasťou vývojového prostredia MS Visual Studio .

Ďalej bolo potrebné implementovať konvertor, ktorý by konvertoval **CameraStatesListDto** objekt na **CameraStatesList** a opačne. Toto samozrejme zahŕňalo aj konvertovanie ostatných tried

- CameraStateDto <--> CameraState
- FPDto <--> FacePoint

Na tento účel bol implementovaný konvertor - trieda **CameraStatesListConvertor** v projekte **UserActivity.Svc**. Táto trieda implementuje interface `IStateConvertor` a poskytuje metódy pre konverziu databázovej entity na XML objekt a opačne.

```
/// <summary>
/// Convertor for camera data, convertign of XML data objects to database
entities and vice versa
/// </summary>
[Export(typeof(IStateConvertor))]
public class CameraStatesListConvertor : IStateConvertor
{
 /// <summary>
 /// This method converts StateDto (XML) object - CameraStatesListDto to
database entity object CameraStatesList
 /// </summary>
 /// <param name="userActivityContainer">Database object used to access
database</param>
 /// <param name="stateDto">CameraStatesListDto object to be
converted</param>
 /// <returns></returns>
 public State CreateDbEntity(UserActivityContainer userActivityContainer,
StateDto stateDto)
 {
     // conversion code

     return cameraStates;
 }

 /// <summary>
 /// Conversion of database entity object CameraStatesList to XML data
object CameraStatesListDto
 /// </summary>
 /// <param name="stateEntity">Database entity object -
CameraStatesList</param>
 /// <returns>Converted CameraStatesListDto object</returns>
 public StateDto CreateDto(State stateEntity)
 {
    // conversion code

    return dto;
 }
}
```

Zdrojový kód 10 Implementácia triedy CameraStatesListConvertor

Do databázy sa ukladá sa všetkých 66 bodov získaných z Luxandu. Jednotlivé body sa nachádzajú v tabuľke **FacePoints**, kde vždy 66 bodov patri jednému záznamu v tabuľke **CameraStates,** čo je vlastne jeden výstup Luxandu - CameraStateDto v XML.

4.10 Zhrnutie šprintu

V treťom šprinte sme sa venovali výpočtu neutrálneho a emočného stavu používateľa. Taktiež sme sa venovali výpočtu stavu pomocou strojového učenia, úprave a preposielaniu dát a rôznym menším podporným úlohám. Na koncišprintu sme boli schopný zo zachyteného obrazu zisťovať rôzne emocionálne stavy používateľa. Niektoré stavy však boli ťažko identifikovateľné a niektoré nie je možné rozpoznať vôbec.

5 Štvrtý šprint – Dodge

V štvrtom šprinte sme pokračovali v riešení niektorých úloh z predchádzajúceho šprintu a rozpoznávaní emocionálneho stavu používateľa. Taktiež sme sa venovali údržbe kódu, testovaniu aplikácie a tvorbe a kompletizácii dokumentácie. V rámci šprintu sme riešili úlohy:

- Výpočet emócie z neutrálneho stavu a uloženie výsledku do DB
- Vytvoriť spúšťač na výpočet emócie
- Vytvoriť DB model
- Normalizácia v loggeri
- Testovanie aplikácie
- Údržba kódu
- Vytvorenie inštalačného balíčka
- Tvorba dokumentácie č. 2
- Strojové učenie prostredníctvom Neurónových sietí
- Získanie neutrálneho stavu používateľa

5.1 Vytvoriť spúšťač na výpočet emócie

5.1.1 Analýza

Niektoré časti riešenia sú serverové aplikácie, ktoré potrebujeme spustiť vždy v tom istom časovom intervale. Na túto úlohu je najlepšie riešenie vytvorenie spúšťača na serveri, ktorý beží v pozadí, potrebné aplikácie na serveri vždy spustí a posiela im údaje na vstup. Spúšťač spracuje výstup spustených aplikácií a uloží ho do databázy.

5.1.2 Návrh

V našom projekte nebude existovať iba jeden spôsob na odhaľovanie emocionálneho stavu používateľa, preto je potrebné, aby sme mali možnosť na nastavenie spúšťača. Je potrebné, aby spúšťač mal používateľské rozhranie aspoň so základnými údajmi:

- Spôsob odhaľovania emocionálneho stavu
- Časový interval, za ktorý vždy spustí aplikácie na serveri
- Mená aplikácií, ktoré má spustiť a posielať im údaje na vstup

5.1.3 Implementácia

Na základe nášho návrhu bol vytvorený spúšťač. Na našom serveri zatiaľ bežia 2 aplikácie, prvá na výpočet neutrálneho stavu a druhá na výpočet emocionálneho stavu. Výpočet emocionálneho stavu potrebuje na vstup aj neutrálny stav, z tohto dôvodu je zadefinovaný

základný neutrálny stav, ktorý je všeobecný a používa sa pri vždy pri prvom spustení. Spúšťač číta potrebné údaje z databázy raz za sekundu, spracuje ich a posiela na vstup aplikácií.

Výpočet emocionálneho a neutrálneho stavu prebieha s aktuálnymi údajmi a aby v budúcnosti bolo možné vytvoriť odporúčania, je potrebné aby vždy boli spracované aktuálne údaje. Údaje do databázy prichádzajú tiež raz za sekundu, čiže stačí, ak spúšťač spracuje najnovšie údaje – ktoré sú zároveň aj aktuálne. Spúšťač údaje spracuje a na vstup aplikácií už posiela body vo presne definovanom formáte.

Z programátorského hľadiska je náš spúšťač aplikácia vytvorená v programovacom jazyku C#. Aplikácia číta databázu každú sekundu, údaje spracuje a uloží do súboru, z ktorého ostatné aplikácie pracujú. Po uložení spúšťač pošle aplikácie, ktoré vykonajú svoje úlohy a uložia ich výsledky. V spúšťači je možnosť nastaviť spôsob odhaľovanie emocionálneho stavu a časový interval, za ktorý má spustiť.

5.2 Vytvoriť DB model

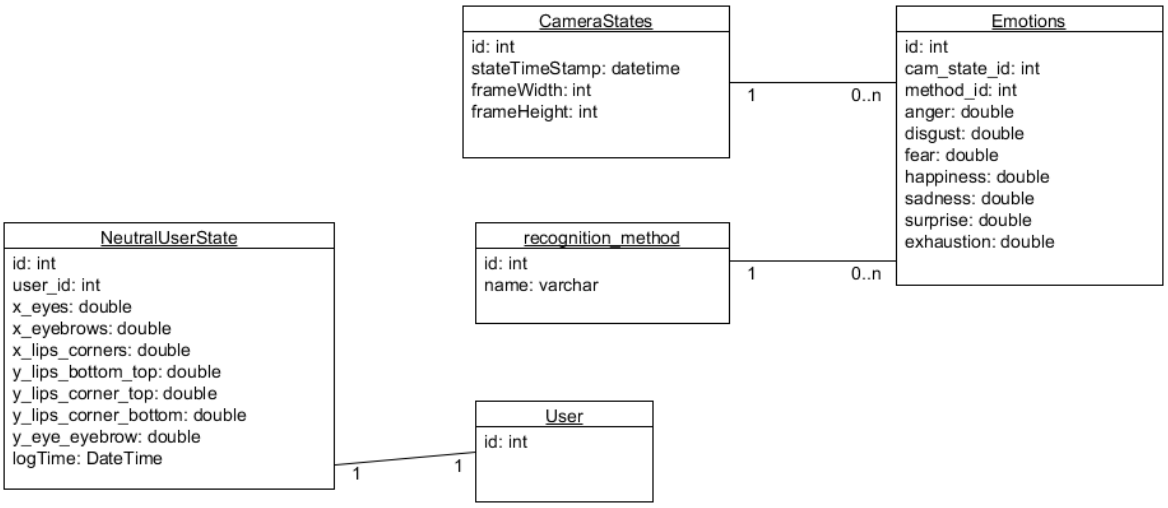
Cieľom tejto úlohy bolo navrhnúť a vytvoriť databázový model pre neutrálny emocionálny stav používateľa a pre vypočítané emócie.

5.2.1 Analýza

Tabuľka pre neutrálny emocionálny stav vychádza z analýzy získavania neutrálneho stavu používateľa, kde sa používa 7 pomerov vzdialeností bodov získaných z kamery. Tabuľka tak obsahuje 7 stĺpcov, kde sa ukladajú tieto pomery. Každý záznam neutrálneho stavu sa vzťahuje na jedného používateľa. Tabuľka tiež obsahuje časovú známku, ktorá udáva kedy bol neutrálny stav vypočítaný.

5.2.2 Návrh

Tabuľka pre ukladanie vypočítaných emócií bola navrhnutá tak, aby mohla byť využívaná všetkými metódami, ktoré počítajú alebo budú počítať emócie. Obsahuje preto 7 stĺpcov - všetky detekovateľné emócie. Tiež obsahuje referenciu na CameraState tabuľku, teda body, z ktorých bola emócia vypočítaná a referenciu na tabuľku RecognitionMethod, teda na metódu, ktorou bola emócia vypočítaná. Jednotlivé stĺpce emócií obsahujú float hodnoty 0-1, v prípade, že daná metóda nepočíta nejakú emóciu, tak v stĺpci je -1. Na Obrázok 23 je UML diagram databázového modelu.



Obrázok 23 UML diagram databázového modelu

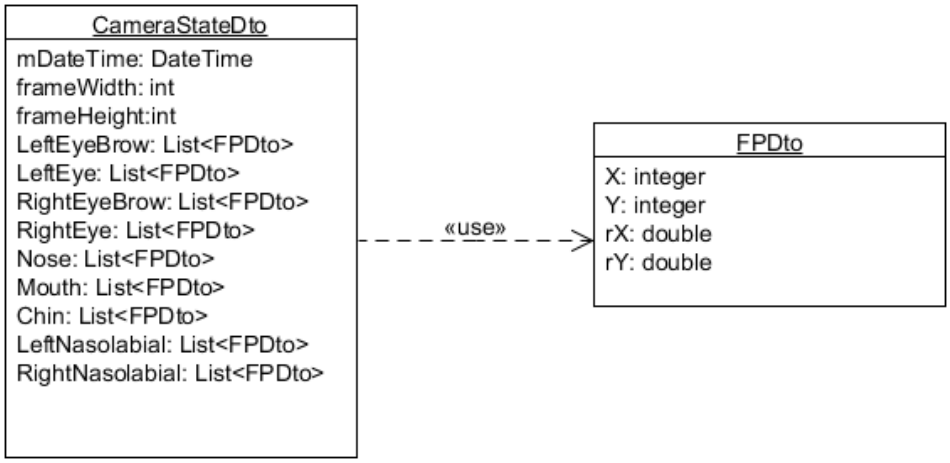
5.3 Normalizácia v loggeri

5.3.1 Analýza

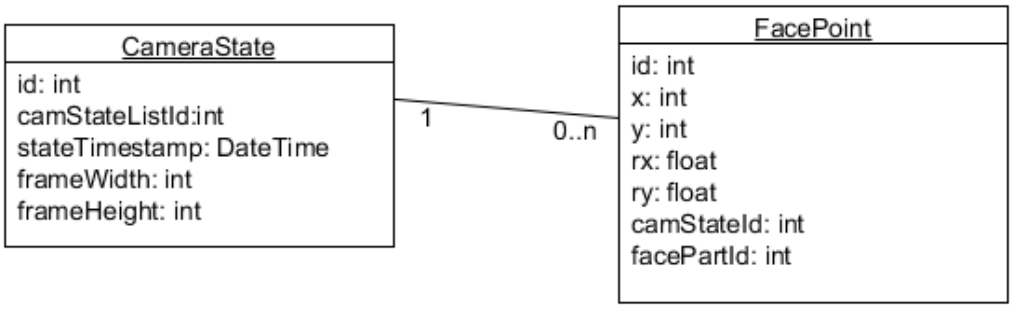
Pre potreby niektorých metód na výpočet emocionálneho stavu (neurónové siete) je potrebné získať normalizované dáta z kamery. Bolo potrebné navrhnúť metódu vyseknutia tváre z obrazu a prepočítania súradníc bodov na nový interval. Takýmto spôsobom sme eliminovali rozdiely v súradniciach pri rôznej vzdialenosti používateľa od kamery.

5.3.2 Návrh

Pre zachovanie spätnej kompatibility a potrebu niektorých metód (neutrálny stav) bolo potrebné ponechať aj pôvodné súradnice. Preto sme teda rozšírili triedy **FPDto** a **FacePoint** o dva nové parametre *rX* a *rY*, teda normalizované súradnice bodov tváre. Na základe analýzy sme sa rozhodli pre každý stav kamery tiež posielať rozmer obrazu, z ktorého snímal tvár používateľa. Do tried **CameraStateDto** a **CameraState** tak pribudli parametre *frameWidth* a *frameHeight*.



Obrázok 24 Triedy s normalizovanými dátami



Obrázok 25 Databázové tabuľky s normalizovanými dátami

5.3.3 Implementácia

Normalizácia sa vykonáva v triede **CameraWatcher**. Pri získavaní súradníc z kamery sa počítajú aj normalizované súradnice v metóde

```
private void createAllStates(FSDK.TPoint[] facialFeatures, FSDK.CImage image)
// Normalization numbers needed before filling DB
// Offset numbers indicate distance from 0, 0 coordinate of a "picture"
int offsetX = facialFeatures[12].x; // Left-most point
int offsetY;                        // Top-most  point

// Acquiring higher(closer to the top) eyebrow
if (facialFeatures[16].y > facialFeatures[17].y) offsetY =
facialFeatures[17].y;
else offsetY = facialFeatures[16].y;

int normalizedSize = 200;    // Setting default expected size of "picture"
int tolerance = 10;          // Acts like 10points frame on every side of face,
in case of incorrect choice of edge points

// Coefficient is number-to-get divided by edge points distance
```

```
double coefX = (normalizedSize - (2 * tolerance)) / (facialFeatures[15].x -
offsetX);
double coefY = (normalizedSize - (2 * tolerance)) / (facialFeatures[55].y -
offsetY);
```

Zdrojový kód 11 Normalizácia dát

Tieto parametre sú potom použité pri výpočte *rX* a *rY* normalizovaných súradníc.

```
createLeftEyeState(cameraStateDto, facialFeatures, offsetX, offsetY, coefX, coefY,
tolerance);
```

Nutné bolo upraviť aj **CameraStatesListConvertor,** aby sa pri konvertovaní dátových objektov skonvertovali aj normalizované súradnice a ukladali sa do databázy spolu s rozmermi obrazu.

5.4 Vytvorenie inštalačného balíčka

Projekt je potrebné otvoriť v programe Visual Studio 2010 SP1. Zároveň by na počítači malo byť nainštalované aj Visual Studio 2010 SP1 SDK [5].

Ak vytvárate inštalačný balíček prvý krát, potrebujete nastaviť oprávnenia v PowerShell-i. Otvorte si program Windows PowerShell a skontrolujte hodnotu pre *Execution policy* spustením príkazu `Get-ExecutionPolicy`. Ak máte túto hodnotu nastavenú na *Restricted*, spustite píkaz `Set-ExecutionPolicy RemoteSigned`.

Postup pri vytvorení samotnej inštalácie je už jednoduchý. Vo *Visual Studio* si otvorte riešenie (Solution). V časti *Solution Explorer* kliknite pravým tlačidlom na projekt *UserActivity.AppSetup.* Vyberte možnosť *Build*. Po úspešnom zbehnutí tohto procesu, inštalačný balíček nájdete v zložke <koreňový_adresár_projektu>/UserActivity.AppSetup/Debug

5.5 Strojové učenie prostredníctvom Neurónových sietí

5.5.1 Analýza

Cieľom strojového učenia pomocou neurónových sietí je dokázať identifikovať zo zachytených súradníc tváre aktuálny emocionálny stav používateľa. Princíp metódy spočíva vo vytvorení neurónovej siete, ktorú naučíme na trénovacích dátach na rozpoznávane jednotlivých emócií. Takto natrénovaná sieť je následne schopná ohodnotiť vstupné dáta a priradiť im rôzne triedy z trénovacej množiny.

5.5.2 Návrh

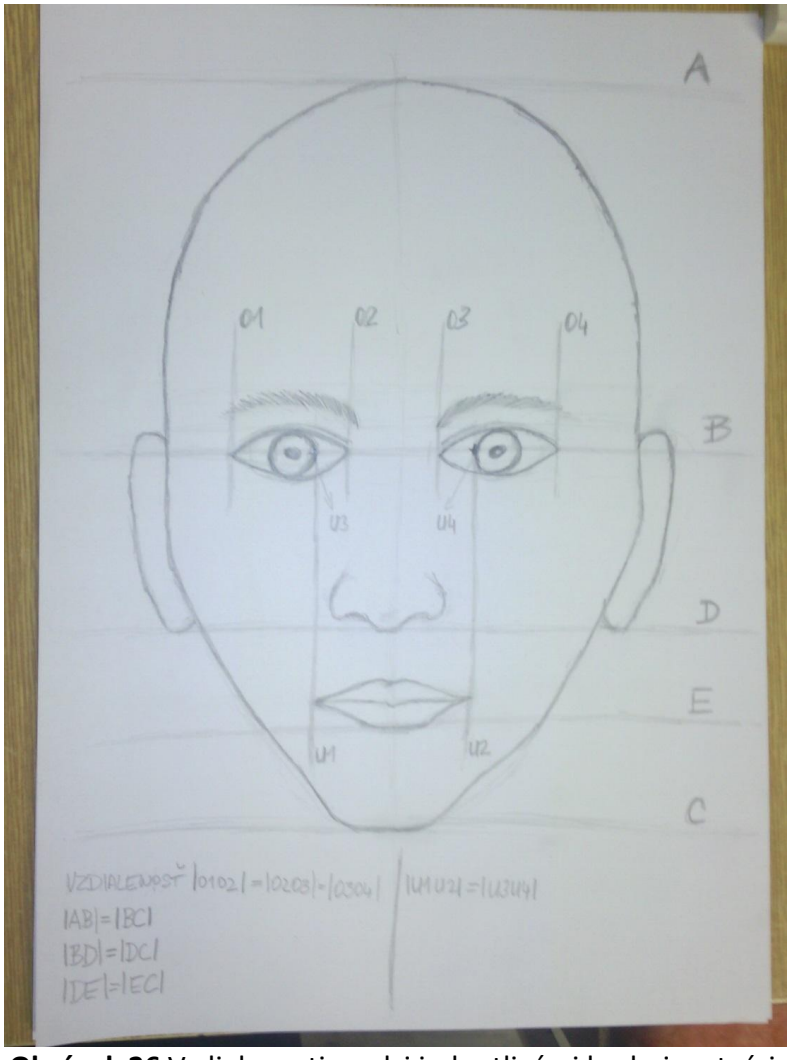
Určovanie emócií pomocou neurónových sietí realizujeme vytvorením programu s dvoma funkciami. Prvá bude vytvorenie neurónovej siete, teda naučenie na známych trénovacích

dátach. Túto funkciu bude možné opakovane spúšťať nad novými dátami, vďaka čomu sa môže presnosť rozpoznávania emócií neustále zvyšovať. Druhá funkcia bude samotné rozpoznávanie, pri ktorom sa použije natrénovaná neurónová sieť na rozpoznanie emócie zo vstupných údajov.

5.6 Získanie neutrálneho stavu používateľa

5.6.1 Analýza

Knižnica Luxand v každej sekunde uloží 66 bodov do databázy. Z týchto bodov sú vypočítané potrebné vzdialenosti na výpočet emocionálneho stavu používateľa. Na to, aby výpočet emocionálneho stavu bol čo najpresnejší, je potrebné vedieť neutrálny stav používateľa. Každý používateľ je iný a má iné jedinečné vzdialenosti na tvári, ale existujú základné pravidlá, ktoré sú u každého používateľa rovnaké. Tieto pravidlá sú súvislosti medzi vzdialenosťami na tvári znázornené na Obrázok 26.



Obrázok 26 Vzdialenosti medzi jednotlivými bodmi na tvári

5.6.2 Návrh

Emócie používateľa trvajú maximálne pár sekúnd, ale väčšinou iba sekundu, alebo ešte menej. Výpočet emocionálneho stavu prebieha na základe neutrálneho stavu používateľa, preto je potrebné definovať základný všeobecný neutrálny stav a potom z tohto stavu upresniť neutrálny stav pre aktuálneho používateľa. Tento základný stav je priemer vzdialeností doterajších používateľov.

5.6.3 Implementácia

Pri výpočte emocionálneho stavu sme vychádzali z faktu, že emócie používateľa trvajú maximálne niekoľko sekúnd a stačí sledovať používateľa 3 minúty, aby sme získali neutrálny stav. Počas 3 minút v každej sekunde sú uložené vzdialenosti potrebné na ďalšie výpočty a z týchto vzdialeností sú vypočítané aktuálne priemery. Týmto spôsobom je v najlepšom prípade možné získať neutrálny stav používateľa už za 20-30 sekúnd a nie sú potrebné ani všetky uložené body z databázy.

5.7 Zhrnutie šprintu

V štvrtom šprinte sme pokračovali vo výpočte emocionálneho stavu používateľa z neutrálneho stavu, ktorý zaznamenávame. Taktiež sme pokračovali v úlohe zisťovania stavu pomocou neurónových sietí. Veľkou časťou šprintu boli udržiavacie úlohy, ako údržba kódu, testovanie a tvorba dokumentácie.

6 Zdroje

[1] http://bosphorus.ee.boun.edu.tr/
[2] A. Savran, B. Sankur, M. T. Bilge, "Comparative Evaluation of 3D versus 2D Modality for Automatic Detection of Facial Action Units", Pattern Recognition, Vol. 45, Issue 2, p767-782, 2012.
[3] http://www.luxand.com/facesdk/documentation/detectedfeatures.php
[4] http://www.gazegroup.org/images/stories/interface3.png
[5] http://www.microsoft.com/en-us/download/details.aspx?id=21835