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, unavení 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 nielen 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 schopní 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.

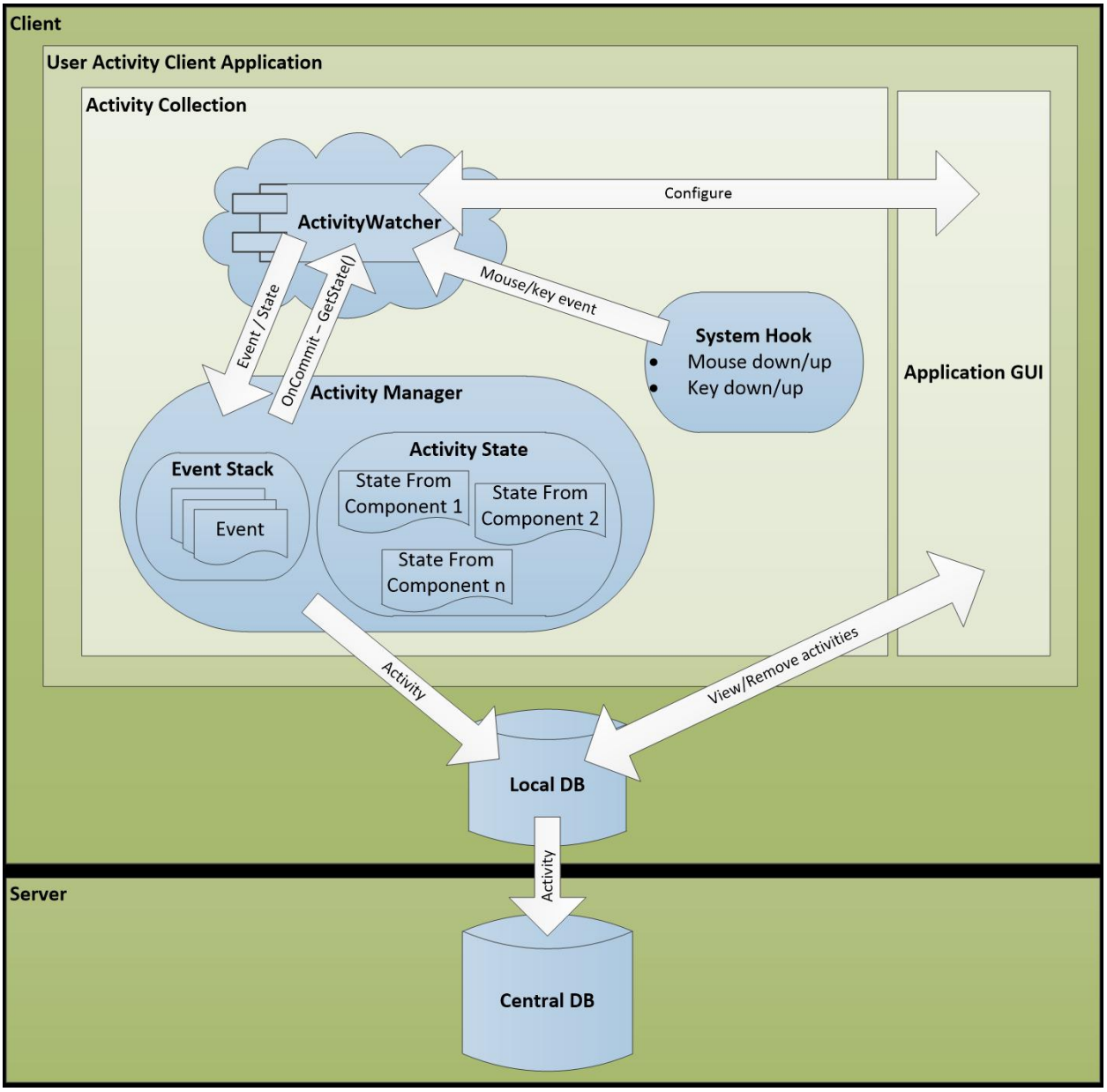
2.1.1 Analýza východzieho stavu

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 systému

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 – aktivity 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)

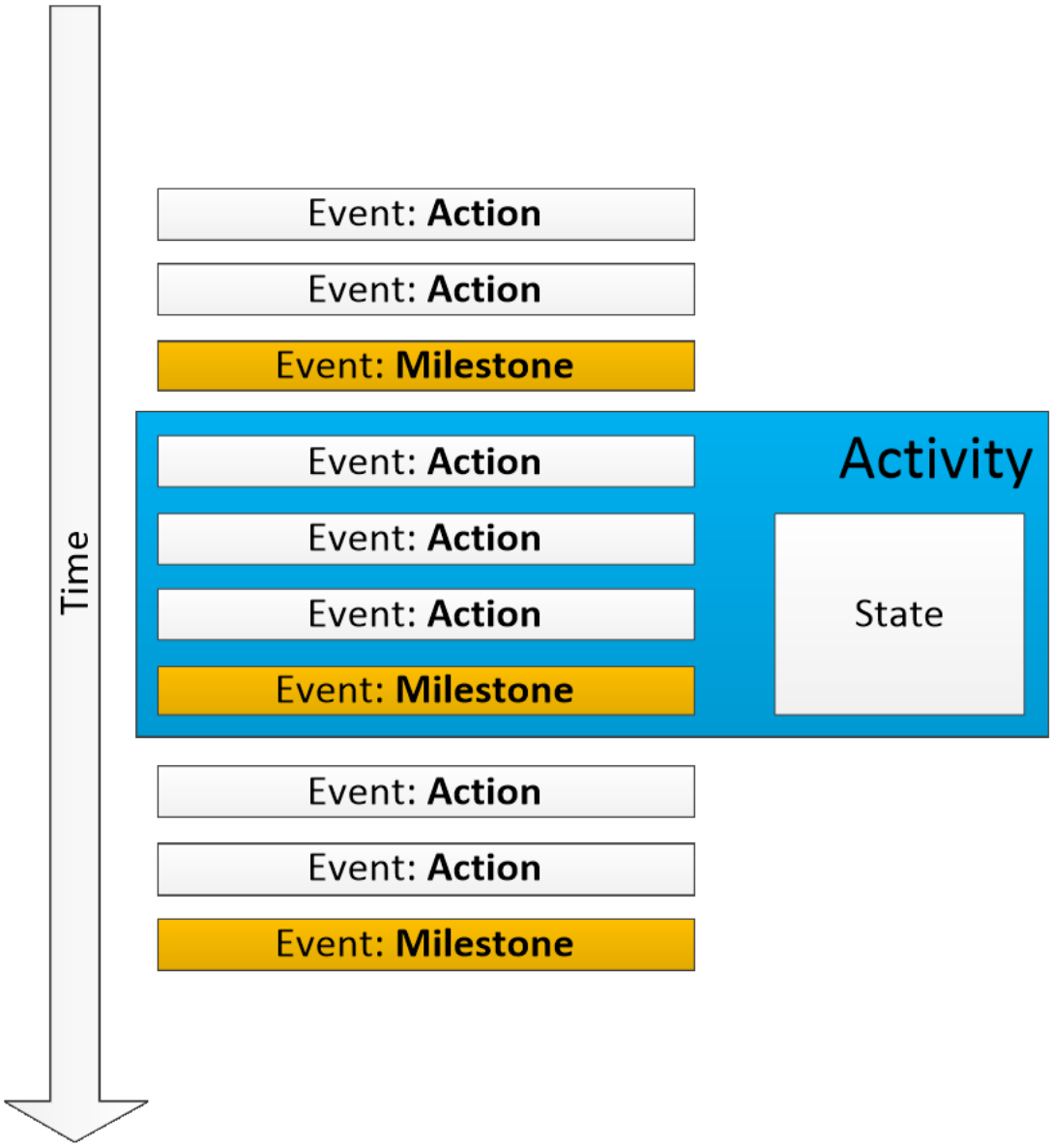
2.1.1.3 Aktivity

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>
```

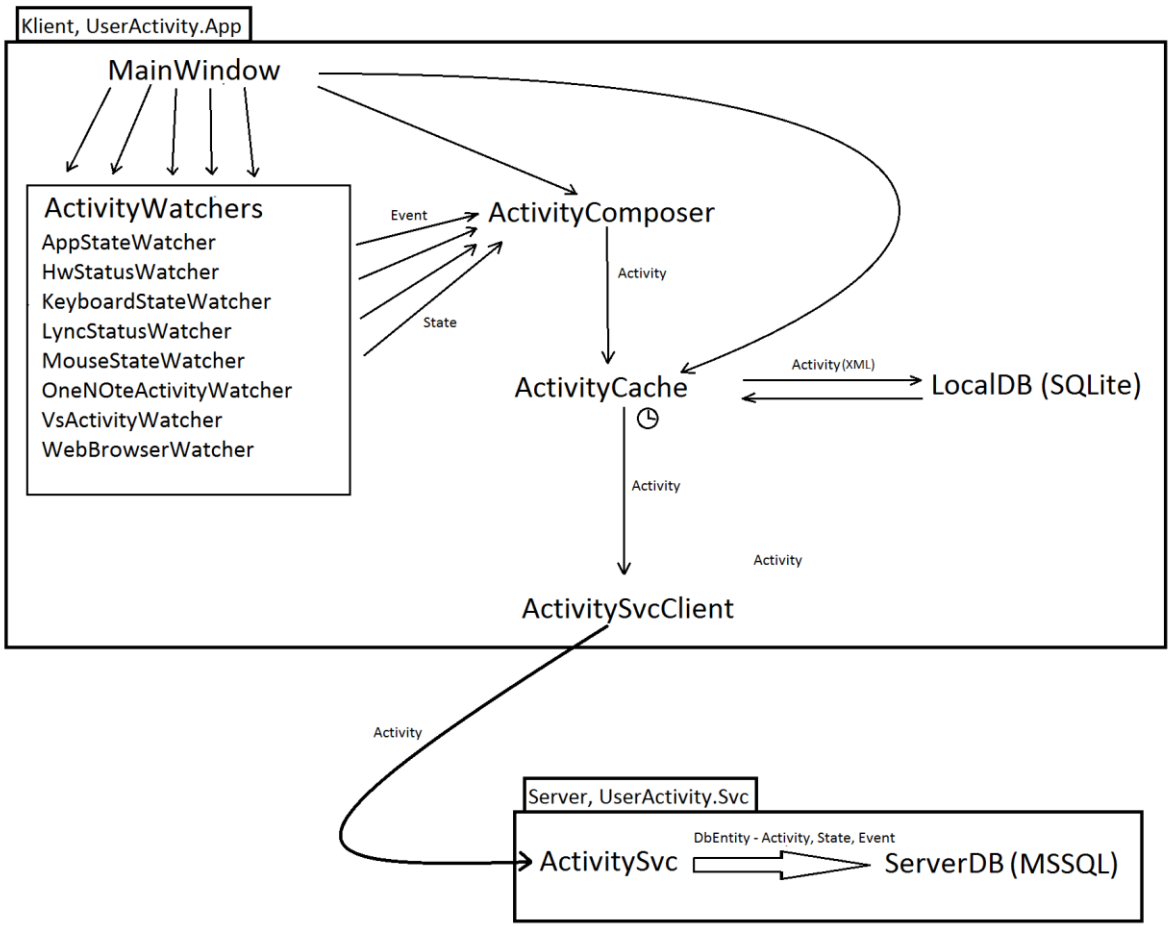
Zdrojový kód 1 Príklad aktivity zapísanej v XML

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

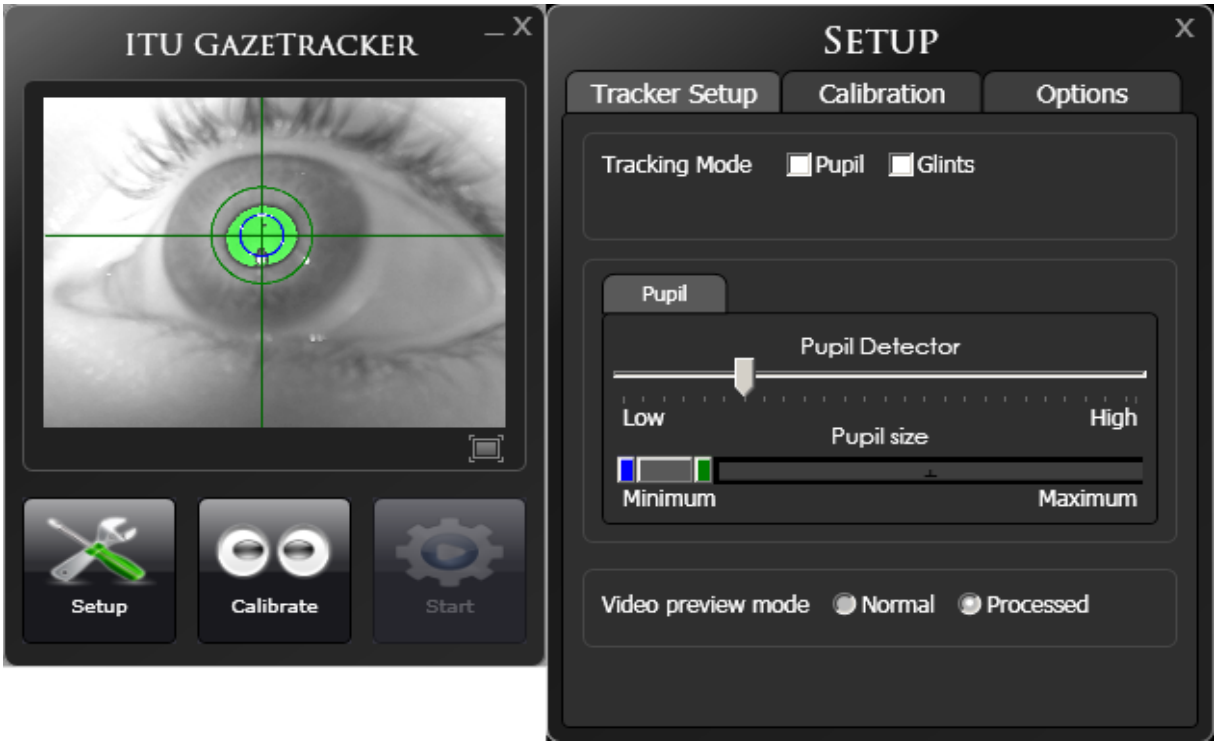
Jedná sa o nekomerčné riešenie. Spolupracuje s kamerami, ktoré obsahujú infračervené diódy ale aj s takými, ktoré ich nemajú.

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 [4]

2.1.2.2 Camera Mouse 2012

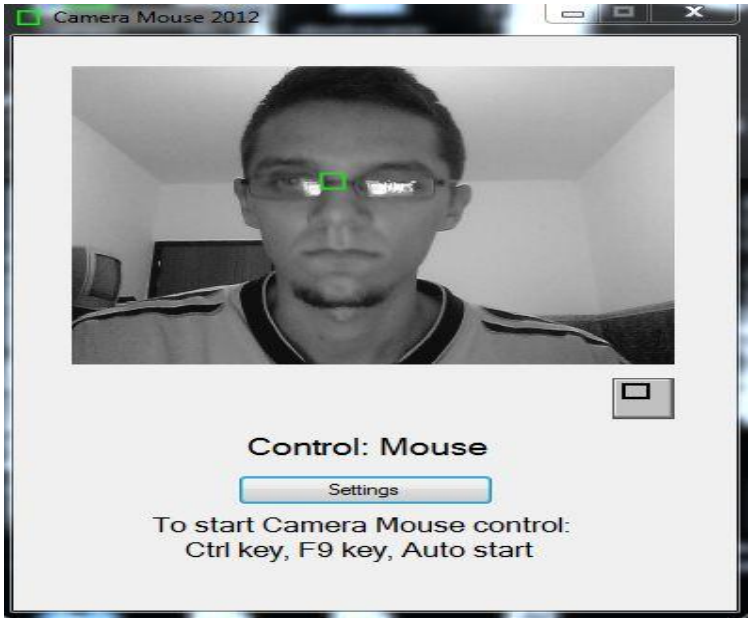
Taktiež nekomerčné riešenie, ktoré sa zameriava na ovládanie kurzora myši pomocou pohľadu.

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 dispozícii
- 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

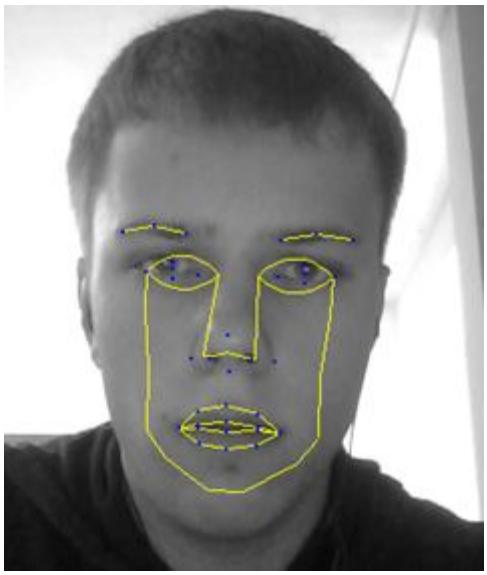
umiestnenie kamery. Lepšie výsledky boli dosahované pri externých kamerách, ktoré boli umiestnené na stole pod úrovňou tváre používateľa na rozdiel od vstavaných kamier v obrazovke. Ďalšími aspektmi sú vhodné osvetlenie či kvalita záznamu, ktorý kamera produkuje.

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

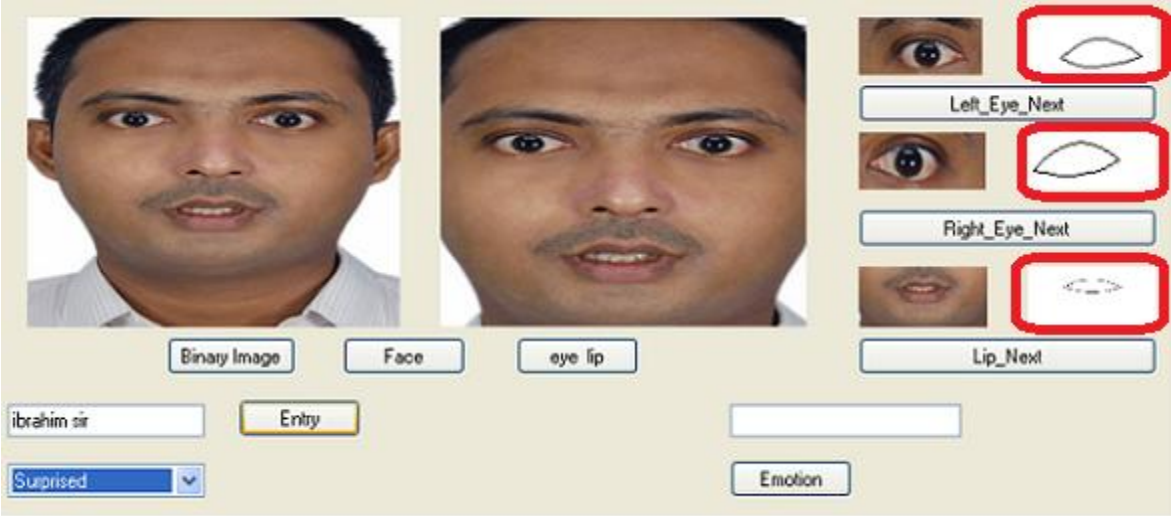


Obrázok 6 Spracovanie tváre pomocou FaceAPI

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

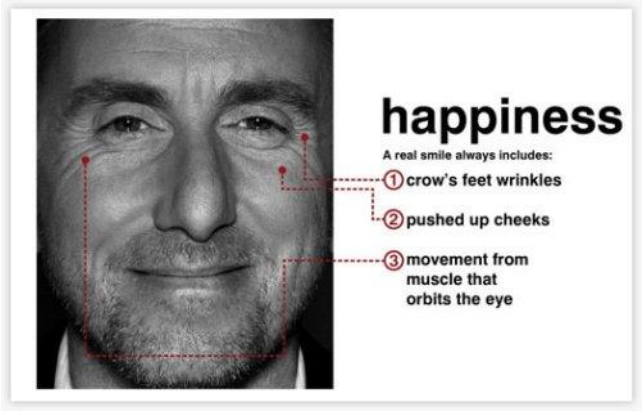
Existujúce rozpoznateľné emočné stavy:

- 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

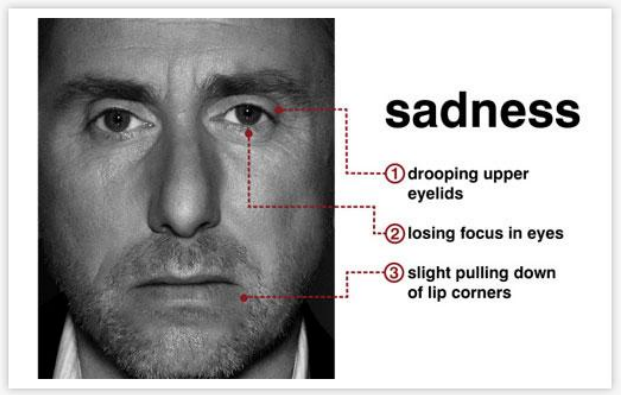


Obrázok 9 Radosť

2.1.4.2 Smútok

Špecifické znaky:

1. Mierne padnuté kútiky úst
2. Padnuté horné viečko

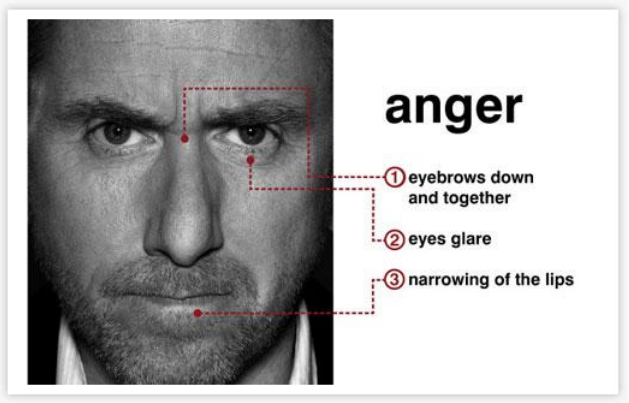


Obrázok 10 Smútok

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

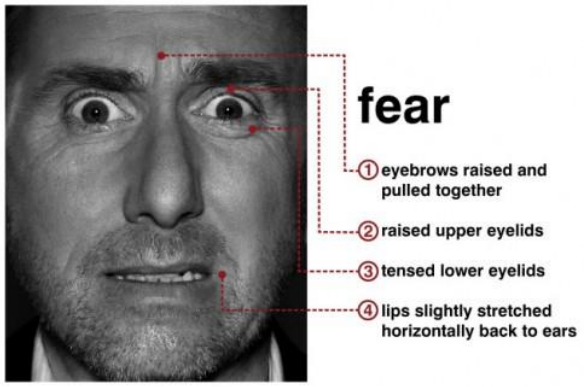


Obrázok 11 Hnev

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

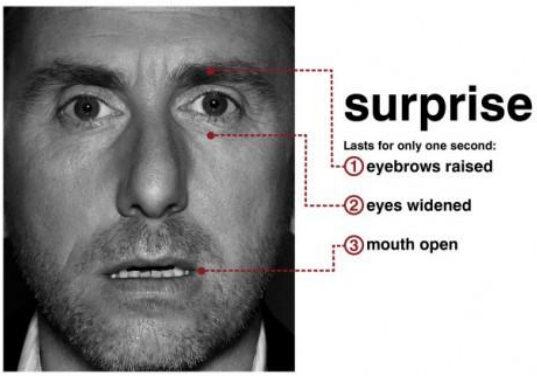


Obrázok 12 Strach

2.1.4.5 Prekvapenie

Špecifické znaky:

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

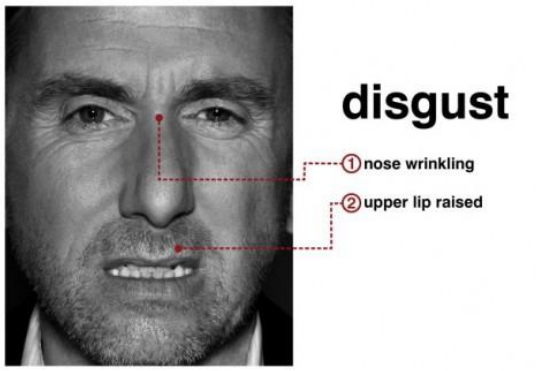


Obrázok 13 Prekvapenie

2.1.4.6 Znechutenie

Špecifické znaky:

1. Vydvihnutie hornej pery
2. Vrásky medzi očami

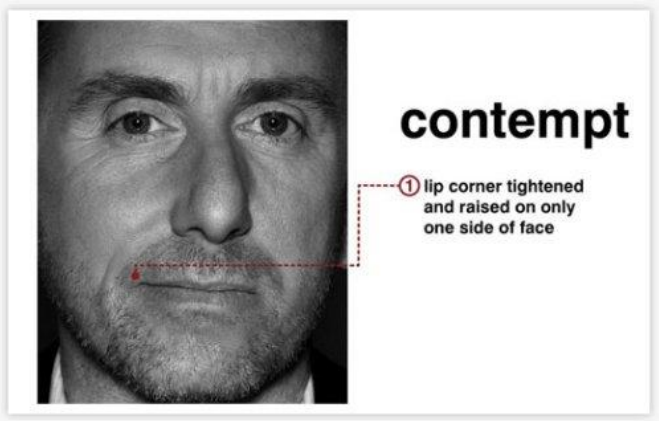


Obrázok 14 Znechutenie

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.

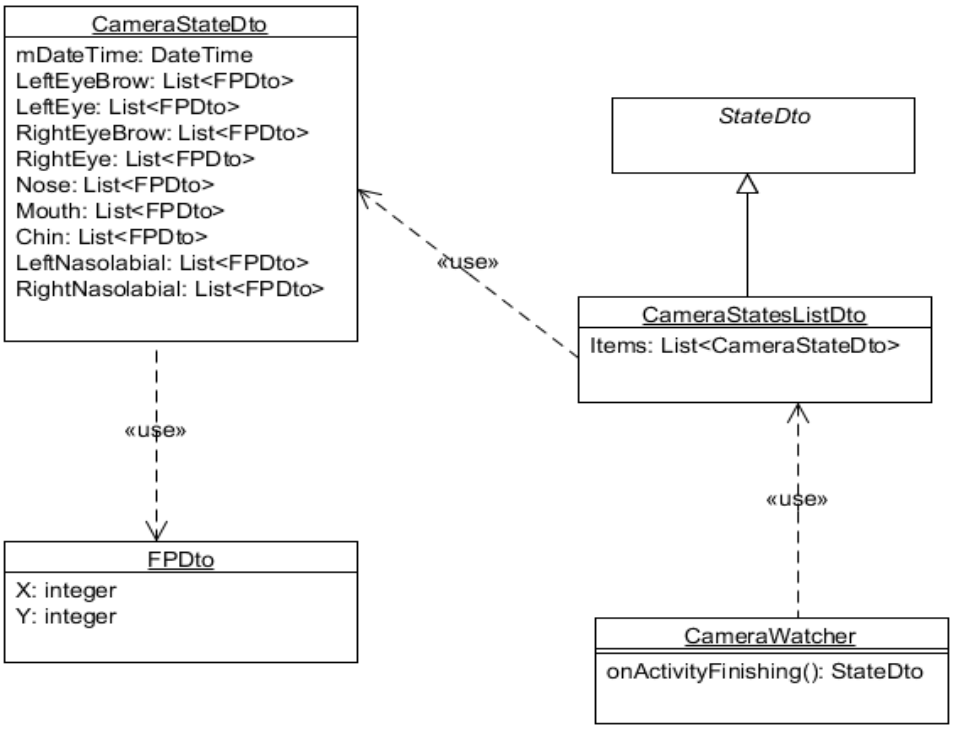
Výsledná podoba dát vyplynula z použitého nástroja Luxand SDK. Luxand FaceSDK rozoznáva 66 bodov tvare, ktoré reprezentuje dvoma číslami: x-ová súradnica a y-ová súradnica. Jednotlivé body boli rozdelené do zoznamov podľa toho akú časť tvare opisujú (ľavé oko, pravé oko, ústa, nos, brada, ľavé obočie, 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>
```

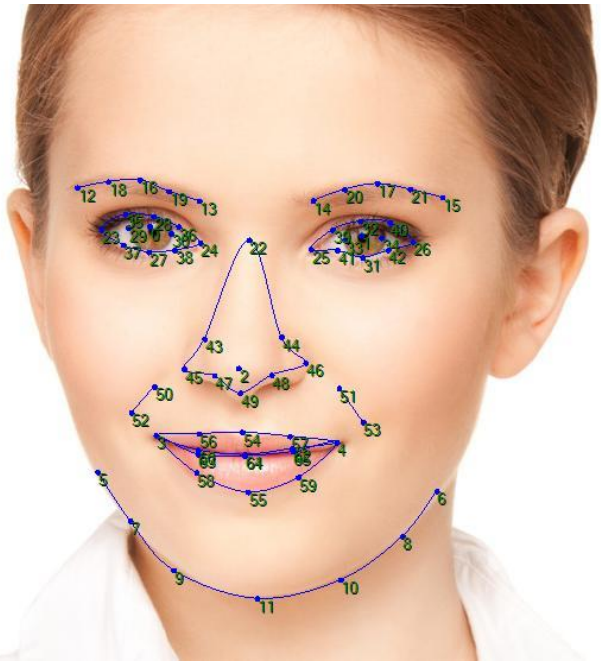
Zdrojový kód 3 XML formát zobrazovaných dát

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

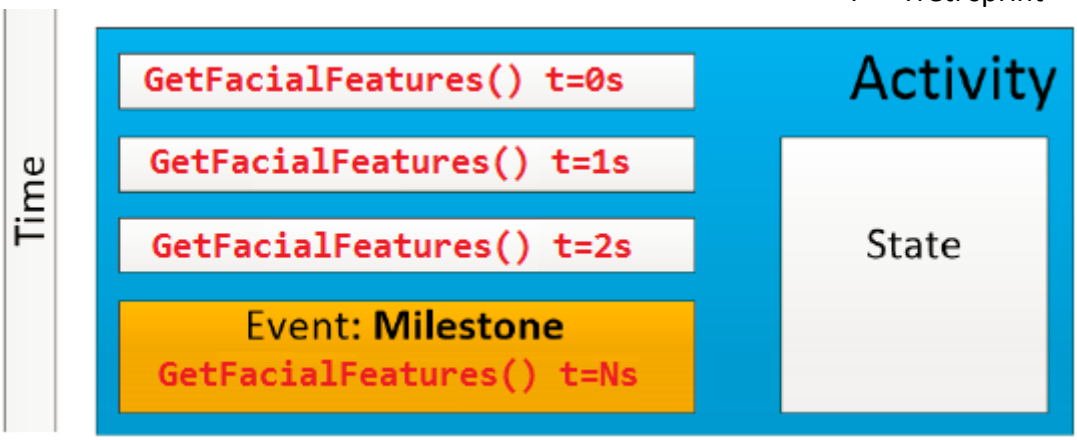
Rozšírenie úlohy Spracovať obraz pomocou Luxandu z predošlého šprintu o získavanie bodov každú sekundu.

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 spondej 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.

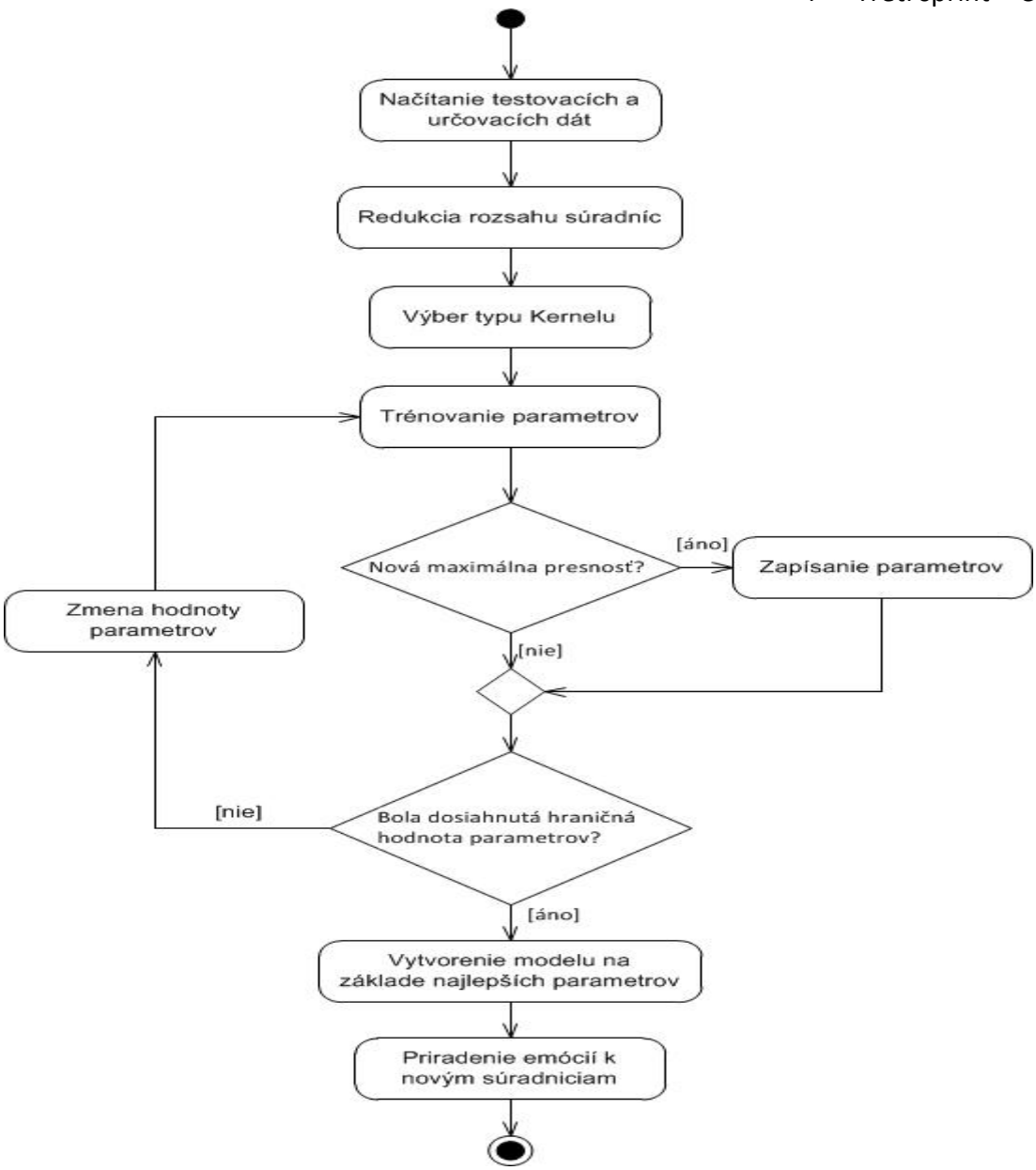
4.3.3 Implementácia

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 Database10[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

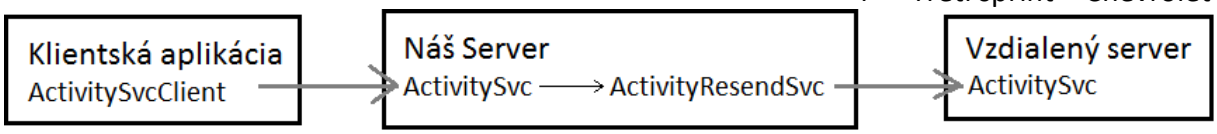
Pôvodný program - logovač, na ktorého základe vytvárame náš projekt, odosiela údaje na vzdialený server. Tento projekt sme zmenili tak, že zaznamenané dáta sa odosielajú na náš server. Tvorcovia pôvodného programu by však boli radi, aby sa údaje zaznamenané našim riešením posielali aj do ich databázy.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

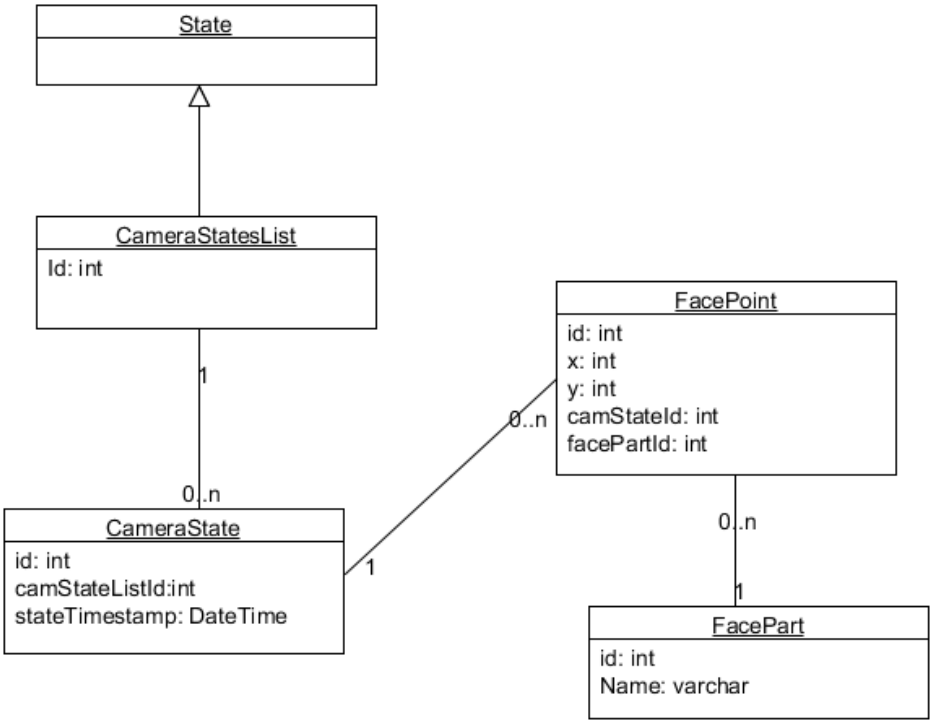
V súbore Web.config, pre *connectionString AnonymizationModelContainer* a *UserActivityContainer* nastavte prihlasovacie údaje do databázy.

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

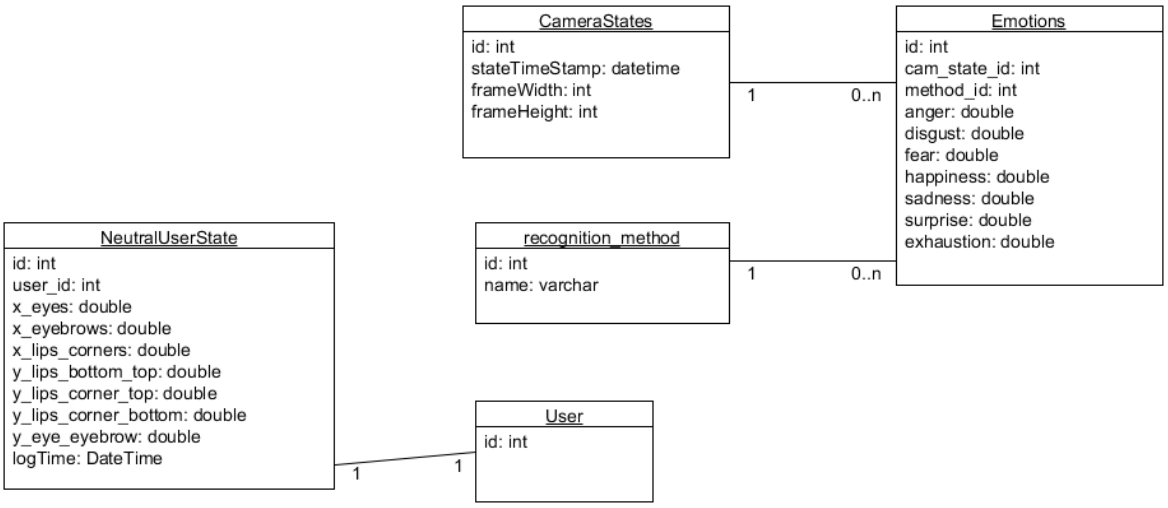
Cieľom tejto úlohy bolo navrhnúť a vytvoriť databázový model pre neutrálny emocionálny stav používateľa a pre vypočítané emócie.

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

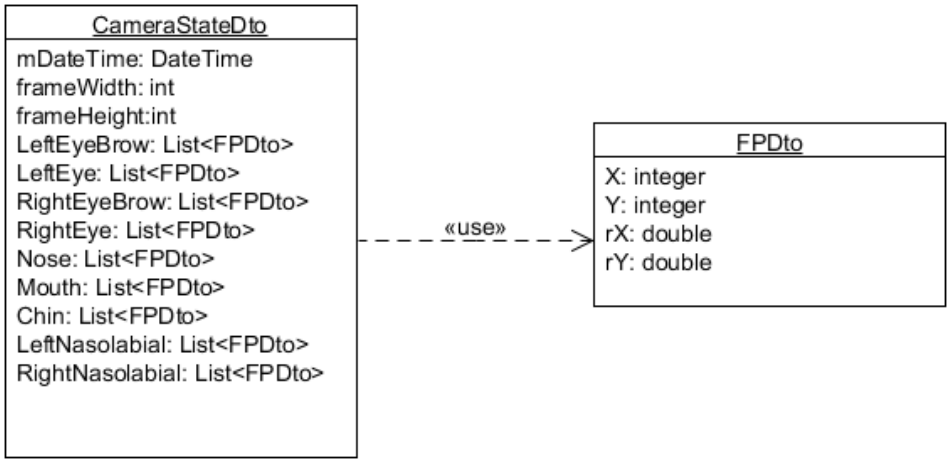
5.3 Normalizácia v loggeri

5.3.1 Analýza

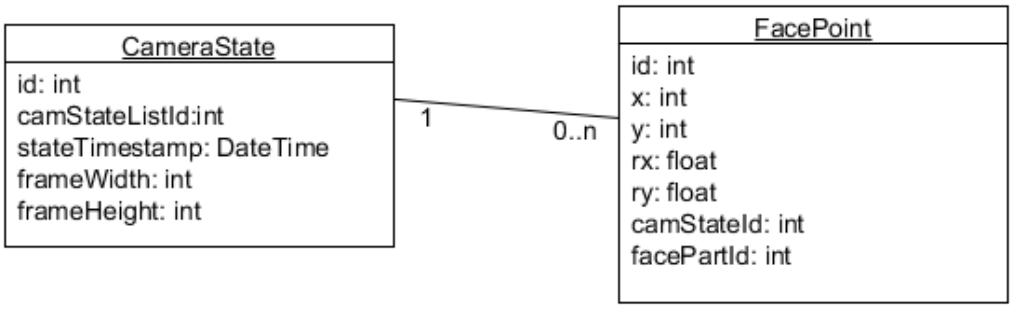
Pre potreby niektorých metód na výpočet emocionálneho stavu (neurónové siete) je potrebné získať normalizované dáta z kamery. Bolo potrebné navrhnúť metódu vyseknutia tváre z obrazu a prepočítania súradníc bodov na nový interval. Takýmto spôsobom sme eliminovali rozdiely v súradniciach pri rôznej vzdialenosti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

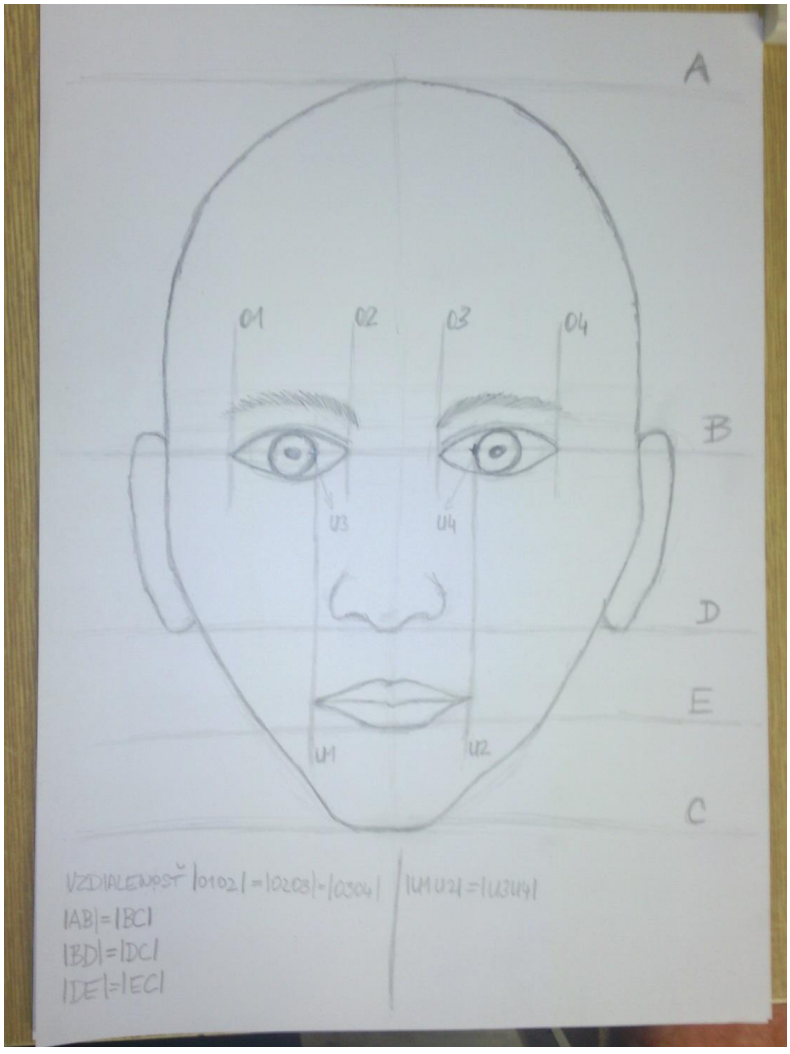
Určovanie emócií pomocou neurónových sietí realizujeme vytvorením programu s dvoma funkciami. Prvá bude vytvorenie neurónovej siete, teda naučenie na známych trénovacích

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 Piaty šprint – Ferrari

V piatom šprinte sme riešili úlohy spojené s dopytovaním stavu a odporúčaním akcie používateľovi. Taktiež sme dokončovali niektoré úlohy z predchádzajúceho šprintu. Zoznam úloh riešených v tomto šprinte:

- Kostra odporúčania
- Kombinácia riešení
- Dopytovač
- LibSVM integrácia
- Pripraviť prezentáciu (PeWe)
- Spojazdniť stránku tímu
- Skontrolovať a uzavrieť posledný šprint
- Strojové učenie prostredníctvom Neurónových sietí
- Získanie neutrálneho stavu používateľa

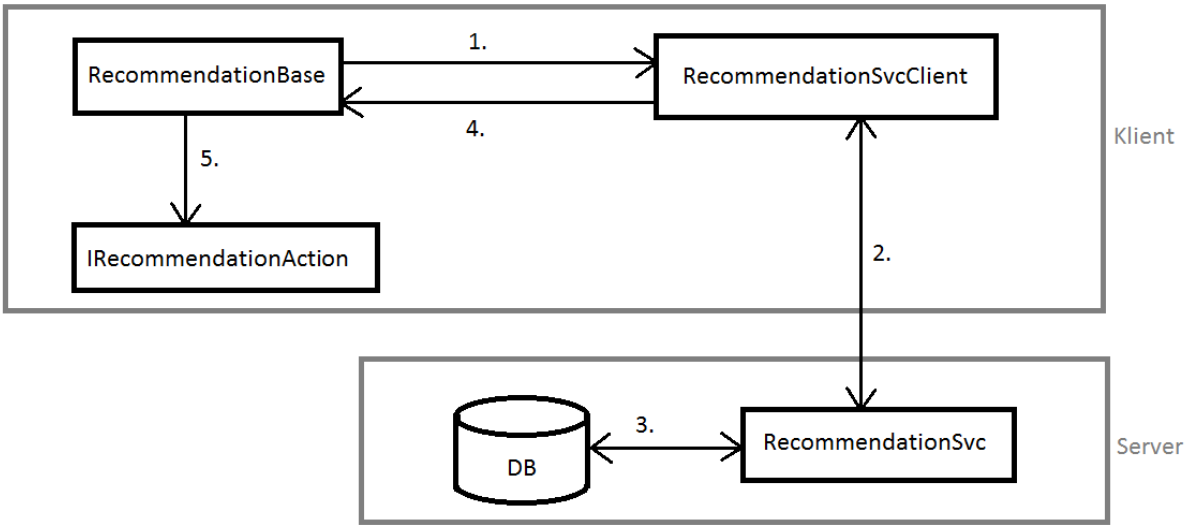
6.1 Kostra odporúčania

6.1.1 Analýza

Odporúčanie je vypočítavané z emócií na strane servera. Kvôli tomu, aby sa na základe odporúčania mohla vykonať akcia na strane klienta, je potrebné vytvoriť spojenie medzi klientom a serverom.

Spojenie so serverom je zabezpečované pomocou dvojice *ActivitySvcClient* (na strane klienta) a *ActivitySvc* (na strane servera). Toto spojenie a zvlášť služba ActivitySvc, ktorá sa stará o ukladanie aktivít do databázy, môže byť pri väčšom počte požiadaviek zahltená. Preto je vhodné oddeliť odporúčania od aktivít a vytvoriť novú službu, ktorá by mohla bežať v samostatnom vlákne (tzv. *Application Pool*). Toto rozdelenie je vhodné aj z hľadiska vývoja aplikácie – chyba v odporúčaniach nezablokuje zbieranie údajov od používateľa.

6.1.2 Návrh



Obrázok 27 Schéma komunikácie objektov pri odporúčaní

Obrázok 27 reprezentuje schému, ako medzi sebou komunikujú jednotlivé objekty zabezpečujúce odporúčanie.

Na klientovi beží vlákno *RecommendationBase*, ktoré podľa určitých kritérií rozhoduje, či je potrebné zobraziť používateľovi nejaké odporúčanie.
RecommendationScvClient a *RecommendationSvc*, ktoré zabezpečujú komunikáciu klienta so serverom. Služba *RecommendationSvc* je spustená na serveri a je nezávislá od služby *UserActivitySvc*.

RecommendationSvc komunikuje priamo s databázou (alebo inou službou) a posiela klientovi odpovede na požiadavky.

IRecommendationAction je interfejs, ktorý implementujú všetky akcie (napr. ChangeThemeAction, AdviceAction). Akcia je samostatná trieda, ktorá obsahuje metódu Perform.
Postup získania odporúčania a vykonanie príslušnej akcie:

1. Vlákno *RecommendationBase* v pravidelnom intervale kontroluje, či je potrebné odporučiť niečo používateľovi. Ak je potrebné niečo odporučiť, *RecommendationBase* zavolá metódu RecommendationSvcClient.*GetRecommendation.*
2. Je vytvorené spojenie klient-server.
3. Služba *RecommendationSvc* získa vhodné odporúčanie pre používateľa a odošle na klientský počítač odpoveď typu *RecommendationDto*.
4. Odpoveď sa vráti do *RecommendationBase*, kde sa na základe tohto odporúčania vypočíta, ktorú akciu treba spustiť používateľovi.
5. Zavolá sa vykonanie príslušnej akcie pomocou metódy IRecommendation.Perform()

6.2 Dopytovač

(časť dokumentácie prebratá od tímu 10)

V tejto časti sme sa sústredili na analýzu zdrojového kódu pri vytváraní logov používateľových akcií, ich uložení na lokálnu databázu, následnom odosielaní na server a pridania emocionálneho stavu používateľa do týchto logov.

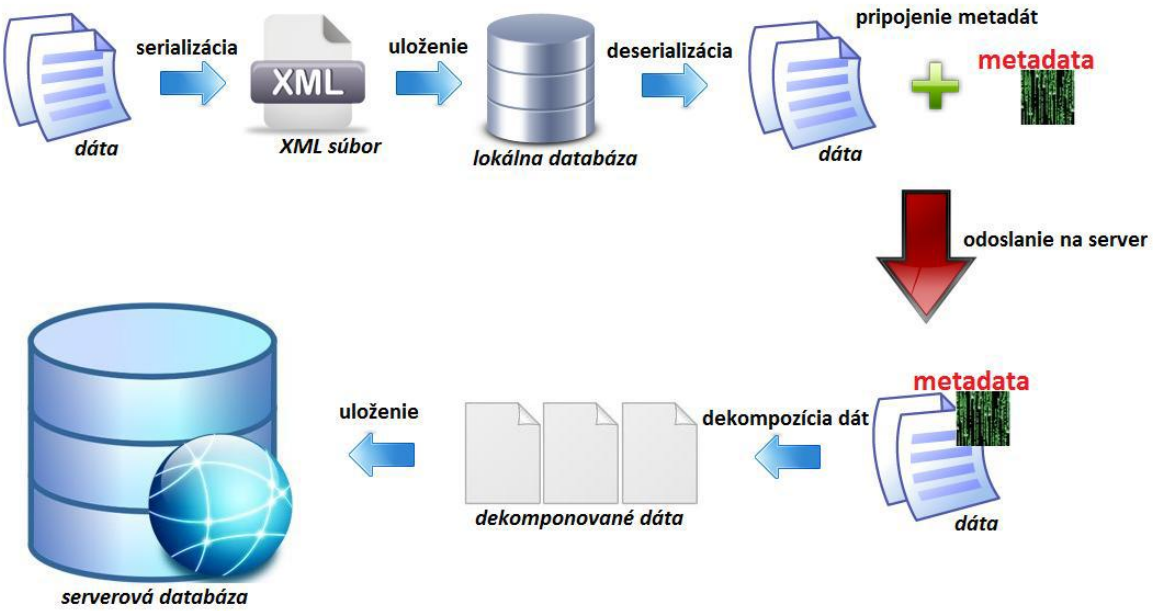
6.2.1 Analýza

Z analýzy vyplýva, že každá sledovaná aktivita (stav klávesnice, myši) je zaznamenávaná triedou zdedenými od rozhrania *IActivityWatcher* (napr. *KeyboardStateWatcher*, *MouseStateWatcher*). Rozhranie *IActivityWatcher* predpisuje metódu *OnActivityFinishing* vracajúcu objekt typu *StateDto* nesúci konkrétny stav. Jednotlivé *Watcher-y* vytvárajú špecializované objekty zdedené od abstraktnej triedy *StateDto* (napr. *KeyboardStateDto*, *MouseStateDto*). Tieto jednotlivé objekty sú následne serializované do formátu *XML* uložené do lokálnej databázy. *Serializácia* vytvára *XML* na základe názvov atribútov dátových objektov *<Type>StateDto* a ich hodnôt.

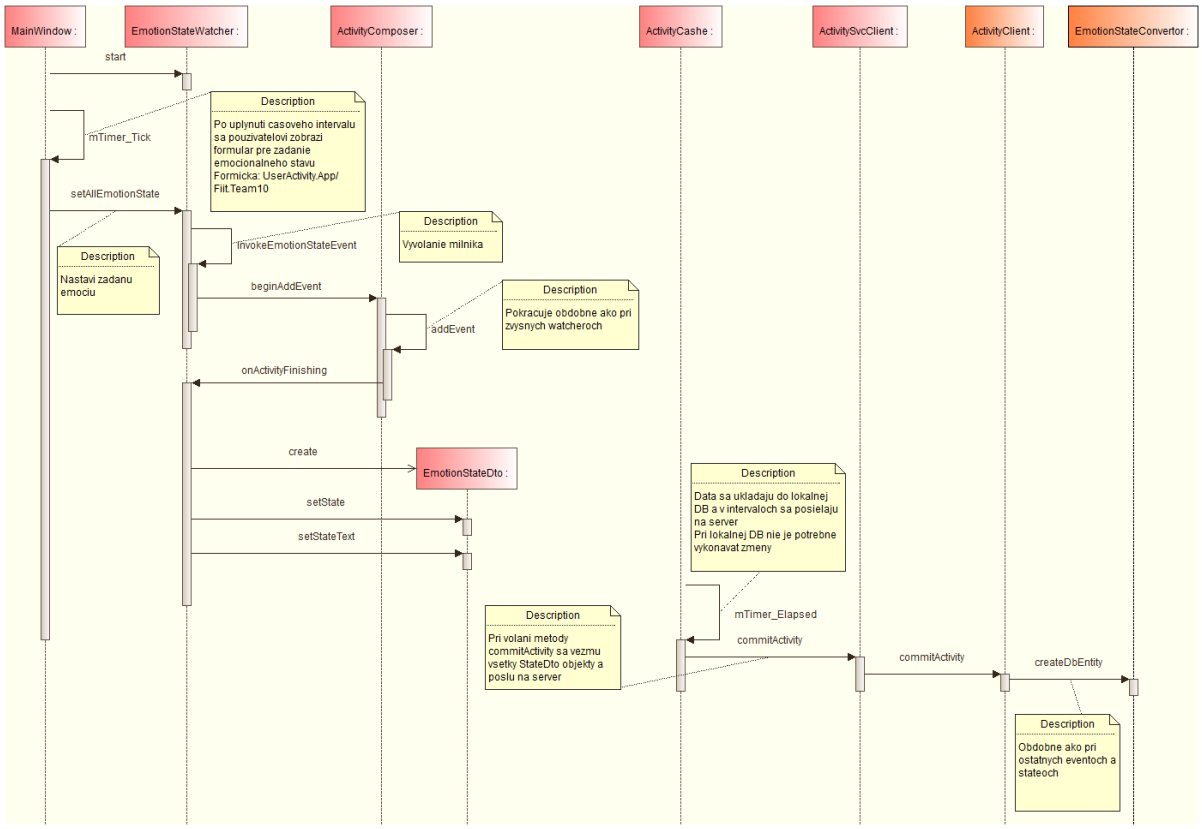
Lokálna databáza pozostáva s jednej tabuľky, ktorej stĺpce sú nasledovné: *id (TEXT PRIMARY KEY)*, *StartTime (DataTim)*, *EndTime (DataTime)* a *Data (TEXT).* Začiatočný a koncový čas nesú informáciu o trvaní konkrétneho logu a v položke *Data* sa nachádza samotný log uložený vo formáte *XML*.

Po uplynutí časového intervalu sa dáta z lokálnej databázy odošlú na server. Časový interval je nastavený v súbore *app.config* s prednastavenou hodnotou 30 minút. V tomto súbore je uložená aj adresa servera, na ktorý sa dáta odosielajú. Po uplynutí stanoveného intervalu sa vyberú dáta z lokálnej databázy, deserializujú sa na objekty typu *ActivityDto*, ktorá obsahuje kolekciu objektov *<Type>StateDto* a pridá metadáta (začiatočný čas, koncový čas, názov pracovnej stanice, identifikátor používateľa). Odosielanie týchto dát je realizované prostredníctvom volania vzdialenej metódy *CommitActivity*, ktorej argumentom je objekt typu *ActivityDto*. V tejto metóde sa dáta skonvertujú na objekty, ktoré automaticky ukladajú hodnoty ich atribútov do databázy.

Autentifikácia používateľa je realizovaná pomocou atribútov triedy *ActivityDto* - názov pracovnej stanice a identifikátor používateľa.



Obrázok 28 Tok získaných dát o používateľovi



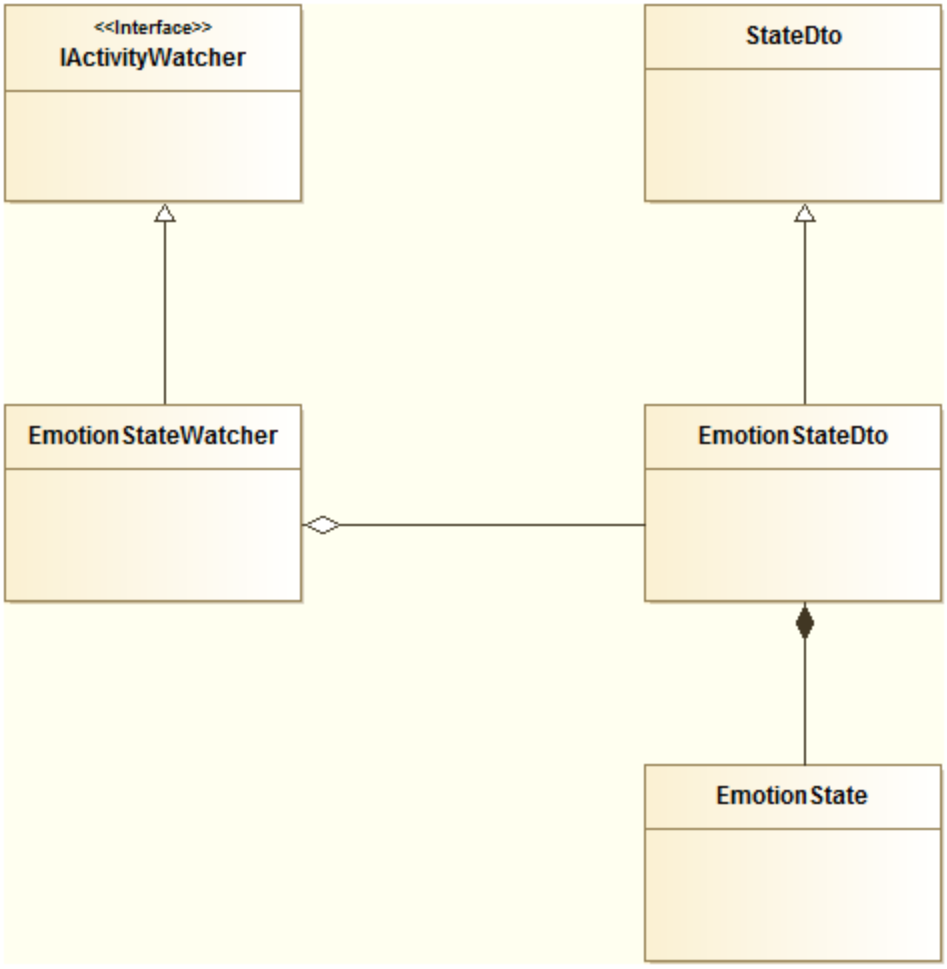
Obrázok 29 Implementácia logovania emocionálneho stavu používateľa

6.2.2 Návrh

Na základe analýzy navrhujeme vytvoriť triedu *EmotionStateWatcher* zdedenú od rozhrania IActivityWatcher, ktorá bude uchovávať aktuálny emocionálny stav používateľa. Ten to stav sa zmení po zadaní nového stavu používateľom v *GUI* časti. Stav uvedený v tejto triede je potrebné odovzdať predpísanou metódou *OnActivityFinishing* v objekte typu *StateDto*. Preto je potrebné vytvoriť triedu *EmotionStateDto* zdedenú od *StateDto*. Táto trieda bude obsahovať jeden argument typu enumeračný typ, ktorý môže nadobúdať hodnoty (Radosť, Normálny, Stres, Únava, Hnev). Informácia o emocionálnom stave používateľa sa automaticky pridá do *XML* súboru, uloží do lokálnej databázy a odošle na server. Tam bude v nasledujúcom kroku potrebné pridať položku emocionálny stav do databázy a pridať konvertor k týmto dátam.

6.2.3 Implementácia

Návrh bol realizovaný implementáciou uvedených tried *EmotionStateWatcher*, *EmotionStateDto* a enumeračného typu *EmotionState*.



Obrázok 30 Implementácia logovania emocionálneho stavu používateľa

6.3 LibSVM integrácia

6.3.1 Analýza

Pri určovaní emócie kladieme v našom projekte dôraz na adekvátnu dynamiku získavania emočného stavu. Preto je potrebné aby sa časti zodpovedné za určovanie emócie vykonávali v najkratšom možnom čase, pričom nebude redukovaná presnosť takéhoto riešenia.

V rámci strojového učenia pomocou metódy SVM(support vector machine) je kritickou časťou v súvislosti s časovou náročnosťou vytvorenie modelu z množiny dát určených na učenie. Ostatné časti strojového učenia prebiehajú rádovo v mikrosekundách, pričom vytváranie modelu trvá niekoľko minút. Dĺžka trvania vytvorenia modelu je samozrejme závislá na množstve dát, z ktorých sa daný model vytvára.

Redukcia časovej náročnosti môže byť dosiahnutá pomocou pripravenia externého modelu, ktorý by sa používal pri strojovom učení. Otázny však ostáva spôsob, pomocou ktorého by sme boli schopný vytvorený model uložiť pre neskoršie používanie.
Programovací jazyk C# ponúka nasledovné spôsoby:

- Možnosť ukladania objektov do súboru
- Ukladanie premenných do *project settings*

Pri prvej možnosti sa objavujú komplikácie pri načítaní takto uloženého objektu. V rámciu druhej možnosti je potrebné deklarovať typ objektu. V rámci svm typov sa však nedala deklarovať premenná typu Model, ktorú by sme potrebovali. Nakoniec sme sa dopracovali k metódam triedy Model, pomocou ktorých sa dá model uložiť ale aj čítať zo súboru. Dopracovanie sa k tomuto spôsobu bolo problematické najmä z dôvodu slabej dokumentácie knižnice pre SVM.

6.3.2 Návrh

Princíp zrýchlenia procesu určenia emócie pomocou metódy SVM bude spočívať v používaní modelu, ktorý bude pripravený a uložený ešte pred samotným určovaním emócie. V rámci tejto úlohy sa zatiaľ bude používať koncepcia vytvorenia modelu zo získaného datasetu bez možnosti obohacovania tohto modelu o nové údaje. Princíp spúšťača, ktorý spúšťa určovanie emócií si vyžaduje aby bolo riešenie vytvorené ako spustiteľný súbor, ktorý zapíše výsledky do súboru.

6.3.3 Implementácia

Na implementovanie navrhnutého riešenia sa budú používať metódy triedy Model, ktoré poskytuje knižnica libsvm. Konkrétne sa jedná o metódy *write*(model,názov súboru) a *read*(názov súboru). Celé riešenie určenia emócie pomocou SVM bude vytvorené v podobe spustiteľného súboru.

6.3.4 Testovanie

Otestovanie riešenia spočívalo v overení funkčnosti mimo vývojového prostredia, len za pomoci spustiteľného súboru a priloženia potrebných súborov. Toto overenie prebehlo úspešne.

6.4 Strojové učenie prostredníctvom Neurónových sietí

Analýzu a návrh tejto úlohy sme vykonali už v štvrtom šprinte, pre problémy s pôvodne vybratou knižnicou sme však túto úlohu presunuli do piateho šprintu a zmenili sme knižnicu, v ktorej implementujeme neurónovú sieť.

6.4.1 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.

6.4.2 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.

6.4.3 Implementácia

Neurónovú sieť pre rozpoznávanie emócií sme implementovali pomocou knižnice Encog Machine Learning Framework, ktorá podporuje množstvo algoritmov strojového učenia. Väčšina trénovacích algoritmov tejto knižnice je viacvláknová, čo umožňuje efektívne využiť viacjadrové systémy.

Pomocou tejto knižnice sme implementovali prototyp neurónovej siete, s ktorým sme následne experimentovali a snažili sme sa dosiahnuť najlepšie výsledky pri trénovaní tejto siete. Sieť sme trénovali pomocou metódy TrainToError.

```
EncogUtility.TrainToError(network, new BasicMLDataSet(input, ideal), errorTreshold);
```

Zdrojový kód 12 Trénovanie neurónovej siete

Táto metóda vykonáva trénovanie, pokým chybovosť siete neklesne pod stanovenú hranicu. Vstupmi tejto metódy sú sieť, ktorú chceme trénovať, trénovacie dáta spolu so správnymi výstupmi a hranica žiadanej chybovosti.

Experimentovaním s trénovaním sme určili štruktúru siete, pozostávajúcu zo vstupnej, jednej skrytej a jednej výstupnej vrstvy. Vstupná vrstva pozostáva zo 132 vstupných neurónov, výstupná zo 6 výstupných neurónov a skrytá vrstva z 30 neurónov. Počet neurónov vstupnej a výstupnej vrstvy sme určili podľa počtu vstupných a výstupných parametrov a počet neurónov v skrytej vrstve experimentovaním.

Ako hranicu pre naučenie siete sme zvolili 6% chybovosť, pretože od tejto hranice už trénovanie prebiehalo pomaly. Po skončení trénovania siete je vytvorený binárny súbor network.eg, ktorý obsahuje natrénovanú neurónovú sieť, použiteľnú na klasifikáciu nových záznamov.

Pre prácu s neurónovou sieťou sme vytvorili rozhrania umožňujúce trénovanie siete a klasifikáciu nových záznamov. Pre dlhé trvanie trénovanie sme sa však zatiaľ rozhodli, že používanú sieť natrénujeme iba raz a následne budeme používať už iba tento model, bez jeho opätovného učenia. V budúcnosti je ešte možné vrátiť sa k tejto úlohe a pokúsiť sa o lepšie vyladenie štruktúry neurónovej siete, ktorá by umožňovala rýchlejšie učenie.

6.5 Zhrnutie šprintu

V piatok šprinte sme dokončili niektoré úlohy, ktoré sme nestihli dokončiť vo štvrtom šprinte a zároveň sme začali uvažovať o kombinácii nášho riešenia s riešením tímu číslo 10. Podarilo sa nám dokončiť strojové učenie prostredníctvom neurónových sietí a integrovali sme ho, spolu s metódou SVM do spúšťača, ktorý však musíme v ďalšom šprinte upraviť pre prácu s dll súbormi.

7 Šiesty šprint – Honda

V šiestom šprinte sme pokračovali na úlohách spojených s odporúčaním používateľovi a na úprave spúšťača, ktorý riadi spúšťanie rozpoznávanie pomocou troch zvolených metód – libSVM, neurónových sietí a neutrálneho stavu. Taktiež sme tu riešili úlohy spojené s dynamikou emócií. Nasleduje zoznam úloh riešených v tom to šprinte:

- Implementácia neutrálneho stavu do spúšťača
- Kostra spúšťača
- Príprava interface pre neutrálny stav
- Príprava interface pre libSVM
- Príprava interface pre Neurónové siete
- Vytvorenie dema projektu
- Tvorba dokumentácia č. 3
- Výber odporúčača + nastavenia
- Zmena farebnej schémy
- Odporúčanie pomocou pop-up
- Výpočet akcie na serveri na základe dynamiky emócií
- Úprava databázového modelu – denormalizácia
- Úprava dopytovača na dopytovanie podľa stavu
- Dynamika emócií
- Upraviť architektúru spúšťača

7.1 Výber akcie na základe odporúčania + nastavenia

7.1.1 Analýza

Na základe údajov zozbieraných od používateľa sa vypočíta jeho aktuálna emócia. Z emócie sa vypočíta vhodné odporúčanie a to je odoslané do klientskej aplikácie, ktorá na základe odporúčania vykoná určitú akciu. Napr. ak výsledné odporúčanie bude „daj si oddych“, tak vykonanou akciou by mohlo byť spustenie uspávanky, zobrazenie upozornenia, alebo stmavenie obrazovky,...

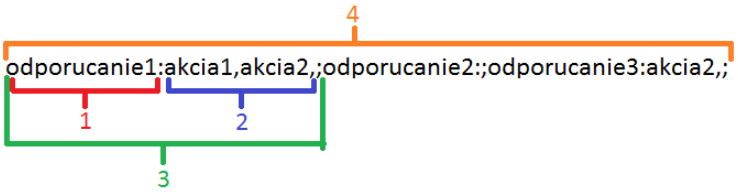
Ale ľudia sú rôzni, každému sa páči niečo iné. Niekto rád počúva hudbu pri práci, iný naopak, počas oddychu. Preto nemôžeme s istotou určiť, ktorá akcia sa bude ktorému používateľov v danej chvíli hodiť. Niektoré akcie môžu používateľovi prekážať, čo by mohlo mať za následok odinštalovanie klientskej aplikácie z počítača.
Preto by bolo vhodné, aby si používatelia mohli nastaviť aké akcie budú chcieť vykonať pri ktorom odporúčaní.

7.1.2 Návrh

Odporúčanie sa vypočítava a získava zo servera. Klient pred prvým spustením nevie aké existujú odporúčania. Zoznam aktuálnych podporovaných odporúčaní si získa zavolaním

metódy *GetSupportedRecommendations()* v službe *RecommendationSvc*. Ak sa mu nepodarí spojenie so serverom, môže si do nastavení nahrať údaje, ktoré má zapísané v kóde.

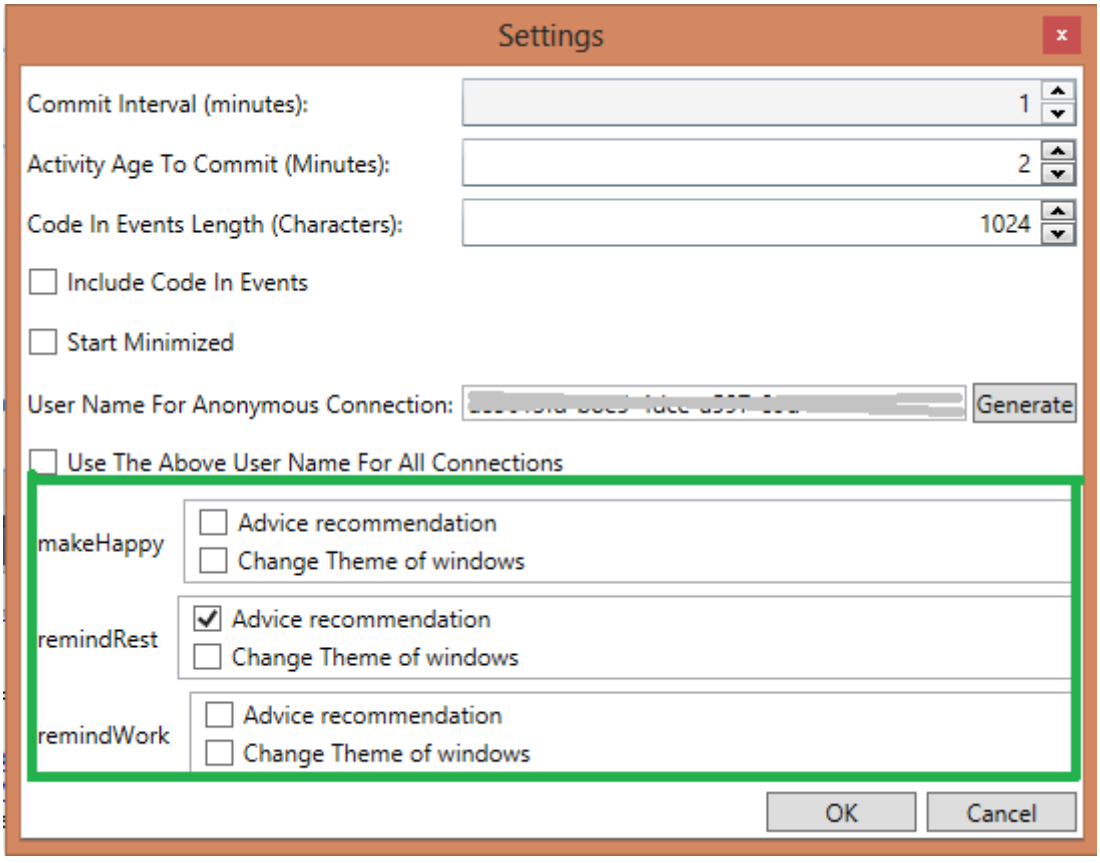
Do nastavení sa uloží reťazec, ktorý bude mať tvar ako na obrázku **Chyba! Nenašiel sa iaden zdroj odkazov.** nebude však zatiaľ obsahovať priradenie akcií k jednotlivým odporúčaniam.



Obrázok 31 Reťazec pre ukladanie nastavení odporúčaní

Obrázok 31 vysvetľuje uloženie nastavení odporúčaní. Číslami sú označené jednotlivé časti reťazca.

1 – kľúč odporúčania, 2 – zoznam akcií oddelených čiarkou, ktoré sú priradené k odporúčaniu, 3 - priradené akcie k odporúčaniu, 4 – odporúčania oddelené bodkočiarkou.



Obrázok 32 Vzor okna s nastaveniami. Zeleným obdĺžnikom sú vyznačené nastavenia odporúčaní

Na Obrázok 32 je znázornené okno s nastaveniami, kde je možné nastaviť to, ktoré akcie sa budú vykonávať pri akom odporúčaní.

7.1.3 Implementácia

Nastavenie klientskej aplikácie pre odporúčania sa nachádza v premennej *ResourcePreferences.* V nej je uložený reťazec v nasledujúcom tvare: *odporucanie1:akcia2,akcia2,;odporucanie2:;*
Nasledújúci kóde je popísané to, ako sa vykreslí časť okna s nastaveniami, kde sa nachádzajú odporúčania.

```
Dictionary<String, String> availableActions = new Dictionary<string,string>();

// získanie všetkých dostupných akcií
foreach (IRecommendationAction recommendation in ComponentCache.Default.Container.GetExportedValues<IRecommendationAction>())
{
	availableActions.Add(recommendation.GetType().Name, recommendation.GetName());
}

// vybratie reťazca s nastaveniami
String availableRecommendations = Settings.Default.RecommendationPreferences;

// rozdelenia reťazca podľa jednotlivých odporúčaní
string[] recommendationChunks = availableRecommendations.Split(';');

// pre každé odporúčanie je získaný zoznam povolených akcií
foreach(String chunk in recommendationChunks) {
	if (chunk.Length == 0) continue;
	String recommendationName = chunk.Substring(0,chunk.IndexOf(':'));

	Dictionary<String, bool> recommendationActions = new Dictionary<String,bool>();
	Array actionChunks = chunk.Substring(chunk.IndexOf(':')+1).Split(',');

	foreach(String key in availableActions.Keys) {
		bool isCheckedAction = false;
		foreach (String checkedAction in actionChunks)
		{
			if (checkedAction.Equals(key))
			{
				isCheckedAction = true;
				recommendationActions[key] = isCheckedAction;
				break;
			}
		}
	}

	// Vytvor blok v okne s nastaveniami
	RecommendationBindingItem rbi = new RecommendationBindingItem(recommendationName, recommendationActions, availableActions);
	mRecommendationBindings.Children.Add(rbi);
}
```

Zdrojový kód 13 Vykreslenie odporúčaní do okna s nastaveniami

Nasledujúci kód je periodicky vykonávaný v triede *RecommendationBase.* Opisuje spôsob akým sa získava odporúčanie zo server, následne nájdenie vhodných akcií pre toto odporúčanie a nakoniec spustenie vykonávania akcie.

```
// získaj odporúčanie zo servera
RecommendationDto
recommendationDto=RecommendationSvcClient.Default.GetRecommendation();

// získaj akcie pre dané odporúčanie
List<IRecommendationAction> enabledRecommendationActionList = null;
enabledActions.TryGetValue(recommendationDto.Type,                                    out
enabledRecommendationActionList);

if (enabledRecommendationActionList.Count != 0)
{
       Random r = new Random();

       // vyber akciu zo zoznamu
       IRecommendationAction                          action                          =
enabledRecommendationActionList.ElementAt(r.Next(enabledRecommendationActionList.Count
));
       int i = 0;
       switch(recommendationDto.Type) {
              case "makeHappy" : i=1; break;
              case "remindRest": i = 2; break;
              case "remindWork": i = 3; break;
              default: break;
       }

       // vykonaj akciu
       action.Perform(i);
}
```

Zdrojový kód 14 Získanie odporúčania, výber akcie, vykonanie akcie

7.2 Zmena farebnej schémy

7.2.1 Analýza

Zmena farebnej schémy je jedným z možných odporúčaní, ktoré môžu zmeniť používateľovu náladu. Windows ponúka možnosť zmeny farebnej témy pomocou Description of Control Panel (.cpl) súbora desk.cpl. Príkazom v konzole je možné jednoducho zmeniť tému na ľubovoľnú tému.(súbor .theme)

7.2.2 Návrh

Príkaz, ktorým je možné zmeniť tému vyzerá nasledovne:

```
rundll32.exe %SystemRoot%\system32\shell32.dll,Control_RunDLL
%SystemRoot%\system32\desk.cpl desk,@Themes /Action:OpenTheme
/file:"C:\Windows\Resources\Themes\landscapes.theme"
```

Zdrojový kód 15 Príkaz pre zmenu témy

7.2.3 Implementácia

Keďže chceme vytvoriť triedu a metódu v C#, ktorou bude možné kdekoľvek meniť farebnú tému, rozhodli sme sa použiť C# triedu Process, ktorá umožňuje spúšťať a zastavovať lokálne systémové procesy.

```
ProcessStartInfo startInfo = new ProcessStartInfo();
            startInfo.CreateNoWindow = false;
            startInfo.FileName = "rundll32.exe";
            //string startuppath = Application.StartupPath.ToString();
            startInfo.WindowStyle = ProcessWindowStyle.Hidden;
            string Arguments = "shell32.dll,Control_RunDLL desk.cpl desk,@Themes
/Action:OpenTheme /File:\"C:\\Windows\\Resources\\Themes\\nature.theme\"";
            startInfo.Arguments = Arguments;
            try
            {
            // Start the process with the info we specified.
            // Call WaitForExit and then the using statement will close.
            Process exeProcess = Process.Start(startInfo);
```

Zdrojový kód 16 Zmena témy

Nevýhodou riešenia, je že po zmene témy vyskočí okno Personalize, ktoré je následne zatvorené. Toto správanie, môže byť pre používateľa obťažujúce.

7.3 Odporúčanie pomocou pop-up

7.3.1 Analýza

Cieľom nášho projektu je na základe zisteného emočného stavu používateľa tento stav, pokiaľ je nevyhovujúci, vhodným spôsobom napraviť. Jednou z možností ako poskytnúť používateľovi takéhoto systému, spätnú väzbu je napríklad pomocou vyskakovacieho (pop-up) okna.

7.3.2 Návrh

Odporúčač na základe stavu používateľa rozhodne o akcii, ktorá sa uskutoční za účelom nápravy tohto stavu. Jednou z týchto akcií je aj zobrazenie odporúčania pomocou vyskakovacieho okna.

7.3.3 Implementácia

Riešenie je zatiaľ v prvotnej fáze, keď ešte nie je známe čo presne ma obsahovať odporúčanie za účelom zlepšenia emočného stavu. Vstupom riešenia je číslo stavu, ktorý chceme riešiť. K tomuto stavu prislúcha množina oznámení, z ktorej sa náhodne vyberie jedno, ktoré sa zobrazí vo vyskakovacom okne. Obsah oznámení je zatiaľ len jednoduchý a neobsahuje žiadne odborné odporúčania.

7.3.4 Testovanie

Riešenie bolo otestované pomocou jednoduchého scenára, ktorého predmetom bolo zaslanie stavu zo servera práve tomuto riešeniu a očakávanou akciou bolo zobrazenie príslušného vyskakovacieho okna. Tento testovací scenár bol v rámci testovanie naplnený.

7.4 Úprava dopytovača na dopytovanie podľa stavu

7.4.1 Analýza

Dopytovač je potrebné upraviť, aby sme v prípade veľkej nezhody našich metód na rozpoznávanie emócií mohli používateľa spýtať na jeho emóciu. Čiže dopytovač nemôže otázku pokladať automaticky, ale v akomkoľvek čase.

7.4.2 Návrh

Pre komunikáciu so serverom sme použili už vytvorenú službu RecommendationSvc. Trieda RecommendationBase kontaktuje server v pravidelných intervaloch a pýta sa servera, či je potrebné vytvoriť formulár a spýtať sa používateľa na jeho stav.

7.4.3 Implementácia

Implementáciu formulára na pýtanie sa používateľa sme prebrali od tímu 10. Do triedy RecommendationSvc sme doplnili metódy GetEmotion, ktorá serveru vráti emóciu, korú používateľ vyklikne vo formulári.(EmotionAskForm)

7.5 Upraviť architektúru spúšťača

7.5.1 Príprava interface pre Neurónové siete

Cieľom tejto úlohy bolo vytvoriť interface pre používanie neurónovej siete a vytvorenie dll knižnice, s ktorou by mohol pracovať upravený spúšťač. Interface sme vytvorili už ako súčasť predchádzajúcej úlohy Strojové učenie pomocou Neurónových sietí, preto sme v tejto úlohe už iba rozdelili a upravili projekt tak, aby z neho bolo možné vytvoriť dll knižnicu.

```
/// <summary>
/// Train neural network on specified training data to specified error
treshold.
/// </summary>
/// <param name="input">Array of input values - double values representing face
points</param>
/// <param name="ideal">Array of ideal values - emotion representation</param>
/// <param name="errorTreshold">Error treshold to train a network</param>
void trainNeuralNetwork(double[][] input, double[][] ideal, double errorTreshold);

/// <summary>
/// Classify input data - compute emotional state
/// </summary>
/// <param name="input">Array of double values - double values representing face
points</param>
/// <returns>Array of double values - computed emotional state</returns>
double[] classifyInputData(double[] input);
```

Zdrojový kód 17 Interface pre trénovanie a používanie neurónovej siete

7.5.2 Príprava interface pre libSVM

7.5.2.1 Analýza

Zmena koncepcie fungovania spúšťača si vyžaduje zmeny aj v riešení, ktoré zabezpečuje strojové učenie pomocou SVM. Konkrétne sa jedná o kooperáciu s daným riešením pomocou dll súboru a nie pomocou spustiteľného súboru.

Ďalším problémom je princíp, na ktorom je založené fungovanie SVM riešenia. Vstupno-výstupné operácie pracujú s dátami, ktoré sú prítomné v súboroch. Vzhľadom na skutočnosť, že náš projekt je vyvíjaný s cieľom použitia v klient-server architektúre, tak práca so súbormi v tomto riešení by bola problematická jednak z hľadiska riadenia prístupu k týmto súborom ale aj z hľadiska časovej a pamäťovej náročnosti práce s nimi.

Najmä kvôli slabému zdokumentovaniu používanej knižnice je problematické prísť na alternatívne riešenia. Podarilo sa však nájsť metódu na načítanie vstupných údajov, ktorá akceptuje údaje typu *Stream*, oproti predtým používanému súboru. Vzhľadom na skutočnosť, že ostatné metódy, ktoré používame na určovanie emócie vrátia pravdepodobnosť príslušnosti ku každej emócii by bolo vhodné ak by sme sa k takémuto rozdeleniu vedeli dostať aj pomocou metódy SVM. Knižnica poskytuje metódu, ktorá vracia distribúciu pravdepodobností.

7.5.2.2 Návrh

Riešenie bude zostavené v podobe dll súboru, ktoré bude poskytovať metódu, ktorej vstupom bude vektor súradníc a výstupom distribúcia pravdepodobností príslušnosti k jednotlivým emóciám pre daný vektor.

7.5.2.3 Implementácia

Dll súbor obsahuje triedu svm, ktorá poskytuje metódu getProbability(*string data_val*). Vstupom je string, ktorý obsahuje vektor súradníc vo formáte: číslo 1:číslo 2:číslo 3:číslo 4:číslo až po 132:číslo. Pričom prvé číslo predstavuje číslo emócie, potom za 1:,2: atd. sa nachádzajú x a y súradnice jednotlivých bodov vektora. Číslo emócie, je vyžadované formátom svm, kvôli tomu, že v niektorých funkciách sa dá po určení pravdepodobností vypočítať aj celková presnosť odhadu ak bola emócia známa. V našom prípade však nepoznáme o akú emóciu sa jedná, pretože to chceme práve zistiť. Navyše v metóde, ktorá sa používa v našom odhade pravdepodobností tento údaj nehrá žiadnu rolu a preto stačí keď tam bude pridaná len náhodná hodnota emócie. Jedná sa o uniformný formát pre všetky funkcie svm, ktorý je nemenný. Výstupom implementovanej metódy je pole typu *double*, ktorého indexy predstavujú čísla emócií a hodnoty na týchto indexoch vyjadrujú pravdepodobnosť príslušnosti vektora k danej emócií.

7.5.2.4 Testovanie

Pri testovaní riešenia v projekte sa prišlo na problém, ktorým bola nevyhovujúca verzia .NET frameworku, ktorý dané riešenie používalo. Náš projekt používa verziu 4.0, pričom riešenia bolo skompilované s verziou 4.5. Po zistení tohto nedostatku, bolo riešenie opätovne skompilované s verziou frameworku 4.0.

7.6 Upraviť architektúru spúšťača

Spúšťač je program napísaný v .NET, ktorý je oddelený od PerConIKa, ale pracuje nad rovnakou databázou. Spúšťač beží na serveri a jeho úlohou je počítať emócie pre jednotlivých používateľov.
Jeho hlavnou úlohou je:

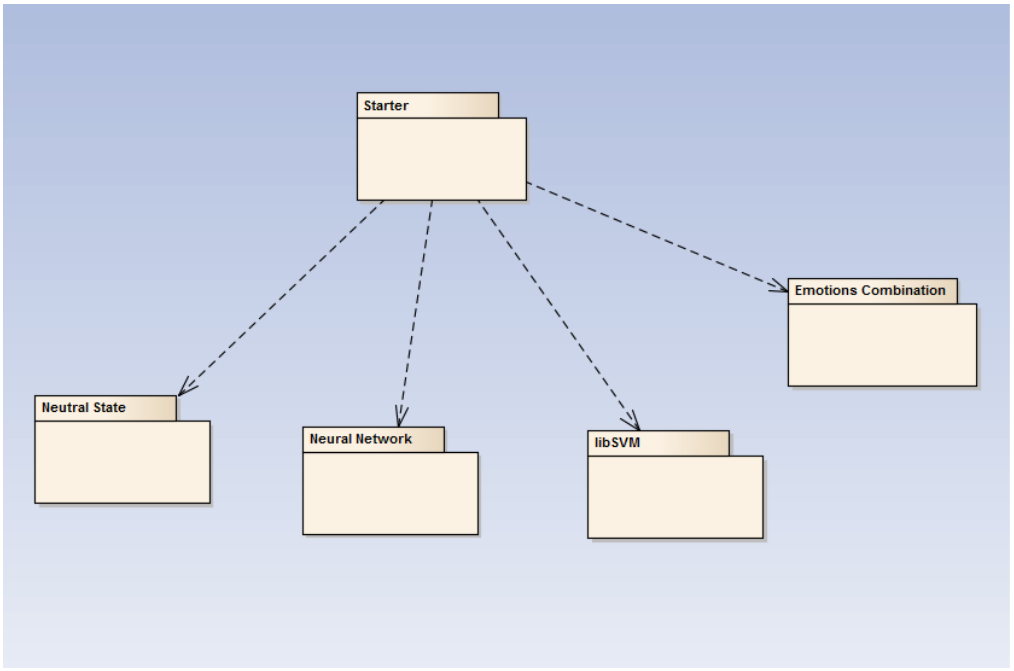
- získavanie dát z databázy
- transformácia dát pre každú z troch metód výpočtu emócie (neurónové siete, libSVM, neutrálny stav)
- výpočet neutrálneho stavu nového používateľa
- využitie všetkých 3 metód na výpočet emócií
- aplikácia kombinácie emócií na výsledky troch metód
- ukladanie výsledkov do databázy

7.6.1 Architektúra

Spúšťač sa skladá z niekoľkých komponentov. Kostra spúšťača má za úlohu v pravidelných intervaloch kontrolovať nové dáta v databáze a v prípade, že pribudli dáta, spustiť výpočet emócie. Jednotlivé metódy výpočtu emócie sú komponenty - externé dll knižnice s definovanými rozhraniami.

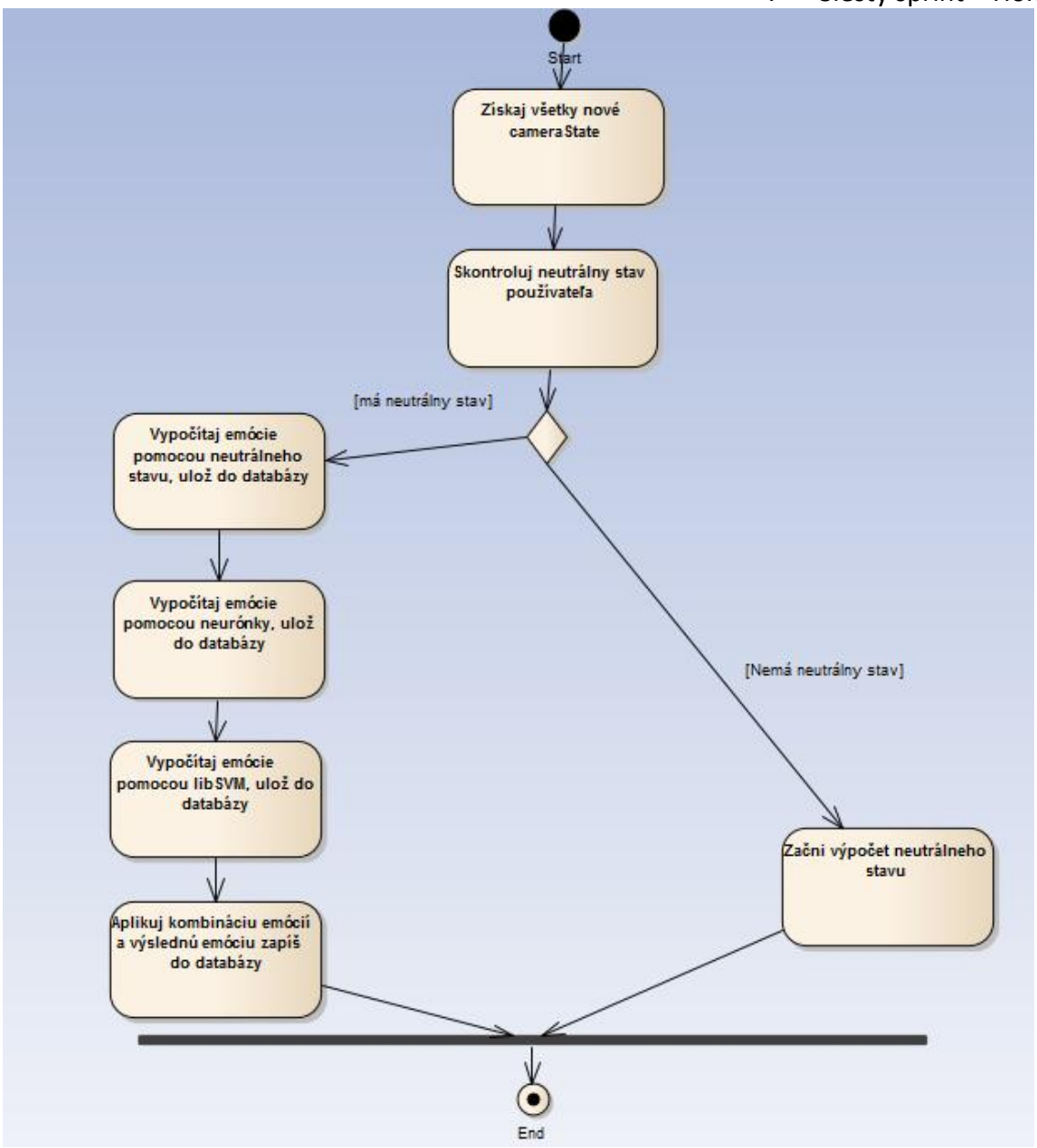
V súčasnosti sa neurónové siete a libSVM používajú naučené zo získaného datasetu fotografií, do budúcnosti sa však uvažuje aj možnosť, že by sa neustále učili na nových dátach.

Na Obrázok 33 je komponentový model spúšťača, kde je názorne ukázané ako kostra spúšťača využíva komponenty pri výpočtoch emócií.



Obrázok 33 Komponentový model spúšťača

Na Obrázok 34 je zobrazený diagram aktivít pre spúšťač, kde sú znázornené jednotlivé akcie spúšťača. Tieto akcie sa dejú periodicky v určitých časových intervaloch.



Obrázok 34 Diagram aktivít hlavného procesu spúšťača

7.7 Zhrnutie šprintu

V siedmom šprinte sme upravili jednotlivé riešenia rozpoznávania emócií do podoby dll knižníc a taktiež sme upravili spúšťač tak, aby s týmito knižnicami dokázal pracovať. Venovali sme sa tiež dopytovaniu na stav používateľa, pričom v tomto prípade sme kostru riešenia prevzali od tímu číslo 10. Taktiež sme sa venovali odporúčaniu pomocou pop-up okna a zmene farebnej schémy systému používateľa. Pri práci na tomto šprinte sa negatívne prejavila skutočnosť, že sme menili termíny stretnutí a dopracovávali niektoré úlohy z predchádzajúceho šprintu, v dôsledku čoho sme mali na úlohy z toto šprintu menej času a začali sme na nich pracovať neskôr.

8 Siedmy šprint – Infinity

V siedmom šprinte sme pokračovali v práci na dynamike emócií a venovali sme sa testovaniu a ladeniu jednotlivých metód zisťovania emócií – neutrálny stav, SVM, neurónové siete. Taktiež sme testovali súšťač a venovali sa rôznym vedľajším úlohám, ako sú úprava inštalátora, vytvorenie REST API a tvorba dokumentácie. Nasleduje zoznam úloh riešených v tomto šprinte:

- Dynamika emócií
- Výpočet akcie na serveri na základe dynamiky emócií
- Demo
- Dokumentácia
- Úprava inštalátora, odstránenie koloniek pre prihlásenie do služby, pridať možnosť zvoliť si unikátne meno
- Odstránenie vyskakovacieho okna pri zmene farebnej schémy
- Testovanie rozpoznávania neutrálneho stavu
- Testovanie spúšťača (vrátane paralelného pripojenia viacerých používateľov)
- Testovanie metódy SVM
- Testovanie neurónie
- Testovanie neutrálneho stavu
- Ošetrenie chybových stavov pri spúšťači
- Zabezpečiť spúšťanie spúšťača
- Vytvorenie REST API pre získavanie údajov z databázy
- Vyladenie parametrov neutrálneho stavu
- Vyladenie neurónie
- Vyladenie SVM
- Tvorba dokumentácie č.4
- Zníženie objemu dát odosielaných na server - implementácia časového míľnika pre CameraWatcher

8.1 Demo

Projekt je napísaný v jazyku C# s využitím frameworku .Net 4. V tomto projekte je zahrnutý aj inštalátor, ktorý inštaluje klientskú aplikáciu (logovač) na používateľov počítač.

Pre vytvorenie inštalačného programu z existujúcich zdrojových kódov je najlepšie použiť Visual Studio 2010. Podmienkou je Visual Studio 2010 SP1 s nainštalovaným rozšírením Visual Studio 2010 SP1 SDK. Používateľský účet, pod ktorým sa bude vykonávať vytváranie inštalátora, musí mať oprávnenia pre spúšťanie skriptov v PowerShell-i nastavené na *RemoteSigned*. Po zmene týchto oprávnení je potrebné reštartovať Visual Studio.

Na vytvorenie inštalátora slúži projekt *UserActivity.AppSetup*, ktorý sa nachádza v zozname projektov (Solution Explorer) po otvorení nášho riešenia vo Visual Studiu 2010. Po kliknutí pravým tlačidlom na tento projekt a vybratí možnosti *Build* alebo *Rebuild* sa spustí proces vytvorenia inštalačného balíčka.

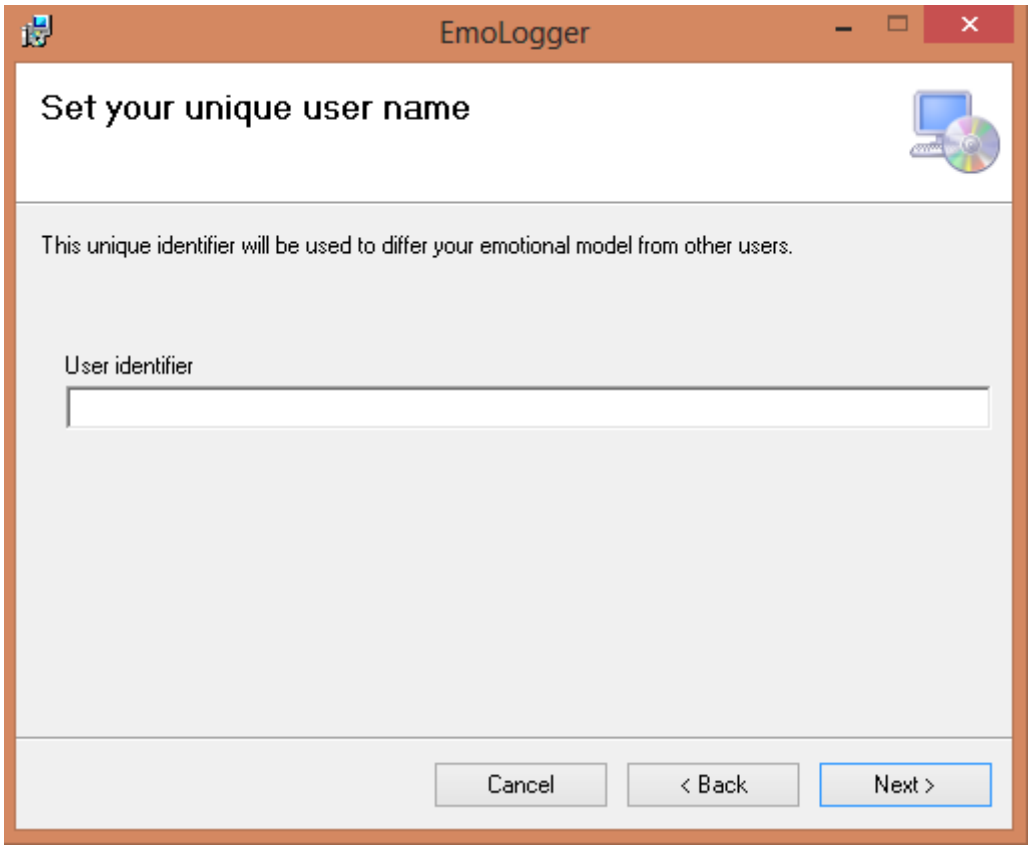
8.1.1 Problémy

Ak vytváranie inštalačného programu skončí s chybou oznamujúcou nenájdenie súboru *extension.vsixmanifest*, je potrebné vymazať adresár C:\Program Files\Microsoft Visual Studio 10.0\Common7\IDE\Extensions\Gratex a proces vytvárania inštalátora spustiť znova.

8.2 Úprava inštalátora, odstránenie koloniek pre prihlásenie do služby, pridať možnosť zvoliť si unikátne meno

Pôvodný inštalátor (z pôvodného logovača) obsahoval políčka pre zadanie údajov potrebných pre pripojenie k serveru. Naša aplikácia využíva iba jeden spôsob pripojenia k serveru, preto nie je potrebné, aby boli tieto údaje zadávané používateľom pri inštalácii programu. Z toho dôvodu sme tieto políčka odstránili a všetky potrebné údaje nastavuje inštalátor automaticky.

Namiesto toho sme pridali políčko pre zadanie unikátneho prihlasovacieho mena, ktoré slúži na odlíšenie emocionálneho modelu jednotlivých používateľov. Ak používateľ toto políčko nevyplní, používateľské meno sa automaticky nastaví na náhodný reťazec znakov. Toto používateľské meno taktiež slúži pri filtrovaní údajov z databázy, ktoré sú získavané pomocou REST API.



Obrázok 35 Obrazovka inštalačného program, kde používatelia zadávajú svoje prihlasovacie meno

8.3 Odstránenie vyskakovacieho okna pri zmene farebnej schémy

8.3.1 Analýza

Pri zmene témy, implementovanej v predchádzajúcom šprinte, vyskakuje nežiaduce oknom, ktoré môže rušiť používateľa. Preto je potrebné vyskúšať rôzne parametre pri vytváraní procesu, ktoré by okno minimalizovali, alebo zabránili jeho vyskakovaniu.

8.3.2 Návrh

Pri implementácii sme vyskúšali aj nový príkaz:

```
shell32.dll,Control_RunDLL desk.cpl desk,@Themes /Action:OpenTheme
/File:\"C:\\Windows\\Resources\\Themes\\nature.theme\
```

8.3.3 Implementácia

Keďže chceme vytvoriť triedu a metódu v C#, ktorou bude možné kdekoľvek meniť farebnú tému, rozhodli sme sa použiť C# triedu Process, ktorá umožňuje spúšťať a zastavovať lokálne systémové procesy. Pri spúšťaní procesu, sa ako argument dá funkcii nastaviť súbor nastavení, ktoré spustenie ovplyvnia. A zrušenie vyskakovacieho okna sme vyskúšali:

```
startInfo.CreateNoWindow = true;
startInfo.WindowStyle = ProcessWindowStyle.Hidden;
startInfo.WindowStyle = ProcessWindowStyle.Minimized;
```

Bohužiaľ, vyskakovacie okno sa odstrániť nepodarilo (Zaujímavé je, že nastavenie týchto parametrov pri spúšťaní iných procesov, napr. Notepad.exe fungovalo bez problémov).

8.4 Testovanie rozpoznávania neutrálneho stavu

Cieľom úlohy bolo vypátrať nedostatky v spúšťači pri získavaní neutrálneho stavu.
Objavené nedostatky:

1. Záporné čísla pri hodnotách v DB.
2. Chyby v konverzii bodov z Luxand poradia do DB poradia.
3. Zistenie rozdielov pri viacerých testoch, z dôvodu nefiltrovania extrémnych stavov.
4. Drobnejšie syntaxové chyby.

Podniknuté kroky:

1. Refaktor dátovej triedy obsahujúcej body Point_(cislo_Luxand), vzhľadom k tomu, že nekorešpondovala s reálne potrebnými bodmi. Z tohto dôvodu ani následné výpočty nemohli byť správne.

2. Odstránenie konverzie medzi DB poradím a Luxand poradím – obe teraz používajú rovnaké usporiadanie bodov.
3. Zavedenie filtrovania extrémnych stavov – odstráni sa 10% extrémov na oboch stranách usporiadaného poradia rozmerov a z týchto sa následne aritmetickým priemerom vypočíta neutrálny stav.

```
// Processing only 10-80 of the field, extreme values are tossed away
for (int i = (filterCount / 2); i < points.Count - (filterCount / 2); i++)
{
    avg_BrowsLeftToRightX_act += BrowsLeftToRightX_act[i];
    avg_EyeOutX_act += EyeOutX_act[i];
    avg_LipsOutX_act += LipsOutX_act[i];
    avg_BrowToEyeY_act += BrowToEyeY_act[i];
    avg_LipsTopBotY_act += LipsTopBotY_act[i];
    avg_LipsCornerBotY_act += LipsCornerBotY_act[i];
    avg_LipsCornerTopY_act += LipsCornerTopY_act[i];
}

// Average out of 10-80 values
avg_BrowsLeftToRightX_act /= Convert.ToDouble(points.Count - filterCount);
avg_EyeOutX_act /= Convert.ToDouble(points.Count - filterCount);
avg_LipsOutX_act /= Convert.ToDouble(points.Count - filterCount);
avg_BrowToEyeY_act /= Convert.ToDouble(points.Count - filterCount);
avg_LipsTopBotY_act /= Convert.ToDouble(points.Count - filterCount);
avg_LipsCornerBotY_act /= Convert.ToDouble(points.Count - filterCount);
avg_LipsCornerTopY_act /= Convert.ToDouble(points.Count - filterCount);
```

Obrázok 36 Ukážka modifikácie kódu

8.5 *Testovanie neutrálneho stavu*

Cieľom bolo objavenie nedostatkov metódy rozoznávania emócií z neutrálneho stavu. Funkciu testovalo viacero kolegov.
Funkcia rozoznáva nasledovné emócie:

- Šťastie
- Smútok
- Hnev
- Prekvapenie
- Únavu (zívanie)

Výsledky testovania:

1. Výrazná nepresnosť pri smútku
2. Prekryvy pri šťastí a zívaní
3. Prekryvy pri šťastí a prekvapení
4. Neúmerné intervaly pre jednotlivé emócie

Podniknuté kroky:

1. Doplnenie hodnôt, ktoré automaticky znížia hodnoty zívania a prekvapenia pri šťastí – rozšírení pier.
2. Úprava hodnôt za účelom dosiahnutia 0 – 100% intervalu pre jednotlivé emócie.

```
private static double getHappiness(double LipsOutCornersX_neutral,
                                   double LipsTopBotY_neutral,
                                   double LipsCornerTopY_neutral,
                                   double LipsCornerBotY_neutral,
                                   double LipsOutCornersX_actual,
                                   double LipsTopBotY_actual,
                                   double LipsCornerTopY_actual,
                                   double LipsCornerBotY_actual,
                                   double toleranceLipsX)
{
    double value;

    value = ((LipsCornerBotY_actual + LipsOutCornersX_actual + LipsTopBotY_actual - LipsCornerTopY_actual)
                      - (LipsCornerBotY_neutral + LipsOutCornersX_neutral + LipsTopBotY_neutral - LipsCornerTopY_neutral));

    value = value * 100 / 60;   // Adjusting to 0 - 100 interval

    if (value < 0) return 0;
    else if (value > 100) return 100;
    else return value;
}
```

Obrázok 37 Ukážka kódu pre výpočet šťastia

8.6 Ošetrenie chybových stavov pri spúšťači

Cieľom tejto úlohy bolo ošetriť možné chybové stavy pri spúšťači, ktoré by mohli spôsobiť chyby programu, prípadne jeho úplnú nefunkčnosť.
Dôležité bolo ošetriť chybové stavy v programe hlavne z dvoch pohľadov:

- prístup do databázy
- viacvláknovosť (multithreading) programu, synchronizácia

Ďalej bolo dôležité, aby bol program dostatočne robustný, keďže je navrhnutý tak, aby bežal nepretržite počas veľmi dlhej doby. Preto bolo potrebné, aby sa v prípade výskytu výnimiek pri behu programu nezrútil celý program a pokračoval ďalej aj v prípade, že určitý čas nie je schopný vykonávať svoju činnosť (napr. v prípade, že prestane pracovať databázový server). Rozhodli sme sa preto na najvyššej úrovni odchytávať výnimky a logovať ich pomocou logovacieho framework-u Apache log4net. Zdrojový kód 18 zobrazuje odchytávania výnimky na najvyššej úrovni.

```
try
{
    emotions = neuralNetwork.classifyInputData(points);
    logger.Debug("Neural network method emotions: " +
convertDoubleArrayToString(emotions));

    logger.Debug("Going to insert emotions from Neural network");
    startDbHandler.saveEmotionsForCameraState(camState, emotions,
StarterDatabaseHandler.NEURAL_NETWORK);
    logger.Debug("Insert of emotions from neural network successful");
}
catch (Exception e)
{
    logger.Error("Calculation of emotions using neural network failed!", e);
}
```

Zdrojový kód 18 Odchytávanie výnimky

Ošetrenie výnimiek je navrhnuté tak, aby v prípade výnimky bol schopný program pokračovať tam, kde skončil

8.6.1 Prístup do databázy

Z hľadiska funkcionality programu spúšťača je prístup do databázy kritický. Preto bolo nutné ošetriť možné chybové stavy pri prístupe do databázy:

- samozrejmosťou je zachytávanie výnimiek pri každej databázovej operácii (SELECT, UPDATE, INSERT)
- pred každou databázovou operáciou sa tiež kontroluje spojenie s databázou a ak nie je v poriadku, tak sa vytvorí nové spojenie

```
try
{
    ensureDbConnectionIsOk(conn);

    userCommand = new SqlCommand("SELECT (User_id), stateTimestamp FROM CameraStates c
WHERE c.Id >" + lastProcessedCameraState + " AND c.User_Id=" + userId, conn);

    userReader = userCommand.ExecuteReader();
    userReader.Read();
    return Convert.ToDateTime(userReader["stateTimestamp"]);
}
catch (Exception e)
{
    throw e;
}
finally
{
    if (userReader != null)
    {
        userReader.Close();
    }
}
```

Zdrojový kód 19 Zachytávanie výnimky pri databázových operáciách

```
private void ensureDbConnectionIsOk(SqlConnection conn)
{

    try
    {
        if (conn == null)
        {
            conn = new SqlConnection(DatabaseHandler.CONNECTION_STRING);
        }


        if (conn != null && conn.State != ConnectionState.Open)
        {
            conn.Open();
        }
    }
```

```
    catch (Exception e)
    {
        throw e;
    }
}
```

Zdrojový kód 20 Kontrola spojenia s databázou

8.6.2 Multithreading

Z pohľadu multithreadingu bolo potrebné synchronizovať prístup ku kolekcii používateľov, ktorí čakajú na výpočet neutrálneho stavu. K tejto kolekcii pristupujú dve vlákna programu a bolo potrebné, aby dáta boli konzistentné. Na tento účel bol použitý mechanizmus zámku zo systémového balíku .NET *System.Threading.*

```
private readonly object lockObj = new Object();
lock (lockObj)
{
    usersToCalculate.Add(user);
    usersToCalculateIds.Add(user.userId);
}
```

Zdrojový kód 21 Synchronizácia prístupu k používateľom

8.7 Zabezpečiť spúšťanie spúšťača

Cieľom tejto úlohy bolo zabezpečiť automatické spúšťanie procesu spúšťača v prípade reštartu systému, náhodného ukončenia spúšťača používateľom alebo ukončenia spúšťača po chybe v programe.

V prvej fáze úlohy bolo potrebné nájsť a analyzovať možnosti pre automatické spúšťanie programov. Keďže pracujeme na platforme Windows a naše riešenie je nasadené na Windows serveri, rozhodli sme sa pre Windows Task Scheduler. Táto súčasť operačného systému Windows Server poskytuje možnosti, ktoré plne vyhovujú naším potrebám:

- Windows Task Scheduler zabezpečuje spustenie programu spúšťača po reštarte systému
- každú minútu Task Scheduler kontroluje, či proces spúšťača beží a ak nie, spustí ho
- úloha, ktorá spúšťa program spúšťača sa dá jednoducho importovať cez grafické rozhranie Windows Task Scheduler z XML súboru, ktorý je priložený k inštalácii nášho riešenia spolu s jednoduchým návodom na nastavenie parametrov
- pre rôzne inštalácie sa úloha môže líšiť v 2 parametroch, ktoré je nutné nastaviť:
    - meno Windows používateľa, pod ktorým program beží
    - cesta k programu

8.8 Vyladenie neurónoiek

Pri práci s neurónovými sieťami sme na trénovanie a klasifikáciu používali výlučne dáta zo získaného datasetu. Pri klasifikácii dát z datasetu sme pomocu natrénovanej neurónovej siete dosahovali dostatočne dobré výsledky, no pri použití v spúšťači na reálnych dátach klasifikácia nefungovala. Pri testovaní sme zistili, že v normalizácii používanej pri trénovaní neurónových sietí bola chyba, ktorá sa prejavovala iba pri použití v spúšťači, kde bola použitá normalizácia bez chyby. Po opravení chyby a novom natrénovaní neurónovej siete sme získavali pomocou testovacieho programu reálnejšie výsledky, ktoré u zodpovedali rôznym emóciám.

8.8.1 Štruktúra siete

Neurónová sieť, natrénovaná na datasete, bola natrénovaná na výslednú chybovosť 6%, pretože trénovanie trvalo veľmi dlho(rádovo desiatky minút a stále sa spomaľovalo). Preto sme sa najprv venovali experimentovaniu so štruktúrou siete, aby sme dosiahli štruktúru, ktorá nám umožní trnovať sieť v kratšom čase, respektíve s lepším výsledkom v porovnateľnom čase.

Experimentovaním sme získali novú štruktúru siete, ktorá pozostáva z dvoch skytých vrstiev, pričom prvá vrstva obsahuje 60 neurónov a druhá vrstva 30 neurónov. S touto štruktúrou sme boli schopný natrénovať sieť na chybovosť 2%, čo predstavuje značné zlepšenie oproti predchádzajúcemu stavu. Ďalším pridávaním skrytých vrstviev sme už nedokázali merateľne zrýchliť trénovanie siete.

8.8.2 Záver

Natrénovanú neurónovú sieť je potrebné ďalej otestovať v spojení so spúšťačom, pre potvrdenie jej schopnosti rozpoznávať emocionálny stav aj na reálnych dátach získaných z kamery.

8.9 Vyladenie SVM

Táto úloha zahŕňa úpravu parametrov strojového učenia pomocou metódy SVM za účelom čo najlepšieho určovania emócie používateľov.

8.9.1 Predošlá práca

Doterajšia práca s SVM bola realizovaná len pomocou dát získaných z datasetu. Naším cieľom je však použitie strojového učenia založeného na metóde SVM na dátach získaných v rámci reálnej interakcie používateľov s našim systémom.

Celé riešenie určovania emócie pomocou metódy SVM, bolo vytvárané a následne validované na údajoch z datasetu. Finálna podoba riešenia bola vyladená tak, že určovala emócie s presnosťou 100%. Bol vytvorený model, pomocou ktorého sa dosahovala táto presnosť za účelom používania na reálnych dátach, ktoré sa budú získavať pri interakcii s našim systémom.

8.9.2 Výsledky s reálnymi dátami

Pri použití modelu, s ktorým sa dosahovala 100% úspešnosť určovania emócií v rámci „laboratórnych“ dát bolo aplikovanie na reálne dáta neuspokojivé. Použitý bol polynomiálny model. Žiadna emócia nebola určená správne, pričom pri väčšine prípadov bol hnev určovaný ako najpravdepodobnejšia emócia a to aj v prípade, že sme sa do kamery usmievali, respektíve simulovali emóciu šťastia. Na trénovanie modelu používame stále údaje z datasetu. Do úvahy ako zvýšiť presnosť určovania na reálnych dátach spadali 2 hlavné obmeny:

- Zmena KERNEL-u svm, pomocou ktorého sa generuje model
- Aplikácia normalizácie dát, ktorá spočíva v zostavení uniformných súradníc, na ktoré nebude vplývať zmena polohy tváre používateľa(vzdialenosť, natočenie...)

8.9.2.1 Zmena KERNEL-u

Knižnica, ktorá sa používa na aplikáciu strojového učenia metódou SVM, ponúka viacero variácií výpočtových KERNEL-ov, pomocou ktorých sa generuje model slúžiaci na určovanie emócií. Testovanie jednotlivých obmien bolo vykonané na viacerých ľuďoch, pričom výsledky medzi jednotlivými používateľmi sa menili len malým podielom, pričom hlavnou príčinou rozdielov spočíva najmä v schopnosti zahrania danej emócie.

Šťastie	OK
Hnev	OK
Smútok	Mierna zámena so znechutením. Pre niektorých používateľov bolo OK.
Znechutenie	OK
Strach	Pomerne vysoká pravdepodobnosť ale podobné čísla malo aj znechutenie.
Prekvapenie	Zle, ale ťažkosti so simuláciou

Tabuľka 5 Lineárny kernel

Šťastie	OK, niekedy obmena so znechutením
Hnev	OK
Smútok	Zámena so znechutením a hnevom
Znechutenie	OK
Strach	Zámena so znechutením
Prekvapenie	Všetky emócie približne na rovnako

Tabuľka 6 SIGMOID kernel

Šťastie	Zle, zámena so znechutením
Hnev	Zle
Smútok	Zle
Znechutenie	OK
Strach	OK
Prekvapenie	Zle

Tabuľka 7 RBF kernel

Ako vidno, najlepšou voľbou sa ukázalo byť použitie lineárneho KERNEL-a. Pri jeho aplikácia bola dosiahnutá presnosť určovania 4 zo 6 emócií. Pričom treba brať do úvahy aj fakt problematickej simulácie emócií, ako napríklad strach či prekvapenie. Zaujímavosťou je, že na výsledky nemala zmena pozície tváre používateľa takmer žiaden vplyv.

8.9.2.2 Aplikácia normalizácie dát

Normalizácia dát spočíva v prepočte súradníc takým spôsobom aby sa eliminoval vplyv zmeny polohy tváre používateľa. Takouto zmenou môže pritom byť zmena vzdialenosti tváre od kamery ale posunutie tváre do strán. Celkovo boli použité 2 normalizačné prístupy avšak ani pri jednom z nich nebola dosahovaná ani len približná miera presnosti určovania emócie. Prvotným indikátorom nevyhovujúceho formátu dát bola nízka presnosť určovania emócií na údajoch z datasetu(okolo 35%), pričom pri testovaniach v predošlej sekcii boli presnosti dosahované na údajoch z datasetu od 75% až po 95%. Vzhľadom na túto skutočnosť bola mnohými spôsobmi vykonaná kontrola procesu normalizácie avšak bez nájdenia závažnej chyby.

8.9.3 Záver

Z vyššie spomenutých dôvodov sa zatiaľ zvažuje použitie nenormalizovaného lineárneho modelu, s ktorým boli dosiahnuté najlepšie výsledky.

8.10 Zníženie objemu dát odosielaných na server - implementácia časového míľnika pre CameraWatcher

8.10.1 Motivácia

Pri odosielaní dát na server sme si všimli, že niektoré aktivity nebolo možné odoslať na server, a preto sa hromadili v lokálnej databáze. Program sa periodicky snažil odoslať tieto aktivity, ale vždy neúspešne. Pri analýze jednotlivých požiadaviek sme si roztriedili úspešné a neúspešné požiadavky. Zistili sme, že neúspešné požiadavky mali väčší objem ako úspešné. Najväčší priestor v požiadavke zaberali údaje pre *CameraState*. Preto bolo potrebné obmedziť počet stavov zachytených z kamery tak, aby sa v jednej požiadavke neodosielalo naraz príliš veľa dát na server.

Pôvodne bola jedna aktivita vytváraná až keď používateľ prepol aktívne okno. Ak to dlhšiu dobu neurobil, za ten čas sa do aktivity pridalo veľa stavov z kamery. Našim cieľom bolo preto vytvorenie aktivity nie len po prepnutí medzi oknami, ale aj po dosiahnutí určitého počtu stavov z kamery.

8.10.2 Analýza

Komponentom, ktorý je zodpovedný pre vytváranie zoznamu stavov pre aktivitu je *CameraWatcher*. Tento komponent vytvára zoznam aktivít, až kým *AktivityComposer* nezavolá metódu *OnActivityFinishing()* z *CameraWatcher*. *ActivityComposer* zavolá *OnActivityFinishing()* vo všetkých *Watchers* vtedy, ak príjme *EventDto* s nastavenou hodnotou *IsMilestone = true.*

8.10.3 Implementácia

Pre spustenie vytvárania aktivity sa používa objekt *ActivityEventDto*, ktorý však obsahuje viacero povinných atribútov a taktiež sa odosiela na server. Preto sme vytvorili nový objekt *TimeHasComeDto* (rozširuje objekt *EventDto*), ktorým bude *CameraWatcher* spúšťať vytvorenie aktivity.

V metóde *createAllStates()* triedy *CameraWatcher* sa do zoznamu aktuálnych stavov kamery pridávajú nové stavy. Ak počet stavov dosiahne určité číslo, je zavolané vytvorenie novej aktivity.

```
// ak počet stavov prekročí určitú hranicu, zavolaj ukončenie aktivity
if (mStates.Count > PHOTOS_PER_ACTIVITY)
{
	var timeHasComeDto = new TimeHasComeDto()
	{
		IsMilestone = true,
		Time = DateTime.UtcNow
	};
	mActivityComposer.BeginAddEvent(timeHasComeDto);
}
```

Zdrojový kód 22 Tento kód sa nachádza v CameraWatcher.createAllStates() a zabezpečuje vytvorenie activity po určitom počte stavov z kamery

8.11 Zhrnutie šprintu

V siedmom šprinte sme sa museli vysporiadať so skutočnosťou, že sme veľkú časť šprintu nemali k dispozícii server, čo nás prinútilo vymyslieť iný spôsob testovania rozpoznávania emócií. Z tohto dôvodu sme vytvorili samostatnú aplikáciu, ktorá zachytávala obraz z kamery, získavala z neho body tváre, urobila potrebné predspracovanie a spustala jednotlivé metódy rozpoznávania emócií. Pomoocu tohto programu sme testovali a ladili jednotlivé metódy. Počas šprintu sme sa tiež rozhodli urobiť zmenu v normalizácii dát, čomu sa musela prispôsobiť aj normalizácia použitá pri trénovaní neurónových sietí.

9 Ôsmy šprint – Jaguar

V ôsmom šprinte sme pokračovali v práci na dynamike emócií a venovali sme sa ladeniu jednotlivých častí riešenia, ako je rozoznávanie smútku pomcou metódy neutrálneho stavu, neurónové siete a chybové stavy v spúšťači. Taktiež sme sa venovali kombinácii emócií z troch metód určovania emocionálneho stavu. Vytvorili sme tiež REST API pre získanie zaznamenaných dát z databázy, čo umožní využívať zaznamenávané dáta aj inými vývojármi, pracujúcimi na iných projektoch. Nasleduje zoznam úloh riešených v tomto šprinte:

- Dynamika emócií
- Výpočet akcie na serveri na základe dynamiky emócií
- Neutrálny stav vyladenie sadness
- Neurónky - trénovanie
- Vyladenie exception v spúšťači
- Vyladenie posielania (počet stavov)
- Ukladanie CameraState v poradí Luxandu
- Kombinácia emócií
- Vytvorenie REST API pre získavanie údajov z databázy
- Testovanie spúšťača (vrátane paralelného pripojenia viacerých používateľov)
- Tvorba dokumentácie č.4
- Tvorba dokumentácie č.5

9.1 Neutrálny stav vyladenie sadness

Došlo k miernemu vyladeniu pre smútok, metóda už nebere do úvahy X-vzdialenosť kútikov pier.

Výpočet smútku z neutrálneho stavu vykazuje nedostatočnú presnosť, čo bude zohľadnené pri Kombinácii emócií. Z tohoto dôvodu nie je potrebné snažiť sa o vyladenie metódy.

9.2 Neurónky – trénovanie

Cieľom tejto úlohy bolo natrénovať neurónky na rôzne hodnoty chybovosti a otestovať ich presnosť určovania emócií. Motiváciou k tomuto bolo podozrenie, že pre veľmi nízke hodnoty chybovosti už dochádza k preučeniu siete a táto je následne menej presná pri určovaní emócií z bodov získaných z kamery.

Pre účely testovania sme preto natrénovali neurónové siete na hodnoty chybovosti od 2 do 10%. Následne sme tieto siete používali v testovacom programe, pričom sme zaznamenávali získané hodnoty pre jednotlivé testované emócie. Počas testovania sme najpresnejšie výsledky získavali s neurónovou sieťou, natrénovanou na 2% chybovosť, čo vyvrátilo našu domnienku, že k nesprávnemu určovaniu emócií dochádzalo v dôsledku preučenia siete. Ako sa ukázalo, tento problém spôsobovala nekonzistencia v normalizácii dát medzi trénovaním neurónových sietí a spúšťaním v spúšťači.

9.3 Vyladenie exception v spúšťači

Cieľom tejto úlohy bolo odstrániť príčinu výnimky, ktorá nastávala pri počítaní neutrálneho stavu v spúšťači, konkrétne pri odstraňovaní používateľa z kolekcií po úspešnom výpočte neutrálneho stavu.

```
// User is removed from list as he
logger.Debug("Removing user with userId " + user.userId + " from list");

lock (lockObj)
{
    usersToCalculate.Remove(user);
    usersToCalculateIds.Remove(user.userId);
}
```

Zdrojový kód 23 Odstraňovanie používateľa z kolekcií

Pri tejto operácii sa vždy objavila výnimka:

```
Collection was modified; enumeration operation may not execute.
```

Takže bolo potrebné nájsť príčinu. Prvé odhady sme zamerali na synchronizačný problém, keďže ku kolekciám pristupujú viaceré vlákna programu. Vyskúšali sme viacero typov zámkov, aj synchronizované kolekcie, ale problém pretrvával. Tak sme hľadali tento problém na diskusných fórach (stackoverflow) a nakoniec sme našli príčinu: cez jednu z kolekcií sa prechádzalo v cykle, pričom vnútri tohto cyklu sa kolekcia zmenila (odstránil sa používateľ s vypočítaným stavom). Toto bolo jednoduché odstrániť tak, že z kolekcie sa vytvoril zoznam a cez ten sa prechádzalo v cykle.

```
foreach (NewUser us in usersToCalculate)
{
    logger.Debug("Checking user with userId: " + us.userId);
    if (dbHandler.hasUserEnoughPoints(us, STATES_NEEDED))
    {
        logger.Debug("User with Id " + us.userId + " has enough points. Calculating
neutral state ...");
        calculateNeutralStateForUser(us);
    }
}
```

Zdrojový kód 24 Pôvodný kód

```
foreach (NewUser us in usersToCalculate.ToList())
{
    logger.Debug("Checking user with userId: " + us.userId);
    if (dbHandler.hasUserEnoughPoints(us, STATES_NEEDED))
    {
        logger.Debug("User with Id " + us.userId + " has enough points. Calculating
neutral state ...");
        calculateNeutralStateForUser(us);
    }
}
```

Zdrojový kód 25 Upravený kód

Po tejto zmene v kóde sa výnimka už neobjavila, výpočet neutrálneho stavu funguje bez problémov.

9.4 Vyladenie posielania (počet stavov)

Na vytvorenie spojenia medzi klientom a serverom je potrebné určité množstvo prostriedkov, čo sa pri väčšom počte požiadavok môže prejaviť spomalenou odozvou servera. Preto sme sa rozhodli, že bolo by vhodné optimalizovať veľkosť odosielanej správy tak, aby odosielanie neskončilo chybou a súčasne aby počet požiadaviek bol čo najmenší. Najjednoduchšou formou zmenšenia posielanej správy bolo obmedzenie počtu stavov z kamery v jednej aktivite. Experimentami sa nám podarilo zistiť, že pri 19 stavoch v jednej aktivite skončia všetky požiadavky úspešne. Nakoniec sme maximálny počet stavov v jednej aktivite nastavili na 17 pre prípad, že by sa jej veľkosť zväčšila kvôli iným faktorom.

9.5 Ukladanie CameraState v poradí Luxandu

Cieľom tejto úlohy bolo upraviť PerConiKa tak, aby sa ukladali body tváre v poradí ako definuje Luxand. V pôvodnom návrhu, ktorý sa robil v začiatočnej fáze projektu totiž nebolo jasné aké body a v akom tvare budeme potrebovať. Preto sme sa rozhodli body logicky rozdeliť podľa častí tváre, ktoré popisujú (ľavé obočie, ľavé oko, nos, atď.). Všetky naše metódy na počítanie emócií však pracujú s bodmi Luxandu v poradí aké definuje Luxand. V spúšťači bolo preto nutné vytvoriť mapovanie bodov medzi databázou a Luxandom, čo mohlo byť zdrojom ťažko odhaliteľných chýb. Preto sme sa rozhodli upraviť PerConIKa tak, aby sme mali v databáze body v poradí ako definuje Luxand.
Pre dosiahnutie tohto cieľa bolo potrebné spraviť viacero krokov:

- úprava XML formátu pre entitu CameraState, bolo potrebné odstrániť jednotlivé časti tváre a nahradiť ich jedným elementom
- úprava tabuľky **FacePoints**, odstránený stĺpec **facePart** a odstránená tabuľka **FaceParts**
- úprava v súbore **CameraWatcher.cs**, odstránené vytváranie jednotlivých častí tváre pre body. Teraz sa vytvára len jeden zoznam (List<FPDto>) so všetkými 66 bodmi z Luxandu v poradí Luxandu.
- úprava **CameraStatesListConvertor.cs,** bolo potrebné odstrániť napĺňanie stĺpca facePart v tabuľke FacePoints a tiež bolo potrebné upraviť konverziu databáza -> XML
- odstránenie mapovania bodov databáza <-> Luxand v spúšťači nakoľko už v databáze sú v poradí aké definuje Luxand
- nasadenie novej verzie spúšťača na server a úprava databázy na serveri, otestovanie riešenia

```
<CameraStateDto      mDateTime="2013-04-11T00:06:05.300157+02:00"      frameWidth="640"
frameHeight="480">
   <LeftEyebrow>
     <FPDto X="124" Y="285" rX="44" rY="38" />
     <FPDto X="172" Y="278" rX="140" rY="31" />
        ...
  </LeftEyebrow>
  <LeftEye>
     <FPDto X="124" Y="285" rX="44" rY="38" />
     <FPDto X="172" Y="278" rX="140" rY="31" />
        ...
  </LeftEye>
  ...
</CameraStateDto>
```

Zdrojový kód 26 Pôvodný XML formát pre CameraState

```
<CameraStateDto      mDateTime="2013-04-11T00:06:05.300157+02:00"      frameWidth="640"
frameHeight="480">
   <FacePoints>
     <FPDto X="124" Y="285" rX="44" rY="38" />
     <FPDto X="172" Y="278" rX="140" rY="31" />
        ...
  </FacePoints>
</CameraStateDto>
```

Zdrojový kód 27 Upravený XML format pre CameraState

Pri tejto úlohe sme nenarazili na žiadne závažnejšie problémy. Upravené riešenie pracovalo správne, bolo otestované. Dostávali sme aj rovnaké alebo veľmi porovnateľné výsledky ako pred úpravou programu, takže môžeme úlohu považovať za splnenú.

9.6 Kombinácia emócií

9.6.1 Vstupné údaje

Aktuálne pravdepodobnosti emočných stavov z DB tabuľky Emotions pre jednotlivé metódy dosadí priamo spúšťač. Kombinácia riešení vyžaduje 3 vstupné polia pre každú z používaných metód.
Používané metódy:

1. Neutrálny stav
2. SVM
3. Neurónové siete

Pre jednotlivé metódy vstupujú pravdepodobnosti výskytu emócií.
Emócie(metódy):

1. hnev (1,2,3)
2. šťastie (1,2,3)
3. smútok (1,2,3)

4. prekvapenie (1,2,3)
5. znechutenie (2,3)
6. strach (2,3)
7. únava (1)

Vstupné údaje sa uložia do zoznamu inštancií triedy EmotionNode. Vytvorí sa 7 inštancií pre každú emóciu.

```
class EmotionNode
{
    public string emoName = "";

    public double mNeutral = 0;
    public double mNeuron = 0;
    public double mLibSvm = 0;

    public int oNeutral = 0;
    public int oNeuron = 0;
    public int oLibSvm = 0;

    public double result = 0;
    public double oResult = 0;
```

Obrázok 38 Formát vlastností objektu EmotionNode

9.6.2 Jadro funkcie

Úlohou funkcie je skombinovať existujúce metódy na výpočet emócie a vypočítať výslednú emóciu.
Jadro funkcia sa skladá z:

1. Namapovania hodnôt SVM a Neurónových sietí na interval 0-100
2. Udelenie poradia jednotlivým emóciam pre danú metódu
3. Stanovenie pravidiel pre špecifické situácie
4. Pridanie metód do testovacieho projektu
5.

9.6.2.1 Mapovanie hodnôt SVM a Neurónových sietí na interval 0 – 100

Maximum a posun od neutrálneho stavu pre funkcie *SVM a Neurónové siete* sú kľúčové pre získanie správneho výsledku z kombinácie.
Maximum – maximálna dosiahnutá hodnota pre danú emóciu
Posun od neutrálneho stavu – hodnota emócie pri neutrálnom stave, snaží sa o vyváženie chýb pri získaní emócie.
Príklad:

1. používateľ sa tváril neutrálne a metóda vyhodnotila 29% smútok
2. používateľ sa následne tváril smutne a metóda vyhodnotila 34% smútok

Na základe vyššie spomenutého príkladu by sme mohli tvrdiť, že pri neutrálnom stave je používateľ na 29% smutný, čo by mohlo výrazne narušiť výsledky kombinácie.

Emócia	SVM – posun	SVM - max	Siete - posun	Siete - max
hnev	0.08	0.28	0.20	1.00
šťastie	0.10	0.40	0.00	1.00
smútok	0.15	0.45	0.30	0.80
prekvapenie	0.10	0.40	0.10	0.90
znechutenie	0.16	0.36	0.15	0.95
strach	0.10	0.50	0.20	0.70
únava	NR*	NR*	NR*	NR*

Tabuľka 8

NR – not recognised – metódy SVM a Siete nemajú natrénované prejavy únavy, nakoľko neboli obsiahnuté v učebnom datasete.
Na základe získaných hodnôt maxima a posunu dokážeme namapovať tieto hodnoty na intervaly 0 – 100%.

```
if (list[i].emoName == "anger")
{
    list[i].mNeuron *= 100;
    list[i].mNeuron -= 20;  // Interval 0,2 - 1
    if (list[i].mNeuron > 100) list[i].mNeuron = 100;
    else if (list[i].mNeuron < 0) list[i].mNeuron = 0;

    list[i].mLibSvm *= 100;
    list[i].mLibSvm -= 8;   // Interval 0,08 - 0,28
    list[i].mLibSvm *= 5;
    if (list[i].mLibSvm > 100) list[i].mLibSvm = 100;
    else if (list[i].mLibSvm < 0) list[i].mLibSvm = 0;
}
```

Obrázok 39 Ukážka normalizácie emócie pre dané metódy

9.6.2.2 Udelenie poradia jednotlivým emóciám v rámci metódy

Pre každú metódu na výpočet emócií sa stanoví poradie výsledných emócií. Dosiahne sa to preusporiadaním zoznamu emócií a následné inkrementálne pridelenie poradia.

9.6.2.3 Stanovenie pravidiel pre špecifické situácie

1. Zhoda všetkých troch metód na jednej emócii
    a. Automatické pridelenie

Zistené špecifikum	Riešenie
Zhoda 3 metód na jednej emócií	99% pridelenie pravdepodobnosti.
Zhoda 2 metód na jednej emócií, pričom 3. vyhodnocuje danú emóciu na 2. miesto	95% pridelenie pravdepodobnosti.
Zvýšená hodnota únavy podľa neutrálneho stavu	1. miesto pridelí únave + % 2. a 3. miesto pridelí štandardne

	podľa skóre.
Znechutenie a strach nie sú súčasťou výpočtu z neutrálneho stavu	Úprava týchto hodnôt pre tieto emócie.

Tabuľka 9

9.6.2.4 Pridanie metód do testovacieho projektu

Klientský testovací projekt zlučuje všetky metódy a umožňuje jednoduché a rýchle testovanie pri ladení jednotlivých výsledkov emócií, ich kombinácie a pod.

9.6.3 Výstup

Výstupné údaje sú vracané spúšťaču. Jedná sa o 3 najvýraznejšie emócie – názov a hodnota umiestnené do objektu CombinedEmotion.

9.7 Vytvorenie REST API pre získavanie údajov z databázy

9.7.1 Motivácia

Počas riešenia tohto projektu sme si uvedomili, že zaznamenávame unikátne údaje, ktoré by možno niektorý výskumný tím vedel využiť a dokázal by vydolovať z týchto údajov nejakú zaujímavú znalosť. Týmto by sa mohol rozšíriť dosah nášho projektu. Preto sme sa rozhodli, že tieto údaje sprístupníme verejnosti vo forme REST API.

9.7.2 Implementácia

REST API pre získavanie údajov z databázy sa nachádza na adrese:
http://team14-12.ucebne.fiit.stuba.sk/RestAPI/RestService.svc/

atribút	**typ**	**povinný?**	**popis**
type	string	Áno	Spôsob zobrazenia výsledkov. Povolené hodnoty: *json*
action	string	Áno	Funkcia pre výber údajov z databázy. Povolené hodnoty: *getEmotionsInfo*
user	string	Nie	Filter podľa používateľovho loginu
method	string	Nie	Filtrovanie emócií podľa metódy ich rozpoznania. Vstupom môže byť prirodzené číslo, alebo prirodzené čísla oddelené čiarkou
count	Integer	Nie	Obmedzenie počtu zobrazených výsledkov. Východzí stav je 20 výsledkov.
from	string	Nie	Filtrovanie výsledkov podľa dátumu. Tento atribút vyjadruje spodné ohraničenie. Formát: YYYY/DD/MM,HH:mm:ss

to	string	Nie	Filtrovanie výsledkov podľa dátumu. Tento atribút vyjadruje horné ohraničenie. Formát: YYYY/DD/MM,HH:mm:ss
descending	integer	Nie	Ak atribút je rovný 1, výsledky budú zoradení zostupne.

Tabuľka 10 Zoznam možných atribútov http požiadavky

Výstupom sú údaje z databázy v zvolenom formáte.


```
{ "GetEmotionsInfoResult" : [
       { "Emotions" : [
              { "Anger" : 0.12450593914821351,
              "Disgust" : 0.975461208089915,
              "Exhaustion" : -1,
              "Fear" : 0.054431247072473266,
              "Happiness" : -0.922367922704028,
              "MethodId" : 3,
              "Sadness" : -0.98273208968063552,
              "Surprise" : 0.79446381956579781
            },
            { "Anger" : 6.3300970873786468,
              "Disgust" : -1,
              "Exhaustion" : 0,
              "Fear" : -1,
              "Happiness" : 0,
              "MethodId" : 1,
              "Sadness" : 55.894590846047059,
              "Surprise" : 0
            }
          ],
        "FrameHeight" : 480,
        "FrameWidth" : 640,
        "StateId" : 10424,
        "StateTimestamp" : "3/16/2013 6:05:38 PM",
        "User" : "pouzivatel1"
      },
    ] }
```


Tabuľka 11 Príklad výstupu akcie getEmotionsInfo

9.8 Testovanie spúšťača (vrátane paralelného pripojenia viacerých používateľov)

Cieľom tejto úlohy bolo testovanie spúšťača na serveri. Táto úloha tiež zahŕňala odstránenie problémov odhalených počas testovania.

Dôležité pre testovanie bolo v prvom kroku nakonfigurovať logovanie v programe, aby sme mohli testovať a pozorovať správanie programu a v prípade problémov odstrániť príčinu. Na logovanie používame framework Apache log4net. Pri logovaní používame dve úrovne logovania: DEBUG a INFO. DEBUG logovanie je veľmi podrobné a loguje sa každá operácia v kóde, pred aj po volaní. INFO úroveň sa používa pri logovaní dôležitých udalostí pri behu programu, aby bolo možné kontrolovať bežiaci proces. Osobitne sa ešte logujú

všetky výnimky, ktoré môžu nastať pri behu programu a tieto nie sú závislé od úrovne logovania.
Pri testovaní bolo potrebné overiť najmä:

- funkčnosť spúšťača ako takého
    - či sa správa tak ako má, či nepadne v dôsledku chybového stavu
    - či sa pravidelne vykonávajú procesy počítania emócií a neutrálneho stavu
    - či sa správne vypočítava neutrálny stav a emócie
    - či sa výsledky zapisujú do databázy
    - či sa spúšťač po ukončení a opätovnom zapnutí správa tak ako má a obnoví svoj pôvodný stav
- správanie sa spúšťača pri viacerých používateľoch
    - či sa správa spúšťač správne aj pri viacerých používateľoch
    - či počíta emócie a neutrálne stavy správne aj pre viacerých používateľov

Pri testovaní sa objavilo niekoľko menších problémov:

- výnimka pri odstraňovaní používateľa z kolekcie pre používateľov čakajúcich na výpočet neutrálneho stavu po tom, ako bol vypočítaný neutrálny stav (odstránené v rámci ďalšej úlohy šprintu)
- emócie sa počítali aj pre fotky, z ktorých bol vytváraný neutrálny stav - vyriešené, emócie sa začnú počítať až z fotiek, ktoré neboli použité pre neutrálny stav
- pri počítaní neutrálneho stavu ak sa prihlási nový používateľ a potom sa odhlási, pričom nepošle dostatočný počet fotiek, spúšťač donekonečna čakal na ďalšie fotky pre používateľa - vyriešené, pridaný časový limit 10 minút od prvej fotky dokedy musí byť neutrálny stav vypočítaný. Potom sa používateľ odstráni z kolekcie a ak sa znova prihlási, začne sa počítanie od prvej novej fotky

V ostatných aspektoch fungoval spúšťač podľa očakávania, spoľahlivo a stabilne. Bol spustený nepretržite niekoľko dní, prichádzali dáta od rôznych používateľov. Vykonali sme aj skupinový test, keď sme sa prihlásili 6 používatelia naraz každý pod novým používateľským menom. Spúšťač sa správal podľa očakávania.

9.9 Zhrnutie šprintu

V ôsmom šprinte sme sa venovali úlohám spojeným s dynamikou emócií a kombináciou troch metód rozoznávania emócií. V súvislosti s kombináciou metód sme potrebovali otestovať spúšťač, aby sme si boli istý, že všetky tri riešenia pracujú správne a spúšťač je pripravený na nasadenie, pričom musí byť schopný zvládnuť aj viacero prihlásených používateľov. Taktiež sme sa snažili vyladiť rozoznávanie emócie sadness (smútok) pomocou metódy neutrálneho stavu, kde sme ale došli k záveru, že ďalšie zlepšenie už nie je možné, metóda nedosahuje dostatočnú presnosť a preto to bude zohľadnené v kombinácií riešení.

10 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