Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Marko Janković

Mobilna nadzorna plošča za dogodkovno vodene arhitekture

diplomsko delo

na univerzitetnem študiju računalništva in informatike, smer informatika

Mentor: izr. prof. dr. Marko Bajec

Ljubljana, 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01796/2012
Datum: 10.01.2012

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MARKO JANKOVIĆ**

Naslov: **MOBILNA NADZORNA PLOŠČA ZA DOGODKOVNO VODENE ARHITEKTURE**

MOBILE DASHBOARD FOR EVENT DRIVEN ARCHITECTURES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Dogodkovno vodene arhitekture so programske arhitekture, ki omogočajo zajem podatkov iz različnih, heterogenih podatkovnih virov, njihovo (inteligentno) obdelavo in reakcijo na pomembne dogodke v (skoraj) realnem času. Zaradi poplave različnih merilnih in drugih naprav, ki so sposobne podatke o svojem stanju ali merjenih količinah posredovati po različnih komunikacijskih protokolih, postajajo dogodkovno-vodene arhitekture čedalje bolj pomembne, saj nudijo naravno in učinkovito ogrodje za zajem, integracijo in obdelavo zajetih podatkov ter posredovanje tako obdelanih podatkov naročnikom. Ena ključnih komponent dogodkovno vodenih arhitektur so nadzorne plošče, ki nudijo vpogled v zajete in obdelane podatke. V okviru diplomske naloge opišite koncept dogodkovno vodenih arhitektur, podajte par primerov rešitev s tega področja ter razvijte mobilno nadzorno ploščo, ki bo omogočala uporabnikom mobilnih telefonov z operacijskim sistemom Android nadzor nad dogodkovno vodeno arhitekturo Occapi.

Mentor:

prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Marko Janković, z vpisno številko **63060089**, sem avtor diplomskega dela z naslovom:

Mobilna nadzorna plošča za dogodkovno vodene arhitekture

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Marka Bajca
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 11. marec 2012 Podpis avtorja:

Za podporo in številne dobre nasvete se zahvaljujem svojemu mentorju, izr. prof. dr. Marku Bajcu. Posebna zahvala gre tudi moji družini in prijateljem, ki so mi v času študija in pisanja te naloge stali ob strani in me podpirali.

Kazalo

Seznam uporabljenih kratic in simbolov

EDA – Event Driven Architecture

SOA – Service Oriented Architecture

OS – Operating System

SOAP – Simple Object Access Protocol

ACID – Atomicity, Consistency, Isolation, Durability

RFID – Radio Frequency Identification

IT – Information Technology

KPI – Key Performance Indicator

M2M – Machine-to-Machine

API – Application Programming Interface

REST – Representational State Transfer

SQL – Structured Query Language

JSON – JavaScript Object Notation

HTTP – HyperText Transfer Protocol

XML – Extensible Markup Language

YAML – Yet Another Multicolumn Layout

ADT – Android Development Tools

SDK – Software Development Kit

MD5 – Message-Digest algorithm 5

Povzetek

Uporaba EDA je danes prisotna na številnih področjih, kot so npr.: pametne hiše in mesta, zdravstvo, nadzor prometa, internet stvari (angl. "Internet of things") ipd. Glavni cilj je omogočiti spremljanje dogajanja v sistemu, v realnem času, kar omogoči hitrejše in bolj učinkovito sprejemanje potrebnih odločitev in izvajanje ustreznih ukrepov. Slednje je bila tudi glavna motivacija in razlog za razvoj mobilne nadzorne plošče za dogodkovno vodeno platformo Occapi, s katero želimo uporabnikom omogočiti, da lahko dogajanje v platformi Occapi spremljajo v vsakem trenutku in so o pomembnih dogodkih obveščeni v najkrajšem možnem času.

Razvoj nadzorne plošče za mobilno platformo Android predstavlja osrednji del diplomske naloge in je potekal v sodelovanju z Laboratorijem za podatkovne tehnologije, ki deluje v okviru Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za boljše razumevanje razvite aplikacije in teoretičnega ozadja je v prvem delu diplomske naloge na kratko predstavljena EDA, kasneje pa je opisana še visokonivojska arhitektura dogodkovno vodene platforme Occapi.

Ključne besede: dogodkovno vodena arhitektura, dogodkovno usmerjena arhitektura, Occapi, Android, dogodek, dogodkovno vodena platforma, pametni telefon, razvoj mobilne nadzorne plošče

Abstract

The use of EDA is nowadays present in various fields like smart houses and cities, health care, traffic control, Internet of things etc. The main objective is to enable the monitoring of system activity in real time, which enables faster and more efficient decision-making and helps taking appropriate measures. The latter was also the main motivation and reason for the development of the mobile control panel for the event-driven Occapi platform, the purpose of which is to enable users to monitor events in the Occapi platform at any moment, while also notifying them about important events in the shortest time possible.

The development of the control panel for the mobile platform Android represents the main part of the diploma thesis and was carried out in cooperation with the Laboratory for data technologies at the Faculty of computer and information science, University of Ljubljana. For a better understanding of the developed application and the theoretical background, the first part of the diploma thesis briefly presents EDA, while a description of the high-level architecture of the event-driven Occapi platform is provided later on.

Keywords: event-driven architecture, event-based architecture, Occapi, Android, event, event-driven platform, smartphone, development of the mobile control panel

Poglavje 1

Uvod

Celotno dogajanje okoli nas in vse nastale spremembe je možno opisati z množico dogodkov. Dejstvo, da je svet okoli nas dogodkovno voden, predstavlja tudi enega izmed glavnih razlogov za izoblikovanje ideje o dogodkovno vodeni arhitekturi, ki je v svetu računalništva prisotna že kar nekaj časa. Povečanje zanimanja za njeno implementacijo in dejansko uporabo v organizacijah in sistemih se je pojavilo z razvojem kompleksnega procesiranja dogodkov, kar je EDA dvignilo na povsem nov nivo.

Glavni namen dogodkovno vodene arhitekture je izkoristiti prednosti hitre rasti števila pametnih naprav, ki so povezane s svetovnim spletom in ustvarjajo ogromne količine podatkov. Z ustrezno analizo, obdelavo in selekcijo relevantnih podatkov oziroma dogodkov je možno spremljanje dogajanja v realnem času, kar omogoča hitro sprejemanje ustreznih odločitev in akcij ter posledično večjo učinkovitost in odzivnost organizacije oz. sistema. Lastnosti EDA in primerjava s SOA je podrobneje predstavljena v drugem poglavju.

Glavni cilj dogodkovno vodenih aplikacij in sistemov je omogočiti organizacijam spremljanje, analiziranje in odziv na relevantne spremembe v čim krajšem času. [3]

V zadnjih letih smo priča tudi hitremu razvoju na področju mobilnih naprav. Večino aplikacij in spletnih portalov se prireja, ali na novo razvija za

potrebe mobilnih naprav, saj uporabniki želijo imeti dostop do podatkov v vsakem trenutku. V okviru diplomske naloge smo razvili mobilno nadzorno ploščo za dogodkovno vodeno platformo Occapi, ki je plod raziskav in razvoja kompetenčnega centra Opcomm in je podrobneje predstavljena v tretjem poglavju. Na kratko je predstavljena njena visokonivojska arhitektura, ki je za bralca pomembna zaradi lažjega razumevanja specifikacije zahtev in samega opisa razvoja mobilne nadzorne plošče.

Tehnična zasnova in zanimivejše stvari s katerimi smo se srečali pri razvoju mobilne nadzorne plošče za pametne telefone, ki jih poganja operacijski sistem Android, so predstavljene v četrtem poglavju. Glavni razlog, da smo se v diplomskem delu odločili razviti aplikacijo za Androida in ne zgolj prilagoditi spletnega vmesnika za potrebe mobilnih naprav je v tem, da smo želeli izkoristiti prednosti, ki jih omogoča "native" mobilna aplikacija.

Oblikovanje zahtev in razvoj je potekal v sodelovanju z Laboratorijem za podatkovne tehnologije, ki deluje v okviru Fakultete za računalništvo in informatiko Univerze v Ljubljani.

Poglavje 2

Dogodkovno vodena arhitektura (EDA)

Z opazovanjem sveta okoli nas lahko kaj kmalu zaključimo, da edino konstanto v našem vsakdanjem življenju predstavljajo spremembe. Spremembe so vedno posledica nastalih dogodkov in od tukaj tudi motivacija za dogodkovno vodeno arhitekturo, katere ideja je v svetu računalništva ni nova. Glavni preboj je EDA doživela z razvojem kompleksnega procesiranja dogodkov, ki je omogočilo naprednejše procesiranje množice dogodkov z upoštevanjem njihove medsebojne odvisnosti in številnih drugih, v naprej določenih, kriterijev in pravil.

Uporaba EDA je danes prisotna na številnih področjih, kot so npr.: pametne hiše in mesta, zdravstvo, nadzor prometa, internet stvari (angl. "Internet of things"), nadzor porabe električne energije ipd. Glavni cilj je omogočiti spremljanje dogajanja v sistemu v realnem času, kar omogoči hitrejše in bolj učinkovito sprejemanje potrebnih odločitev in izvajanje ustreznih ukrepov. Nekaj možnih primerov uporabe je predstavljenih v nadaljevanju poglavja.

V tem poglavju želimo bralca seznaniti z osnovnimi lastnostmi EDA, kar je pomembno za lažje razumevanje ostalih poglavji. Pogledali si bomo prednosti in slabosti ter navedli nekaj primerov uporabe EDA. Zanimalo nas bo tudi,

kako je definiran dogodek, se na kratko spoznali s plastmi v dogodkovnem toku in si pogledali različne pristope pri njihovem procesiranju. V drugem delu poglavja so na kratko predstavljene glavne komponente s katerimi se srečamo pri implementaciji EDA ter primerjava in povezava EDA s storitveno usmerjenimi arhitekturami (angl. "service oriented architecture (SOA)").

2.1 Uvod v dogodkovno vodeno arhitekturo

"Dogodkovno vodena arhitektura predstavlja arhitekturni stil v katerem se ena ali več komponent v programskem sistemu izvede, kot odziv na prejetje enega ali več obvestil o dogodku." [2]

2.1.1 Osnovne značilnosti EDA

Začnimo s kratko predstavitvijo pojma dogodek, ki opisuje nekaj, kar se je zgodilo in predstavlja osnovno enoto EDA. Prišlo je do uresničitve dejanja in posledično opazne spremembe v resničnem svetu. S pomočjo dogodkov lahko opišemo skoraj celotno dogajanje okoli nas v določenem trenutku, kar pa je tudi eden izmed glavnih razlogov za uporabnost in zanimivost EDA.

Dogodkovno vodena arhitektura predstavlja množico arhitekturnih elementov, vključujoč dizajn, načrtovanje, tehnologijo in podobno, katerih glavna naloga je omogočiti zmožnost širjenja podatkov o dogodku do vseh odjemalcev, ki bi jih dogodek utegnil zanimati. Odjemalci morajo nato imeti možnost oceniti dogodek in pričeti z izvajanjem ustreznih akcij in ukrepov, kot odgovor na posledice dogodka. V poslovnem svetu bi tako morda kot akcijo sprožili nov poslovni proces. Za EDA velja tudi, da je zelo ohlapno sklopljena in porazdeljena. Vir novo nastalega dogodka ne ve nič o odjemalcu in akcijah, ki so bile izvedene kot posledica dogodka. Edino povezavo med posameznimi komponentami predstavlja dogovor o kodiranju podatkov dogodka in njihov pomen. [1, 3]

Glavne lastnosti, ki določajo EDA so [1]:

- je šibko sklopljena,
- uporablja metode asinhronega komuniciranja, običajno mehanizme, ki delujejo na principu objavi/naroči (angl. "publish/subscribe"),
- osnovna enota EDA je dogodek,
- vsebuje poslušalce, generatorje, procesne enote in odjemalce dogodkov. V idealnem primeru komunikacija temelji na SOAP protokolu in spletnih storitvah,
- uporablja splošno dostopne sisteme komuniciranja, ki omogočajo prenos sporočil,
- ni odvisna od centralnih kontrolnikov.

EDA omogoča [1]:

- agilnost pri operativnem upravljanju,
- vzpostavljanje povezav med podatki, ki so potrebne za izvajanje analize, oblikovanje poslovnih procesov in upravljanje,
- agilnost pri izvajanju poslovnih analiz in dinamično spreminjanje analitičnih modelov,
- dinamično upravljanje s poslovnimi pravili, ki določajo odziv na posamezne dogodke oz. serije dogodkov.

2.1.2 Razlogi za uporabo EDA

V nadaljevanju je predstavljenih nekaj razlogov iz vira [2] za uporabo EDA:

- Uporaba EDA je zelo primerna, kadar imamo opravka s problemom, ki je že po svoji naravi dogodkovno usmerjen. Pri takih primerih imamo večinoma opravka s številnimi senzorji, ki zbirajo podatke o dogodkih,

katere aplikacija kasneje analizira in preuči ter na podlagi ugotovitev izvede ustrezne akcije in ukrepe. Glavna prednost takega pristopa je v hitri odzivnosti arhitekture, ki omogoča spremljanje dogajanja v skoraj realnem času.

Za primer poglejmo spremljanje prometa. Na pomembnejših cestnih odsekih so nameščeni senzorji, ki merijo tip vozila, njegovo hitrost in ostale stvari, ki nas zanimajo. Za vsako vozilo se tako ustvari nov dogodek, ki vsebuje ustrezne podatke, ti podatki pa se posredujejo procesni enoti oz. dogodkovno vodeni platformi. Na podlagi množice dogodkov se lahko spremlja število vozil in povprečno hitrost na posameznem odseku, kar z ustrezno analizo in predikcijo omogoča napoved zastojev in posledično njihovo preprečitev. EDA omogoča tudi, da smo o povečanem prometu in ostalih izjemnih situacijah obveščeni v skoraj realnem času, kar omogoča hitrejše sprejemanje ustreznih ukrepov.

- V primerih, kadar želimo z opazovanjem in analiziranjem množice dogodkov oz. podatkov odkriti nepričakovano obnašanje sistema in nekatere v naprej določene situacije, v čim krajšem času po pojavitvi. Za razliko od serijskega pristopa, kjer občasno preverimo ali je v sistemu prišlo do morebitnih sprememb, je pri dogodkovno vodenem pristopu aplikacija o dogodku obveščena v najkrajšem možnem času. S tem dosežemo večjo odzivnost.

  Pomembno področje uporabe EDA je zdravstvo. V našem primeru si bomo pogledali primer predpisa zdravil pacientu. Zdravnik pri pacientu odkrije bolezensko stanje, za katerega je potrebno zdravljenje z zdravili. Predpis zdravil predstavlja nov dogodek v sistemu, ki nosi podatke o zdravilu, predpisani količini, načinu uporabe in podobno. Procesna enota obdela dogodek ter na podlagi zgodovine zdravljenja oz. kartoteke pacienta in sestave zdravila ugotovi, ali zdravilo vsebuje snov na katero bi znal biti pacient alergičen. V takem primeru se o tem v najkrajšem

možnem času obvesti zdravnika, da sprejme ustrezne ukrepe in morda predpiše drugo zdravilo. EDA omogoča takojšne obveščanje o izredni situaciji, ki zahteva hitro ukrepanje.

- Velika prednost dogodkovno vodenih aplikacij je šibka sklopljenost, ki je značilna med posameznimi komponentami EDA. Procesna enota, ki skrbi za procesiranje nastalih dogodkov in izvajanje ustreznih akcij na podlagi v naprej določenih kriterijev in pogojev, je ločena od aplikacije. To omogoča hitro prilagajanje aplikacije novim poslovnim zahtevam, s čimer se doseže nižja cena sprememb in večja konkurenčnost na trgu. V primeru dobre dogodkovno vodene aplikacije lahko skrbniki sami spreminjajo kriterije in lastnosti procesne enote.

  Zelo pomembno področje uporabe EDA je spremljanje oz. nadzor porabe in generiranja električne energije. To predstavlja tudi enega izmed sklopov pametnih hiš in mest, kjer želimo doseči, da je poraba energije čim bolj varčna in učinkovita. Šibka sklopljenost EDA omogoča, da lahko neodvisno in brez spreminjanja ostalega sistema dodajamo nove porabnike oz. generatorje električne energije, ki v našem primeru predstavljajo generatorje dogodkov. Obenem pa lahko, neodvisno od porabnikov oz. generatorjev, spreminjamo kriterije in pravila, ki prožijo ustrezne akcije. Kot primer poglejmo dodajanje novih sončnih celic v pametnem mestu. Sončne celice preprosto, brez ostalih sprememb v sistemu, priključimo v že obstoječe omrežje. Podatek o ustvarjeni električni energiji sončne celice sporočijo preko generiranja novih dogodkov, ki jih procesna enota brez dodatnega prilagajanja in spreminjanja ustrezno obdela. Zanima nas delež porabljene energije in ob dosegu v naprej določene meje o tem obvestimo uporabnika, ki ima možnost spreminjanja vrednosti meje, neodvisno od generatorjev dogodkov.

- Dogodkovno vodene aplikacije so zelo primerne tudi v primerih, kadar želimo z analizo velike količine podatkov priti do relevantnih rezultatov,

ki jih od nas zahteva uporabnik ali druga aplikacija. V tem primeru podatke uredimo v tok dogodkov, katerega se lahko razdeli med več procesnih enot ter se tako analizo vrši vzporedno, nad manjšimi množicami podatkov. To nam omogoči hitrejšo pot do rezultata in posledično večjo odzivnost aplikacije.

Eno od področji, kjer se uporablja EDA je odkrivanje goljufij. Kot primer poglejmo prometno nesrečo dveh osebnih vozil in prijavo škode pri zavarovalnici. Dogodki bi v tem primeru vsebovali podatke o udeležencih prometne nesreče, kraju nesreče, vrsti vozila, višini škode in podobno. V sistemu novo prispele dogodke ustrezno obdelamo, kar pomeni, da ga povežemo z ostalimi podobnimi dogodki in ugotavljamo možne časovne, prostorske, sorodstvene in druge odvisnosti. V množici dogodkov, ki lahko obsega dogodke za več let nazaj, želimo odkriti sumljive vzorce, ki bi predstavljali potencialno goljufijo. V našem primeru bi nas tako na primer zanimalo ali sta udeleženca kakorkoli posredno ali neposredno povezana, ali sta bila že kdaj skupaj udeležena v nesreči, število nesreč v katerih sta bila udeleža, tip nesreč in podobno. EDA omogoča, da iz množice dogodkov z upoštevanjem njihove medsebojne odvisnosti in na podlagi v naprej določenih pravil in kriterijev odkrivamo smiselne povezave, ki bi bile zanimive za uporabnika.

2.1.3 Slabosti EDA

Kljub številnim prednostim EDA pa je tudi nekaj negativnih vidikov. V nadaljevanju si bomo pogledali nekaj izmed njih:

- Težko, če ne celo nemogoče, je zagotoviti atomarnost, skladnost, izolacijo in trajnost (angl. “ACID - atomicity, consistency, isolation, durability”) med komponentami sistema, ki si medsebojno izmenjujejo podatke o dogodkih. Za zagotovitev integritete transakcij, se morajo le-te zaključiti v posamezni komponenti.

- Kljub dejstvu, da je EDA prisotna na trgu že nekaj časa, je na tem področju še vedno občutiti pomanjkanje standardov, ki bi vse skupaj bolj natančno opredelili.

2.2 Dogodek

"Dogodek lahko definiramo kot opazno spremembo stanja." [5]

"Dogodek je opazen pojav, ki se dogaja znotraj ali zunaj podjetja. Poslovni ali sistemski dogodek lahko predstavlja problem, priložnost, doseg praga ali odstopanje." [3]

V nadaljevanju bomo podrobneje obdelali pojem dogodek (angl. "event"), ki predstavlja osnovno enoto dogodkovno vodenih arhitektur. Prav tako bomo podrobneje pogledali posamezne plasti dogodkovnega toka in se na koncu seznanili z različnimi pristopi pri njihovi obdelavi.

2.2.1 Zgradba dogodka

Za potrebe dogodkovno vodenih arhitektur je potrebno dogodke iz realnega sveta digitalizirati in zapisati v računalniku razumljivi obliki. Posamezni dogodek je zgrajen iz glave in telesa.

Glava dogodka

Glava dogodka (angl. "event header") je sestavljena iz podatkov, ki opisujejo okoliščine v katerih se je dogodek zgodil. Vsebuje podatke o tipu dogodka, časovnem žigu, imenu dogodka, številki ponovitve dogodka, izvoru dogodka in podobno. [3]

Telo dogodka

Telo dogodka (angl. "event body") vsebuje podatke o tem, kaj se je zgodilo. Glava dogodka nosi metapodatke o okoliščinah v katerih se je dogodek zgodil, v telesu pa so zbrani vsebinski podatki o samem dogodku. Pomembno je, da telo dogodka nosi dovolj podrobne informacije o samem dogajanju ob pojavitvi dogodka, s čimer se prepreči kasnejše vračanje k njegovemu izvoru. [3]

2.2.2 Plasti dogodkovnega toka

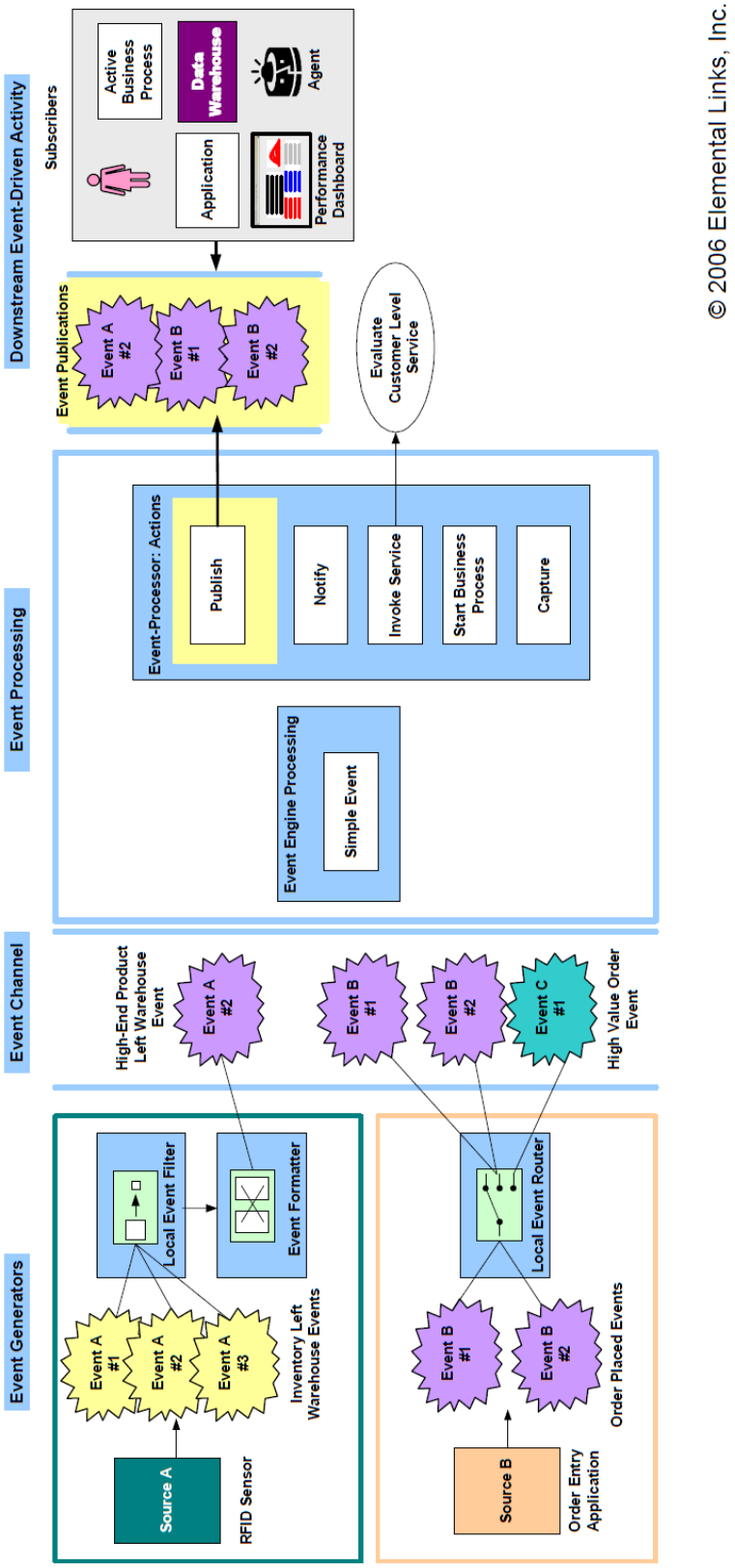
V nadaljevanju bomo podrobneje pogledali štiri plasti v dogodkovnem toku, ki so prikazane tudi na sliki 2.1.

Generator dogodka

Življenjski cikel dogodka se začne z opazno spremembo stanja na enem od virov dogodka. Vir novih dogodkov, znotraj EDA, najpogosteje predstavljajo razne aplikacije, storitve, poslovni procesi, senzorji ipd. Sprememb stanja v sistemu je običajno veliko, zato se pred generiranjem novega dogodka s pomočjo dogodkovnega predprocesorja in na podlagi v naprej določenih kriterijev oceni, ali je generiranje le-tega smiselno. Zaradi množice različnih virov dogodkov je pred oddajo dogodka potrebno zagotoviti transformacijo podatkov v naprej določeno, standardizirano obliko. [3]

Kanal prenosa dogodka

Glavna naloga kanala prenosa dogodka (angl. "event channel") je transport novo ustvarjenega dogodka med viri dogodkov, procesnimi enotami in nadaljnjimi odjemalci, na katere vpliva dogodek. [3]



Slika 2.1: Plasti v dogodkovnem toku (vir: [3])

Obdelava dogodkov

Dogodki se po kanalu prenesejo od vira do procesnih enot, kjer se vrši obdelava dogodka. V procesnih enotah uporabnik v naprej določi kriterije in merila na podlagi svojih potreb in zahtev ter glede na to kaj je zanj zanimivo in pomembno. Ob vstopu novega dogodka v procesno enoto se dogodek obdela na podlagi predhodno postavljenih pogojev in meril ter se prične z izvajanjem ustreznih akcij. Pod akcijami imam tukaj v mislih predvsem začetek novega poslovnega procesa, obveščanje posameznikov oz. sistemov, generiranje novih dogodkov in podobno. [3]

Posledične aktivnosti dogodka

Posamezni dogodek oz. korelacija več dogodkov lahko vodi do številnih posledičnih aktivnosti dogodka (angl. "downstream event-driven activity"). To predstavlja zaključno fazo v življenjskem ciklu dogodka. Tipov odjemalcev, za katere je dogodek zanimiv, je lahko več. Pod pojmom odjemalci imamo tukaj v mislih predvsem ljudi, aplikacije, podatkovne shrambe ipd. [3]

2.2.3 Procesiranje dogodkov

"Obdelava dogodkov (angl. "event processing") je proces, v katerem se izvede množica operacij nad dogodkom. Tipične operacije so branje, kreiranje, transformiranje in brisanje dogodkov." [2]

Preprosto procesiranje dogodkov

O preprostem procesiranju dogodkov (angl. "simple event processing") govorimo takrat, kadar imamo opravka s specifičnimi, merljivimi spremembami stanja, kot je na primer sprememba temperature. Torej gre za pomembne dogodke, ki na podlagi podatkov v telesu dogodka vodijo do pričetka izvajanja določenih akcij. Preprosto procesiranje dogodkov je primerno predvsem za primere, ko želimo celotno dogajanje spremljati v realnem času. [3, 6]

Tokovno procesiranje dogodkov

Pri tokovnem procesiranju dogodkov (angl. "stream event processing") nas zanima, kot že samo ime pove, predvsem tok dogodkov. Pri tej vrsti procesiranja procesna enota dobiva tok podatkov, katerega vir so lahko na primer: RFID čitalci, termometer in podobno. Glavni cilj procesiranja toka dogodkov je odkrivanje pomembnih dogodkov na podlagi v naprej določenih kriterijev. Uporabnika se obvešča samo o dogodkih, ki so zanj zanimivi in zahtevajo takojšnje ukrepanje. Tak primer procesiranja dogodkov je primeren pri spremljanju toka informacij znotraj sistema v realnem času, kar omogoča pravočasno sprejemanje odločitev. [1, 3]

Kompleksno procesiranje dogodkov

Kompleksno procesiranje dogodkov (angl. "complex event processing") je EDA dvignilo na povsem nov nivo. Opravka imamo z več različnimi tokovi dogodkov in tudi individualnimi dogodki, ki so se zgodili v daljšem časovnem obdobju. Kompleksna procesna enota nato med vsemi množicami dogodkov, na podlagi v naprej določenih logičnih pogojev in kriterijev, ugotavlja njihovo medsebojno prostorsko, časovno ali slučajno odvisnost, ter na podlagi le-te pride do smiselnih zaključkov, ki bi bili zanimivi za uporabnika. Z drugimi besedami kompleksna procesna enota "razmišlja" kot človek, za kar so potrebna napredna orodja za interpretacijo dogodkov, odkrivanje vzorcev v množici dogodkov, ugotavljanje soodvisnosti ipd. V današnjem času je kompleksno procesiranje dogodkov prisotno predvsem pri odkrivanju in odzivu na morebitne anomalije, grožnje in priložnosti znotraj podjetja. Zelo pomembno področje pa predstavlja tudi odkrivanje goljufij. [1, 3, 6]

2.3 Komponente implementacije EDA

2.3.1 Metapodatki dogodka

Dobro zasnovana EDA ima natančno definirano arhitekturo metapodatkov, v katerih so zapisani podatki o lastnostih oz. specifikah dogodka ter pravilih njegove obdelave. Specifikacija dogodka mora biti dostopna in razumljiva virom dogodka, procesnim in transformacijskim enotam ter odjemalcem. [3]

2.3.2 Procesiranje dogodka

Procesiranje dogodkov poteka v procesnih enotah na podlagi v naprej določenih pravil in kriterijev. Pri procesiranju se večinoma uporablja podatke iz telesa dogodka. Za preproste dogodke je razmeroma preprosta tudi njihova obdelava, zato lahko procesno enoto razvijemo in implementiramo kar sami. Pri kompleksnih dogodkih pa se večinoma uporablja procesna enota od enega izmed večjih ponudnikov, ki jo integriramo v naš sistem. Podatke o dogodku se shrani za kasnejše potrebe in analize. [3]

2.3.3 Orodja za delo z dogodki

Orodja za upravljanje nam omogočijo administracijo in nadzor infrastrukture za obdelavo dogodkov ter spremljanje toka dogodkov. Na drugi strani pa imamo razvojna orodja, ki so potrebna za definiranje specifik dogodka, procesnih pravil ter za upravljanje naročnikov dogodkov. [3]

2.3.4 Integracija v podjetje

Integracijska hrbtenica predstavlja pomembno vlogo v EDA. Pri integraciji so pomembne predvsem naslednje storitve: procesiranje dogodkov, sprožitev poslovnega procesa, transportni kanal dogodkov, objava in naročanje, ter dostop do podatkov podjetja. [3]

2.3.5 Viri in cilji

Pod viri in cilji imamo v mislih resurse podjetja, kot so aplikacije, storitve, poslovni procesi, podatkovne shrambe, ljudje ipd. Njihova glavna naloga je generiranje dogodkov in izvajanje ustreznih akcij. [3]

2.4 EDA in SOA

V nadaljevanju bomo pogledali razlike, podobnosti in možnosti sodelovanja storitveno usmerjene in dogodkovno vodene arhitekture.

SOA je primerna predvsem v primerih, ko želimo vzpostaviti dvosmerno komunikacijo med posameznimi entitetami sistema na podlagi v naprej dogovorjenega vmesnika. V teh primerih komunikacija temelji na principu zahteva-odgovor (angl. "request-response"), kjer ena stran poda zahtevo, druga stran pa zahtevo obravnava in vrne odgovor. EDA po drugi strani omogoča enosmerno komunikacijo, torej posledično ne omogoča interakcije obeh strani. S tem je dosežena zelo majhna sklopljenost komponent omenjene arhitekture. Vir dogodkov ne ve ničesar o posledicah, ki jih je dogodek povzročil in ničesar o odjemalcih, ki so bili o dogodku obveščeni. EDA zaradi teh lastnosti omogoča bistveno bolj učinkovito, hitrejše in cenejše prilagajanje na spremembe v poslovnem okolju. Za vsako od arhitektur obstaja več različnih možnosti in pristopov pri implementaciji, vendar večina današnjih primerov implementacije temelji na uporabi spletnih storitev (angl. "web services").

Obe arhitekturi omogočata gradnjo večjih sistemov in aplikacij na podlagi sestavljanja manjših programskih modulov. Prav tako je pri obeh sama implementacija ločena od specifikacije interakcije preko vmesnika. Za različne implementacije SOA je značilno, da se storitev izvaja linearno in je njen potek dirigiran s strani uporabnika. Ko se enkrat prične tok izvajanja operacij znotraj storitve, le-ta poteka avtonomno in nanj ne vplivajo zunanji vhodi. Za razliko od SOA, EDA omogoča dinamično, asinhrono in vzporedno izvajanje procesa. Vir dogodka nima nobenega nadzora nad kasnejšim tokom ustvarje-

nega dogodka zunaj ali znotraj sistema. V proces lahko vstopajo tudi zunanji dogodki, ki spreminjajo potek izvajanja. EDA je bolj primerna za spremljanje dogajanja v realnem času, saj so vsi odjemalci v sistemu o dogodku obveščeni skoraj takoj, ko se le-ta zgodi. [4]

Spodaj sta na kratko predstavljeni dve možnosti sodelovanja obeh arhitektur [3]:

- Ko se dogodek pojavi na viru, se le-ta razpošlje vsem odjemalcem, ki bi jih dogodek zanimal. Na podlagi dogodka se odjemalci odločijo katere akcije bodo izvedli, kot odgovor na novo nastalo stanje v sistemu. Kot akcijo si lahko predstavljamo začetek izvajanja ene ali več storitev. Te storitve lahko izvajajo preproste operacije ali pa celotne poslovne procese. Tak primer interakcije med dogodki in storitvami se običajno navaja kot oblika dogodkovno vodene storitveno usmerjene arhitekture, ki je bolj poznana pod oznako SOA 2.0.

- Storitve lahko nastopajo tudi kot vir dogodka. Dogodek lahko označuje problem, priložnost, doseg neke, v naprej postavljene meje ali odstopanja, ki bi lahko bilo zanimivo za odjemalca in morda zahteva začetek izvajanja ustreznih ukrepov. Odjemalci so o dogodku obveščeni v najkrajšem času, ko se le-ta pojavi na viru. Torej smo tudi tukaj priča sodelovanju in interakciji storitev ter dogodkov.

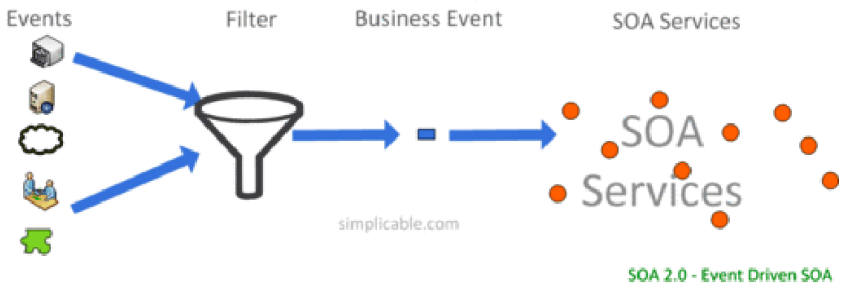
2.4.1 Kdaj je EDA bolj primerna kot SOA in obratno

Dogodkovno vodena in storitveno usmerjena arhitektura v veliko primerih nastopata skupaj, saj se IT strokovnjaki vedno bolj zavedajo prednosti prepletanja obeh arhitektur in to izkoriščajo. SOA tako lahko nastopa kot vir in odjemalec dogodkov znotraj EDA. Vseeno pa je med njima veliko razlik, zato je potrebno v vsaki situaciji dobro preučiti katera od arhitektur je bolj primerna. V nadaljevanju je predstavljenih nekaj smernic (vir: [4]) za izbiro ustrezne arhitekture:

- SOA je bistveno bolj primerna za primere, ko želimo, da je nova poslovna aplikacija zgrajena iz že obstoječih sistemov.
- SOA je bolj primerna za situacije, ko imamo opravka s poizvedovalnimi sistemi, kjer se kot vhod poda zahteva in se kot rezultat dobi podatke, ki ustrezajo tej zahtevi.
- EDA je najbolj primerna izbira v sistemih, ki opravljajo nadzor. Še posebno kadar imamo opravka s spremljanjem dogajanja v realnem času. EDA omogoča tudi hitrejšo odzivnost sistema, saj je uporabnik oz. odjemalec o dogodku obveščen skoraj takoj, ko se le-ta zgodi.
- EDA je bolj primerna za komunikacijo med podjetji (angl. "business-to-business"). Predvsem na račun dejstva, da so poslovni sistemi med seboj v večini primerov šibko povezani, saj se v vsakem izvajajo poslovni procesi ločeno in neodvisno od drugega poslovnega sistema. Komunikacija med poslovnimi sistemi tako poteka preko poslovnih dogodkov, ki opisujejo spremembo stanja v poslovnem procesu poslovnega sistema.
- EDA je zelo primerna v primerih, ko imamo opravka z visoko dinamičnimi sistemi, znotraj katerih smo priča velikemu številu sprememb stanja in posledično dogodkov. Prej omenjena arhitektura je že v svoji naravi bolj prilagodljiva na spremembe in nove, nedefinirane tokove dogodkov.

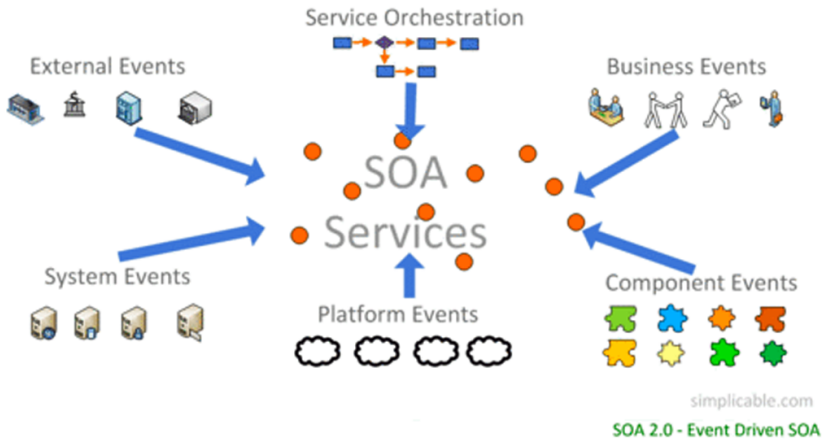
2.4.2 SOA 2.0

Dogodkovno vodena storitveno usmerjena arhitektura (angl. "event-driven SOA") oziroma na kratko SOA 2.0 (slika 2.2), je oblika SOA, ki združuje prednosti EDA z organizacijskimi prednostmi, ki jih ponujajo storitve. Za SOA je značilno, da orkestrira potek storitve preko v naprej definiranega poslovnega procesa, ki ne predvideva dogodkov, ki se lahko zgodijo znotraj ali zunaj poslovnega procesa. S SOA 2.0 se uporabniku omogoči večjo odzivnost celotnega sistema, nadzor, analizo in iskanje povezav med različnimi dogodki (slika 2.3),



Slika 2.2: SOA 2.0 (vir: [8])

ki sprva niso očitne. V nadaljevanju je predstavljenih še nekaj prednosti in novosti (vir: [7]), ki jih prinaša SOA 2.0:



Slika 2.3: Podrobneje predstavljeni dogodki SOA 2.0 (vir: [8])

- Predstavlja možnost ustvarjanja visoko nivojskih poslovnih dogodkov, na podlagi filtriranja številnih nizko nivojskih sistemskih dogodkov v realnem času. Vir sistemskih dogodkov so lahko razne aplikacije, podatkovne baze in podobno. Pridobljene podatke združi s podrobnostmi o morebitnih povezavah, ki so ugotovljene na podlagi ugotavljanja odvisnosti z ostalimi dogodki.

- SOA 2.0 uvaja možnost dolgotrajnega procesiranja, kar omogoči zbiranje številnih podatkov o asinhronih dogodkih skozi daljše časovno obdobje. Nad temi podatki se nato ugotavlja njihova medsebojna odvisnost ter na podlagi v naprej določenih kriterijev se v množici dogodkov odkriva vzorce, ki so lahko vzrok za začetek novega poslovnega procesa oziroma katere druge akcije.

- SOA 2.0 je osnovana na osnovi nizke sklopljenosti, ki je značilna za EDA. Odjemalce dogodkov ne zanima kje in zakaj se je dogodek zgodil. Glavna zahteva je, da so o dogodku obveščeni v najkrajšem možnem času. Zelo pomembno vlogo tako igra transportni kanal, ki skrbi za prenos podatkov o dogodku, od vira do uporabnikov oz. odjemalcev.

Poglavje 3

Occapi platforma

Occapi platforma je ena izmed rešitev, ki so jo pripravili v kompetenčnem centru Opcomm. Gre za dogodkovno vodeno platformo, ki zbira, integrira in shranjuje podatke iz množice raznih naprav, kot so: senzorji, merilniki, pametne kartice ipd. Tako pridobljene podatke nato obogati z internetnimi podatki ter s podatki drugih baz, ter jih analizirane in pomensko obdelane ponuja drugim aplikacijam ter storitvam. [9]

Potencialni uporabniki Occapi platforme so telekomunikacijski operaterji, ponudniki naprednih internetnih storitev in aplikacij ter sistemski integratorji, ki lahko platformo prilagodijo in namestijo za svoje potrebe, kot so: spremljanje prometa, pametne hiše in mesta, upravljanje električne energije in podobno. Glavni namen uporabe platforme je izkoristiti prednosti, ki jih ponujajo s svetovnim spletom povezane pametne naprave in senzorji, katerih število se v zadnjem času bliskovito povečuje in s tem posledično narašča tudi količina ustvarjenih podatkov. Zaradi velike količine in različnih tipov podatkov se povečuje potreba po njihovi klasifikaciji in analizi, ki loči pomembne podatke od manj pomembnih, ter pomenski obdelavi, ki omogoči, da se na podlagi podatkov sprejema najbolj ustrezne in racionalne odločitve.

V nadaljevanju poglavja so na kratko predstavljeni pomembnejši deli iz visokonivojske arhitekture platforme Occapi, ki je prikazana na sliki 3.1.

3.1 Opis arhitekture

Generatorji dogodkov

Vhod v platformo predstavljajo dogodki, ki se ustvarjajo v opazovanem okolju. Vir dogodka predstavljajo razni senzorji, naprave, SOA/EDA okolja ter zastareli sistemi. Komunikacija s platformo poteka preko vmesnikov, ki so za večino sodobnejših naprav standardizirani. Prav tako so relativno enostavni vmesniki za SOA/EDA okolja. Večji problem nastane pri zastarelih sistemi, kjer je potrebno vmesnik prilagoditi vsakemu sistemu posebej.

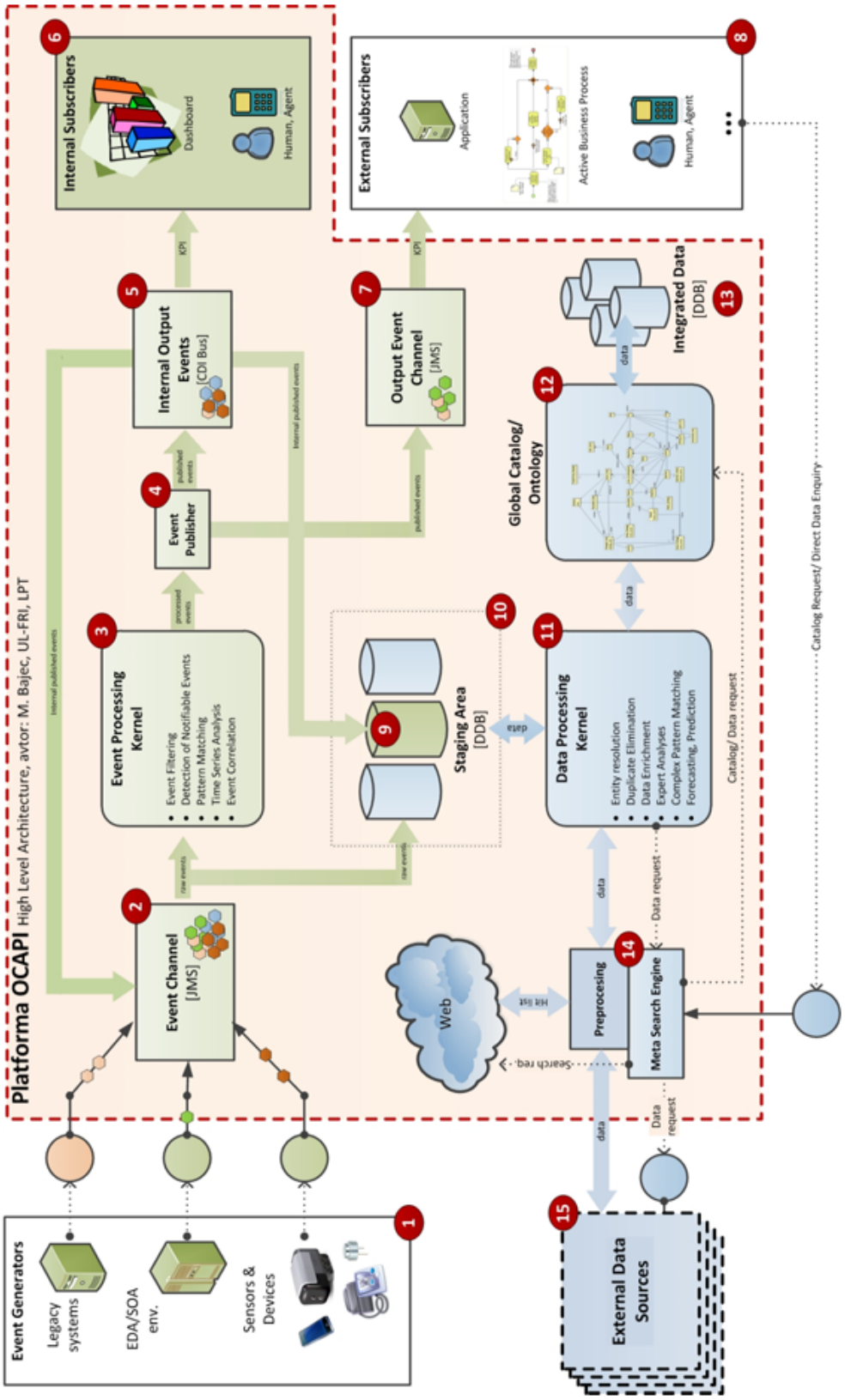
Vhodna vrsta

Dogodki, ki vstopijo v platformo, se zapišejo v vhodno vrsto, od koder so kasneje posredovani na dva kanala: (a) v komponento za procesiranje dogodkov ter (b) v podatkovno bazo. Komunikacija je asinhrona.

Procesiranje dogodkov

Znotraj platforme se dogodki procesirajo. V sklopu komponente za procesiranje je omogočeno:

- **Filtriranje dogodkov:** komponenta dogodke razvrsti po v naprej oblikovanih skupinah, odstrani dvojnike, doda metapodatke in podobno.
- **Detekcija pomembnih dogodkov:** glavna naloga komponente je, da v množici dogodkov razpozna pomembne dogodke, ki so zanimivi za prejemnike. Kot rezultat, komponenta pripravi dogodke, ki vsebujejo vse potrebne podatke in so primerni za objavo.
- **Identifikacija vzorcev:** komponenta omogoča odkrivanje vzorcev v množici dogodkov, ki lahko nakazujejo možnost obstoja ali pojava določene želene ali neželene situacije. Vzorci, ki jih komponenta išče, morajo biti v naprej definirani.



Slika 3.1: Visokonivojska arhitektura platforme Occapi (vir: [10])

- **Analiza časovnih vrst:** omogoča obravnavo množice dogodkov, ki so se pojavili v nekem časovnem intervalu. Velikost časovnega okna, za potrebe analize, je podana kot vhodni parameter in je odvisna od zmogljivosti strojne opreme.
- **Korelacija med dogodki:** komponenta skrbi za ugotavljanje odvisnosti med posameznimi dogodki, kar je zelo pomemben podatek pri analizi časovnih vrst in identifikaciji vzorcev. Poznavanje odvisnosti med dogodki nam omogoči, da z večjo verjetnostjo napovemo pojav drugega dogodka ali skupine dogodkov.

Objava dogodkov

Ob izhodu iz komponente za procesiranje dogodkov, se t. i. generirani dogodki posredujejo komponenti za objavo dogodkov, ki poskrbi, da so o dogodku obveščeni vsi zainteresirani prejemniki.

Interni naročniki

Interni naročniki lahko dogajanje znotraj platforme spremljajo preko centralne nadzorne plošče, kjer sami določijo, kateri ključni zmogljivostni indikatorji (angl. "Key Performance Indicator - KPI") jih zanimajo in jih želijo spremljati. KPI so lahko enostavni (posamezen dogodek, npr.: trenutna hitrost vozil na določenem odseku) ali kompleksni (sestavljeni iz več dogodkov, npr.: trenutna hitrost posameznih vrst vozil v prometu). Poseben primer KPI, ki ga lahko sistem posreduje tudi na elektronski naslov ali prek SMS-a, je alarm.

V naslednjem poglavju je predstavljen razvoj centralne nadzorne plošče za mobilne naprave z Androidom.

Eksterni naročniki

Eksterni naročniki so lahko npr.: SOA procesi, zunanje aplikacije in podobno, ki so o dogodkih obveščeni prek zunanje oziroma eksterne izhodne vrste. Plat-

forma ne omejuje sistemov oziroma zunanjih naročnikov na dogodke.

Začasna hramba podatkov

V primeru visoke frekvence nastajanja novih dogodkov postane zapisovanje v integrirano podatkovno bazo ozko grlo sistema. Zato je v okviru platforme predvideno distribuirano shranjevanje podatkov, pri čemer je nivo distribucije odvisen od intenzivnosti nastajanja dogodkov. Začasna hramba podatkov je torej namenjena, kot že samo ime pove, začasnemu shranjevanju podatkov.

Integrirana podatkovna baza

Glavna naloga integrirane podatkovne baze, ki je porazdeljena in nameščena na klasičnem podatkovnem strežniku, je hranjenje podatkov, skladnih z globalnim katalogom. Baza je namenjena tudi direktnemu poizvedovanju ter poglobljenim analizam.

Procesiranje podatkov

Komponenta za procesiranje podatkov omogoča:

- **Razpoznavo entitet:** komponenta je zmožna razpoznati podatke, ki predstavljajo primere konceptov in so opredeljeni v globalnem katalogu. Podatki, ki jih komponenta ne razpozna, se ignorirajo.
- **Eliminacijo duplikatov:** komponenta zazna morebitne dvojnike v sistemu in jih eliminira.
- **Obogatitev podatkov:** komponenta za vsako entiteto preveri, ali vsebuje vse podatke, ki so predvideni v globalnem katalogu. Če določeni podatki manjkajo, jih komponenta poskusi pridobiti iz alternativnih virov, kot npr.: javno dostopni podatki na spletu. Gre za zelo uporabno in pomembno, vendar kompleksno funkcionalnost komponente.

- **Ekspertne analize:** zaradi domenske specifičnosti v platformi niso v naprej realizirane. Obstaja pa možnost razširitve.
- **Identifikacijo kompleksnih vzorcev:** komponenta omogoča odkrivanje vzorcev, ki morajo biti v naprej definirani in bi lahko biti zanimivi za uporabnika.
- **Predikcijo in napovedovanje:** komponenta vsebuje tudi funkcije za enostavno napovedovanje in predvidevanje.

3.2 Konkurenčne platforme

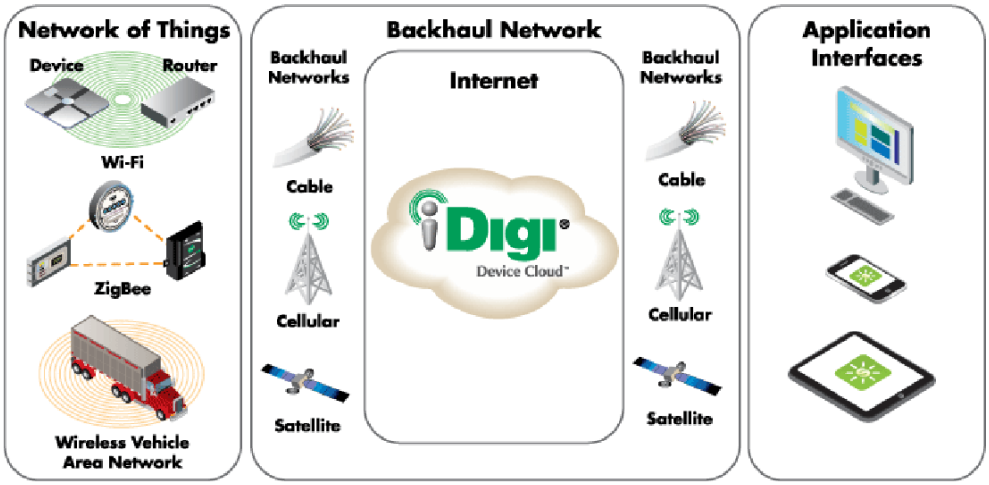
3.2.1 iDigi

iDigi[1] ponuja rešitev *iDigi Device Cloud*, ki predstavlja infrastrukturno storitev, zasnovano za namen zagotavljanja povezave med aplikacijami in omrežji naprav. Oblačne tehnologije omogočajo, da uporabniške aplikacije do podatkovnih storitev dostopajo preko v naprej določenih vmesnikov. Obenem pa skrijejo kompleksnost integracije M2M in omogočajo vključevanje novih naprav brez spreminjanja aplikacije. iDigi Device Cloud arhitektura torej omogoča razvijalcu, da se osredotoči na razvoj aplikacije in posveti zelo malo časa ukvarjanju z infrastrukturo ter morebitnim vprašanjem glede zanesljivosti, razširljivosti in varnosti. Njihov cilj je omogočiti možnost povezave česarkoli, kjerkoli s katerokoli aplikacijo. Vloga platforme je prikazana na sliki 3.2.

V nadaljevanju je na kratko predstavljenih šest ključnih stebrov iDigi Device Cloud rešitve [11].

- **Povezovanje z napravami:** iDigi uporablja protokol za integracijo naprav, ki vključuje možnost pošiljanja in prejemanja podatkov na številne različne načine. Protokol je odprt in dostopen vsem proizvajalcem naprav.

[1] www.iDigi.com

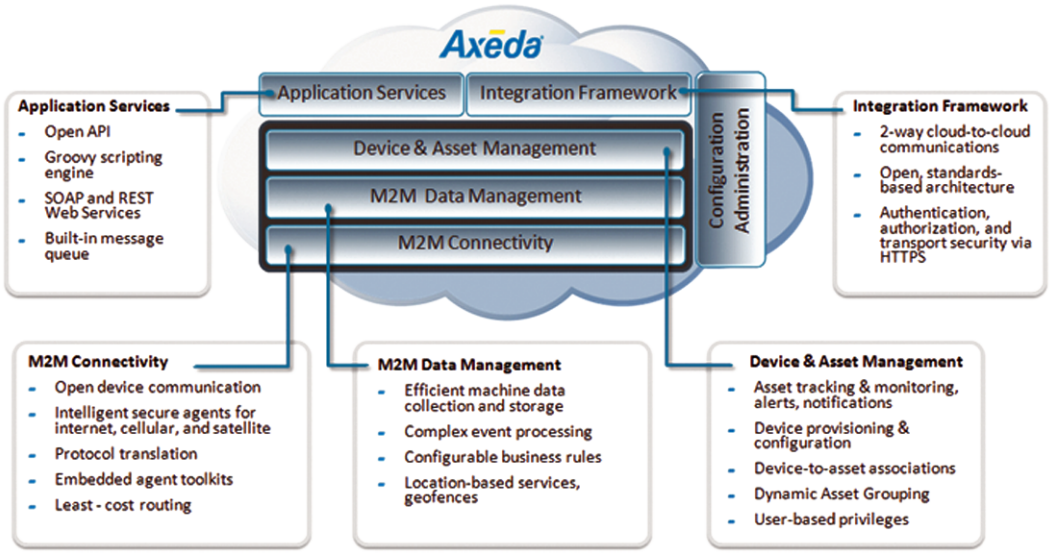


Slika 3.2: iDigi Device Cloud (vir: [11])

- **Razširljivost:** iDigi je grajen tako, da omogoča prožno in učinkovito uporabo resursov, zato povečanje števila registriranih naprav ne zahteva posebnega načrtovanja in časa za zagotovitev ustreznih virov v oblaku.
- **Integracija z aplikacijo:** za dostop do iDigi-ja so razvijalcem na voljo številni vmesniki (API).
- **Zanesljivost:** zanesljivost je izražena kot celotni čas, ko je storitev na voljo. iDigi zagotavlja enako razpoložljivost, kot jo zagotavljajo mediji, omrežja, ki se uporabljajo za prenos podatkov.
- **Zmogljivost:** aktivno se spremlja pretok dejanskih in predvidenih transakcij ter njihova uspešnost. Aplikacije za naprave so bralno intenzivne rešitve, zato se na podlagi števila branj na sekundo ugotavlja zahteve glede pretoka transakcij.
- **Varnost:** oddelek za varnost in zasebnost je izoblikoval varnostno politiko, ki temelji na 164 kontrolnih elementih, ki se jih aktivno nadzoruje in se ugotavlja njihovo skladnost z varnostno politiko.

3.2.2 Axeda

Axeda[2] platforma omogoča celovito M2M podatkovno integracijo in razvoj aplikacij. Infrastruktura je na voljo kot storitev v oblaku. Ponašajo se z visokim nivojem razširljivosti in varnosti. Obenem pa so na voljo številna razvojna orodja in prilagodljivi vmesniki (API), ki omogočajo hiter razvoj in integracijo po meri izdelanih M2M aplikacij v že obstoječ sistem.



Slika 3.3: Axeda platforma (vir: [12])

Spodaj je podrobneje predstavljenih nekaj komponent Axeda platforme s slike 3.3 [12].

- **M2M aplikacijske storitve** omogočajo razvijalcem, da razširijo in prilagodijo funkcionalnosti platforme preko napredne in vgrajene skriptne enote. Prav tako jim je na voljo široka množica SOAP in REST spletnih storitev.

[2] www.axeda.com

- **M2M integracijsko ogrodje** omogoči hitro integracijo že obstoječih sistemov z Axeda platformo.

- **M2M podatkovno upravljanje** skrbi za procesiranje in shranjevanje vhodnih M2M podatkov. Upravlja z različnimi tipi naprav, podatkov, pravicami za dostop, uporabniki, uporabniškimi skupinami ipd.

- **M2M povezljivost** predstavlja skupino programskih agentov, knjižnic, orodji in storitev, ki skrbijo za komunikacijo med napravami in Axeda platformo. Podpira več različnih metod komunikacije.

3.2.3 ioBridge

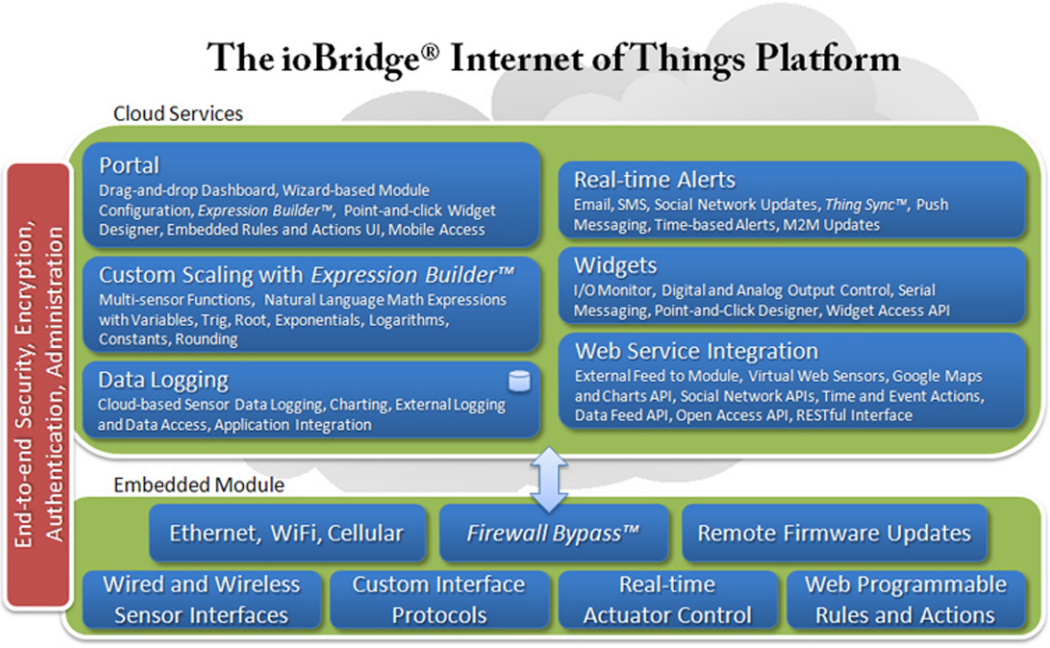
ioBridge[3] ponuja dve glavni tehnološki komponenti, ki omogočata preprosto in cenovno učinkovito spremljanje in povezovanje v internet. Prvo komponento predstavlja ioBridge platforma, ki je enako, kot že prej omenjene platforme, na voljo kot storitev v oblaku, s čimer so se izognili težavam pri namestitvi in morebitnih razširitvah. Drugo komponento pa predstavlja preprosta strojna oprema z ustreznimi gonilniki, ki omogoči priključitev raznih naprav v internet in integracijo s platformo. Elementi ioBridge platforme so prikazani na sliki 3.4.

V nadaljevanju je predstavljenih nekaj prednosti s katerimi se ponaša ioBridge platforma [13]:

- Za doseg enake funkcionalnosti se lahko pri ioBridge tehnologiji uporabi nizko cenovne mikrokontrolerje in ni potrebna uporaba dražjih procesorjev, kot je to praksa pri večini ostalih operacijskih sistemih.

- Omogoča preprosto namestitev in upravljanje. Običajno pri priključitvi novih naprav v sistem ni potrebno narediti večjih sprememb v nastavitvah, kar zniža stroške vzdrževanja in namestitve. Za dolgoročno vzdrževanje

[3] www.iobridge.com



Slika 3.4: ioBridge platforma (vir: [13])

platforma predvideva oddaljeno posodabljanje gonilnikov, posredovanje obvestil in razne podporne programe.

- Za komunikacijo med storitvami v oblaku in napravami se uporablja 128-bitno kriptiranje.
- Omogoča lahko prilagodljivost in razvoj namenskih aplikacij, saj imajo razvijalci dostop do vmesnika (API) za komuniciranje z oblačnimi storitvami.

Poglavje 4

Mobilna nadzorna plošča za Android OS

Priča smo bliskovitemu razvoju in povečanju zmogljivosti mobilnih naprav, ki uporabnikom omogočajo nadzor in dostop do informacij v vsakem trenutku, ne glede nato, kje se nahajajo. Večina aplikacij za osebne računalnike se v prirejeni obliki seli na mobilne telefone, zato smo se odločili za razvoj nadzorne plošče za naprave z operacijskim sistemom Android, ki bi uporabnikom v vsakem trenutku omogočila spremljanje in nadzor dogajanja v dogodkovno vodeni platformi Occapi. Uporabnik je prav tako v najkrajšem možnem času obveščen, da se je zgodil nepričakovan oz. izjemen dogodek, ki zahteva njegovo pozornost in ustrezno ukrepanje. Z uporabo mobilne nadzorne plošče postane nadzor stanja in sprejemanje odločitev hitrejše in bolj učinkovito. Za razvoj aplikacije za operacijski sistem Android in ne zgolj prilagoditev spletnega vmesnika, smo se odločili zato, ker smo želeli izkoristiti vse prednosti in funkcionalnosti, ki jih ponuja "native" aplikacija.

Poglavje je v grobem razdeljeno na dva dela. V prvem delu si bomo bolj podrobno pogledali tehnično zasnovo aplikacije, v drugem pa se bomo osredotočili na zanimivejše dogodke, s katerimi smo se srečali pri razvoju in se spoznali z uporabniškim vmesnikom aplikacije.

4.1 Tehnična zasnova aplikacije

Pred začetkom razvoja smo zasnovali podatkovni model, pripravili specifikacijo zahtev ter izbrali ustrezne tehnologije in orodja za realizacijo. Odločili smo se za razvoj nadzorne plošče za operacijski sistem Android, pri čemer je razvoj potekal v razvojem okolju Eclipse. Pri oblikovanju zahtev smo se osredotočili na funkcionalnosti, ki so primerne za mobilne telefone.

4.1.1 Uporabljena tehnologija in orodja

Android

Android je operacijski sistem za mobilne naprave, ki je osnovan na operacijskem sistemu Linux. Leta 2005 je Android, po prevzemu, prišel pod okrilje podjetja Google in takrat se je začel njegov bliskovit razvoj. Je odprtokoden operacijski sistem, večina izvorne kode pa je izdana pod licenco Apache, kar v praksi pomeni, da lahko vsak, ki želi uporabljati Android, izvorno kodo brezplačno prenese z interneta ter jo prilagodi glede na svoje potrebe.

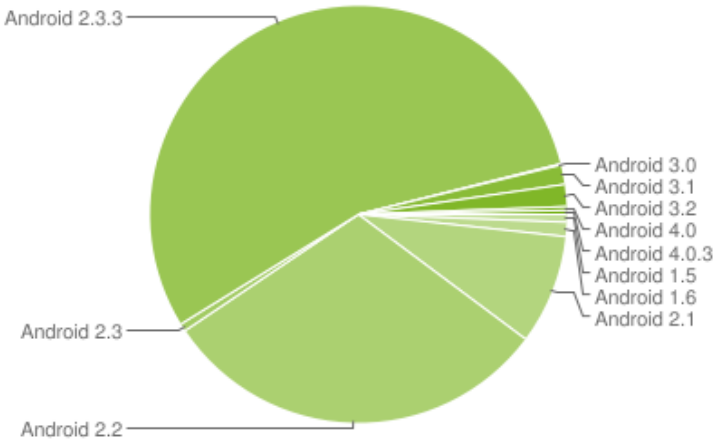
Verzija	Kodno ime	Delež
1.5	Cupcake	0,6 %
1.6	Donut	1,1 %
2.1	Eclair	8,5 %
2.2	Froyo	30,4 %
2.3.x	Gingerbread	55,5 %
3.x	Honeycomb	3,3 %
4.x	Ice Cream Sandwich	0,6 %

Tabela 4.1: Verzije Androida in deleži naprav (vir: [14])

Android omogoča enoten pristop pri razvoju, kar razvijalcem omogoča preprost razvoj aplikacij za različne naprave. To pa je le nekaj razlogov, ki so

omogočili, da je Android danes prisoten na številnih različnih napravah ter je tako dostopen najširšemu naboru uporabnikov.

Do sedaj je bilo izdanih že kar nekaj verzij Androida, ki so podrobneje predstavljene v tabeli 4.1. Zbrani so podatki o različnih verzijah Androida s pripadajočimi kodnimi imeni in deležem naprav. Podatki o deležu naprav s posamezno verzijo Androida iz tabele 4.1 so predstavljeni tudi na sliki 4.1.



Slika 4.1: Delež naprav z določeno verzijo Androida (vir: [14])

SQLite

Za potrebe shranjevanja podatkov, lokalno na mobilni napravi, smo uporabili SQLite, ki je ACID skladen sistem za upravljanje relacijskih podatkovnih baz in je že vključen v operacijski sistem Android. Za razliko od večine drugih SQL podatkovnih baz SQLite nima posebnega strežniškega procesa, ampak bere in piše neposredno v navadno datoteko na datotečnem sistemu. Celotna SQL podatkovna baza, torej vse tabele, sprožilci, pogledi in relacije, so tako zbrane in zapisane v eni sami datoteki.

Spletna storitev REST

Komunikacijo med mobilno aplikacijo in zalednim sistemom smo osnovali na principu RESTful spletne storitve. Glavni razlog za slednjo odločitev je posledica dejstva, da Android že vključuje knjižnice, ki so potrebne za delo z RESTful spletnimi storitvami in JSON nizi, ki predstavljajo tip podatkov, ki jih vrača spletna storitev. Za delo s klasičnimi SOAP spletnimi storitvami se običajno uporablja zunanja knjižnica. RESTful spletna storitev je tako bolj preprosta za implementacijo, obenem pa za enak rezultat zahteva prenos bistveno manjše količine podatkov.

RESTful spletna storitev je osnovana na HTTP protokolu in principih REST arhitekture. Pri RESTful spletni storitvi govorimo o zbirki resursov, ki so opredeljeni z naslednjimi štirimi vidiki:

- URI naslov za dostop do storitve (http://primeri.si/resursi/),
- Tipi podatkov podprti s strani spletne storitve (XML, JSON, YAML),
- Množica operacij spletne storitve, ki so realizirane s pomočjo HTTP metod (GET, PUT, POST, DELETE),
- API mora biti osnovan in voden na osnovi hiperteksta.

Eclipse

Za razvoj mobilne nadzorne plošče smo uporabili razvojno okolje Eclipse (verzijo Indigo). Poleg okolja Eclipse moramo namestiti tudi komplet za razvoj programske opreme za operacijski sistem Android (angl. "Android SDK"), ki vsebuje emulator, knjižnice, dokumentacijo, razhroščevalnik ipd. Za konec moramo v okolju Eclipse namestiti še razširitev ADT (angl. "Android Development Tools"), ki nam omogoči še nekaj dodatnih funkcionalnosti kot so: kreiranje novih projektov, izvoz aplikacij, iskanje napak v kodi in podobno.

4.1.2 Specifikacija zahtev

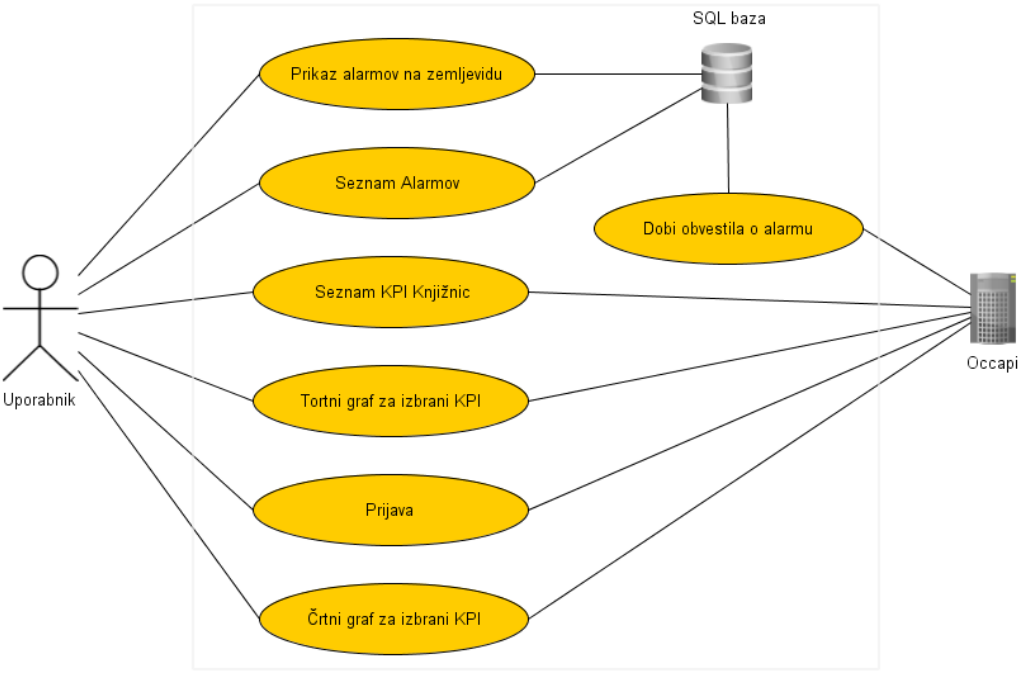
Specifikacija zahtev mora zajemati vse funkcionalnosti, ki jih pričakujemo od končne aplikacije. Cilj in glavna zahteva naše aplikacije je, da uporabniku omogočimo spremljanje trenutnega dogajanja v platformi Occapi oz. opazovanem sistemu, v realnem času. Poseben poudarek je potrebno posvetiti uporabniški izkušnji, saj želimo preprosto aplikacijo, katere uporaba bo za uporabnika intuitivna. Pri snovanju in oblikovanju uporabniškega vmesnika je potrebno upoštevati, da lahko aplikacija teče na zmogljivostno in fizično različnih napravah. Temu primerno pa moramo prilagoditi tudi razvoj in testiranje. Potrebno je zagotoviti varno povezavo med platformo Occapi in mobilno nadzorno ploščo.

V nadaljevanju poglavja je za vsak primer uporabe, ki je prikazan na diagramu primera uporabe (slika 4.2), predstavljen kratek opis zahtev. Kot komentar k diagramu naj dodam, da "SQL baza" predstavlja lokalno bazo na mobilni napravi, siva obroba pa označuje meje mobilne nadzorne plošče.

Prijava

Ob zagonu aplikacije oz. poteku seje se uporabniku odpre prijavna stran, ki mora izpolnjevati naslednje:

- uporabnik mora imeti možnost vnosa podatkov o uporabniškem imenu in geslu, ki so potrebni za nadaljnjo uporabo aplikacije,
- potrebno je preveriti veljavnost vnesenih podatkov uporabnika ter ga v primeru napake o tem ustrezno obvestiti. Uporabnika je potrebno obvestiti tudi o morebitnih drugih napakah,
- aktivnost za pravilno izvajanje potrebuje dostop do svetovnega spleta. V primeru, da uporabnik nima povezave, se ga o tem obvesti in tako se onemogoči nadaljnjo uporabo aplikacije.



Slika 4.2: Diagram primera uporabe

Seznam KPI knjižnic

Po uspešni prijavi se uporabniku odpre glavna stran, kjer so prikazani vsi KPI, ki jih spremlja uporabnik. Aktivnost mora izpolnjevati naslednje:

- predstavitev posameznega KPI je potrebno vizualno ločiti glede na njegovo aktivnost, vrsto in skupino, kateri pripada,
- ob kliku na posamezni KPI se uporabniku prikaže seznam akcij, ki jih lahko izvede. Število akcij je dinamično glede na izbrani KPI,
- uporabnik mora imeti v vsakem trenutku jasen pregled in dostop do podatka o trenutnem številu aktivnih alarmov. V primeru novega alarma se to prikaže tudi uporabniku.

Seznam alarmov

Uporabnik ima za izbrani KPI tipa alarm možnost pregleda vseh alarmov, ki so bili posredovani mobilni napravi. Podatke o alarmih se uporabniku predstavi v obliki seznama. Aktivnost mora izpolnjevati naslednje:

- v seznamu se prikaže aktivne in neaktivne alarme, ki so urejeni po datumu sprožitve. Podatki o alarmih so zapisani v lokalni podatkovni bazi,
- poleg vsebinskih podatkov, ki so lahko dinamične dolžine, se mora uporabniku vizualno predstaviti tudi metapodatke, kot sta nivo alarma in čas sprožitve,
- uporabnik mora imeti možnost potrditve posameznega alarma, s čimer sporoči, da je bil o alarmu obveščen in ga želi umakniti s seznama aktivnih alarmov.

Prikaz alarmov na zemljevidu

Uporabnik ima za določene KPI tipa alarm možnost prikaza alarmov na zemljevidu. Pogoj je, da podatki o alarmu hranijo vrednost o zemljepisni širini in dolžini. Aktivnost mora izpolnjevati naslednje:

- na zemljevidu se prikaže aktivne in neaktivne alarme, ki se jih prikaže z oznakami, ki se vizualno razlikujejo glede na vrsto oz. nivo alarma. Podatke o alarmih aplikacija bere iz lokalne podatkovne baze,
- ob kliku na oznako se uporabniku poleg vsebinskih podatkov, ki so lahko dinamične dolžine vizualno predstaviti tudi metapodatke, kot sta nivo alarma in čas sprožitve,
- uporabnik mora imeti možnost potrditve posameznega alarma, s čimer sporoči, da je bil o alarmu obveščen in ga želi umakniti s seznama aktivnih alarmov,

- uporabnik ima možnost filtriranja alarmov glede na njihovo aktivnost in nivo.

Tortni graf za izbrani KPI

Za KPI tipa tortni graf ima uporabnik možnost spremljanja podatkov v realnem času. Podatki se uporabniku predstavijo na tortnem grafikonu s pripadajočo legendo. Zahteva se naslednje:

- legenda mora vsebovati številske vrednosti s pripadajočimi enotami, kot tudi deleže vizualiziranih podatkov,
- uporabnik ima možnost nastavljanja frekvence osveževanja podatkov, ki jih aplikacija dobi preko RESTful spletne storitve.

Črtni graf za izbrani KPI

Za KPI tipa črtni graf ima uporabnik možnost spremljanja podatkov v realnem času. Podatki se uporabniku predstavijo na črtnem grafikonu. Legenda v tem primeru ni potrebna, saj je podatek, ki ga grafikon prikazuje razviden iz naslova. Zahteva se naslednje:

- uporabnik ima možnost nastavljanja frekvence osveževanja podatkov, ki jih aplikacija dobi preko RESTful spletne storitve.

Dobi obvestila o alarmu

V sklopu aplikacije je potrebno razviti tudi storitev za poizvedovanje o novih alarmih. Zahteva se naslednje:

- uporabnik lahko upravlja s storitvijo preko nastavitev znotraj aplikacije. Imeti mora možnost vklopa in izklopa storitve, nastaviti pogostost poizvedovanja o novih alarmih ter način obveščanja o novem alarmu,

- storitev mora teči v ozadju tudi takrat, kadar uporabnik ne uporablja aplikacije. Njen namen je periodično poizvedovanje, če se je sprožil kakšen nov alarm in obveščanje uporabnika,

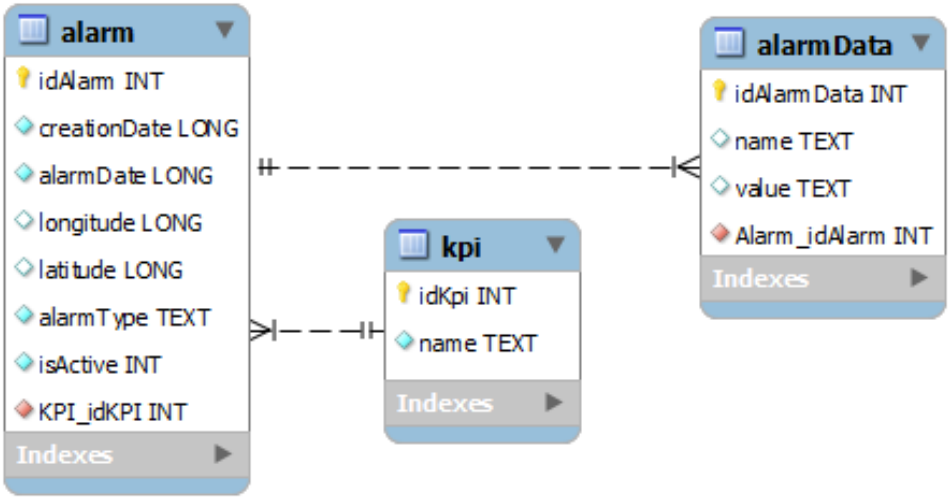
- do podatkov o novih alarmih storitev dostopa preko vmesnika RESTful spletne storitve. Dobljene podatke mora zapisati v lokalno podatkovno bazo, s čimer želimo omogočiti neodvisno delovanje storitve in pregled zgodovine alarmov znotraj aplikacije,

- za pravilno delovanje je potrebna povezava s svetovnim spletom, za kar je potrebna ustrezna obravnava znotraj storitve. Potrebno je zagotoviti, da se ne izgubi noben alarm.

4.1.3 Zasnova podatkovne baze

Konceptualni model zasnove podatkovne baze je predstavljen na sliki 4.3 in obsega 3 entitetne tipe. Kot je razvidno s slike, imamo opravka z relativno preprostim podatkovnim modelom, katerega glavna naloga je hranjenje podatkov o alarmih. Glavni razlog, da smo se odločili za hranjenje podatkov o alarmih lokalno, na mobilni napravi, je posledica funkcionalnosti, ki uporabniku omogoča, da je o alarmu obveščen tudi takrat, ko aplikacije ne uporablja. Celotna stvar je tako realizirana preko storitve, ki teče v ozadju in periodično preverja, ali se je morda sprožil kakšen nov alarm. Dobljene podatke o alarmih storitev nato zapiše v lokalno podatkovno bazo in uporabnika, preko obvestila v statusni vrstici (angl. “Status bar notification”), obvesti o novem alarmu. Zaradi hranjenja podatkov o alarmih ima uporabnik tudi možnost pregleda zgodovine.

V nadaljevanju poglavja so bolj podrobno predstavljeni posamezni entitetni tipi oz. tabele. Imena entitetnih tipov so zapisana v narekovajih, imena atributov pa v oglatih oklepajih.



Slika 4.3: Konceptualni model podatkovne baze

Alarm

Tabela "alarm" hrani metapodatke o posameznem alarmu. Za vsak alarm tako hranimo podatke o enoličnem identifikatorju [idAlarm], identifikator za KPI kateremu alarm pripada [KPI_idKPI], datum in čas zapisa alarma v podatkovno bazo [creationDate], datum in čas, ko se je alarm sprožil [alarmDate], tip oz. nivo alarma [alarmType], podatek o tem, ali je alarm še vedno aktiven oz. ali je bil uporabnik o njem že obveščen [isActive] ter podatki o lokaciji, zemljepisna širina [latitude] in dolžina [longitude], kjer se je alarm sprožil. V primeru, da podatek o lokaciji ni na voljo, se zapiše vrednost "null".

AlarmData

Tabela "alarmData" hrani vsebinske podatke o alarmu. Na zapis v tabeli "alarm" je lahko vezanih nič ali več zapisov v tabeli "alarmData". Vsak alarm ima lahko poljubno količino podatkov, ki so predstavljeni kot dvojice naziv–vrednost. Posamezen zapis v tabeli je tako sestavljen iz enoličnega identifika-

torja alarma [Alarm_idAlarm], enoličnega identifikatorja za podatek [idAlarmData] ter naziva podatka [name] in njegove vrednosti [value].

Kpi

Tabela "kpi" hrani podatke o ključnih zmogljivostnih indikatorjih (angl. "Key Performance Indicator - KPI") opazovanega sistema, katere spremlja uporabnik. Vsakič, ko storitev dobi podatke o novem alarmu, se pred zapisom podatkov v tabelo "alarm" preveri, ali v tabeli "kpi" že obstaja zapis za KPI novega alarma. V primeru, da ne obstaja, se v tabelo zapiše podatke o enoličnem identifikatorju [idKpi] in nazivu [name] novega KPI.

4.2 Razvoj in uporaba aplikacije

Sedaj, ko smo se seznanili s tehnično zasnovo aplikacije, si bomo podrobneje pogledali zanimivosti pri samem razvoju in predstavili uporabo aplikacije. Podrobneje si bomo pogledali zanimivosti pri delu s podatkovno bazo, razvoj storitve za poizvedovanje o novih alarmih in o njihovem obveščanju uporabnika. Pogledali si bomo tudi težave in pristop, ki je bil potreben, da razvita aplikacija lahko deluje na zmogljivostno in fizično različnih napravah ter posebnosti s katerimi smo se srečali pri pridobivanju podatkov preko RESTful spletne storitve. Za konec so posamezno predstavljene tudi vse aktivnosti, ki nastopajo v končni aplikaciji. Aktivnost si najlažje predstavljamo tako, da jo enačimo z okni pri razvoju namiznih aplikacij. Za vsako aktivnost so na kratko opisane posebnosti s katerimi smo se srečali pri razvoju in njena uporaba.

4.2.1 Delo s podatkovno bazo

Za upravljanje relacijske podatkovne baze na mobilni napravi smo uporabili SQLite. Android podatkovne baze shranjuje v mapo */data/data/[packagename] /databases* ter privzeto za vsako podatkovno bazo velja, da je privatna in do-

stopna samo aplikaciji, ki jo je ustvarila. Ker je SQLite implementiran kot knjižnica in ne kot ločen proces, je vsaka SQLite baza integrirana v aplikacijo, ki jo je ustvarila.

Večina mobilnih naprav je pomnilniško in diskovno zelo omejenih, zato je posebno pozornost potrebno nameniti nadzoru prostora, ki ga zavzame baza. Pri načrtovanju podatkovne baze je potrebno izvesti postopek normalizacije, obenem pa je priporočljivo, da se v naprej določi obseg podatkov, ki jih je smiselno hraniti.

Pri interakciji s podatkovno bazo znotraj aplikacije velja kot dobra praksa to, da se ustvari poseben razred, ki nastopa v vlogi vmesnika med podatkovno bazo in aplikacijo ter skrbi za vso potrebno komunikacijo med njima. Podatkovni vmesnik vsebuje vse metode, ki so potrebne za dodajanje, brisanje in posodabljanje zapisov v bazi. Prav tako mora imeti podporo za obravnavanje raznih povpraševanj in metode za kreiranje, odpiranje in zapiranje podatkovne baze.

V našem primeru za to skrbi razred *DBAdapter*, znotraj katerega smo definirali privatni razred *AlarmDBOpenHelper*, ki razširja abstraktni razred *SQLiteOpenHelper* in skrije logiko, ki je potrebna za nadgradnjo že obstoječe baze oz. njeno ustvaritev. Kot zanimivost si poglejmo realizacijo dostopa do baze, kar prikazuje tudi spodnja koda. Za odpiranje baze se pokliče metodo *getReadableDatabase* ali *getWritableDatabase*. Kot že samo ime pove, je prva metoda namenjena bralnemu, druga pa bralno-pisalnemu dostopu do baze. Pri slednjem dostopu je potrebno upoštevati, da se v primeru pomanjkanja prostora na disku ali neustreznih pravic sproži izjema, zato je dobra praksa, da se ob izjemi izvede metodo *getReadableDatabase*, s čimer se omogoči vsaj bralni dostop in se s tem omogoči nadaljnje izvajanje aplikacije.

```
private SQLiteDatabase mDB;
private alarmDBOpenHelper dbHelper;

public DBAdapter(Context _context) {
        dbHelper = new alarmDBOpenHelper(_context, DATABASE_NAME,
```

```
                null, DATABASE_VERSION);
}
public void open() throws SQLiteException {
        try {
                mDB = dbHelper.getWritableDatabase();
        } catch(SQLiteException ex) {
                mDB = dbHelper.getReadableDatabase();
        }
}
```

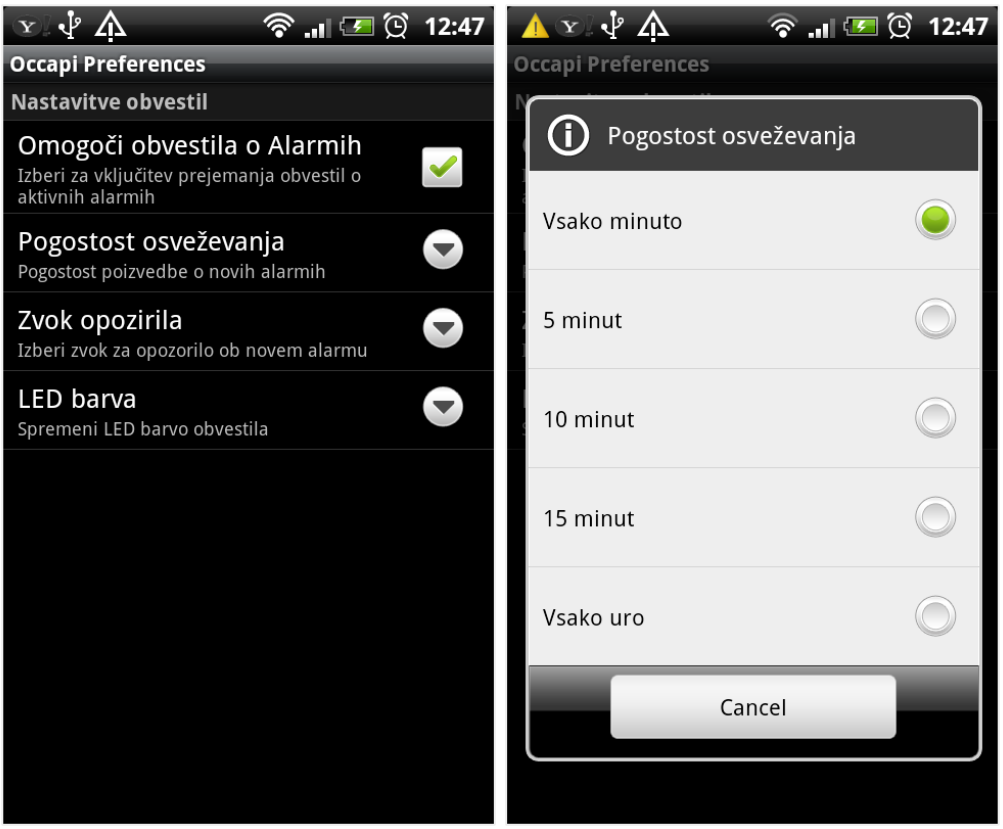
4.2.2 Storitev in obvestila za alarme

Ena izmed pomembnejših zahtev je, da mora biti uporabnik o novem alarmu obveščen v vsakem trenutku tudi takrat, ko aplikacije ne uporablja. S tem se uporabniku omogoči, da je vedno obveščen o izjemnem dogodku v sistemu in lahko hitro sprejema ustrezne odločitve in ukrepe. Ta funkcionalnost predstavlja enega izmed pomembnejših doprinosov nadzorne plošče za mobilne naprave v primerjavi s spletnim vmesnikom.

V ta namen smo v sklopu aplikacije razvili storitev, ki teče v ozadju ter periodično preverja ali se je sprožil kakšen nov alarm in o tem obvesti uporabnika. Delovanje storitve je možno upravljati preko nastavitev v aplikaciji, kar prikazuje slika 4.4. Za naše potrebe smo ustvarili razred *KPIAlarm_Service*, ki razširja razred *Service* in vsebuje vse metode, potrebne za upravljanje in nadzor storitve. Iz razreda *Service* smo ponovno definirali metode *onCreate*, *onBind* in *onStartCommand* ter jih prilagodili našim potrebam.

```
public class KPIAlarm_Service extends Service {
    @Override
    public void onCreate() {
        //operacije, ki se izvedejo ob kreiranju storitve

        V našem primeru odpremo podatkovno bazo, pripravimo vse potrebno za
        generiranje obvestil o alarmih. Uredimo dostop do nastavitev in
        konteksta aplikacije.
```



Slika 4.4: Nastavitve storitve, ki jih lahko določa uporabnik

```
}
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    //operacije, ki se izvedejo pred začetkom izvajanja storitve.

    V našem primeru preberemo nastavitve, ki jih je določil uporabnik
    in pričnemo s periodičnim poizvedovanjem o novih alarmih.

    return Service.START_NOT_STICKY;
}
@Override
public IBinder onBind(Intent intent) {
```

```
        return null;
    }
}
```

Za pravilno delovanje in dodelitev pravic, za izvajanje prej omenjene storitve, je potrebno v manifest datoteki dodati naslednjo vrstico:

```
<service android:enabled="true" android:name=".services.KPIAlarm_Service"/>
```

Slika 4.5: Obvestilo o novem alarmu

Kadar uporabnik aplikacije ne uporablja, se ga v primeru novega alarma o njem obvesti preko obvestila v statusni vrstici (slika 4.5) ter zvočnih in svetlobnih opozoril, ki jih je določil v nastavitvah storitve. Ob kliku na obvestilo

se odpre seznam vseh aktivnih alarmov, ki so urejeni padajoče glede na datum sprožitve. Med uporabo aplikacije takšen način obveščanja ni najbolj primeren, saj želimo, da je uporabnik o alarmu obveščen znotraj aplikacije same. V ta namen smo ustvarili razred *KPIAlarm_AlarmReceiver*, ki razširja razred *BroadcastReceiver*. Njegova glavna naloga je, da ob novem alarmu o njem obvesti vse, ki jih to zanima. Za pravilno delovanje je med drugim potrebno dodati naslednjo kodo v manifest datoteko.

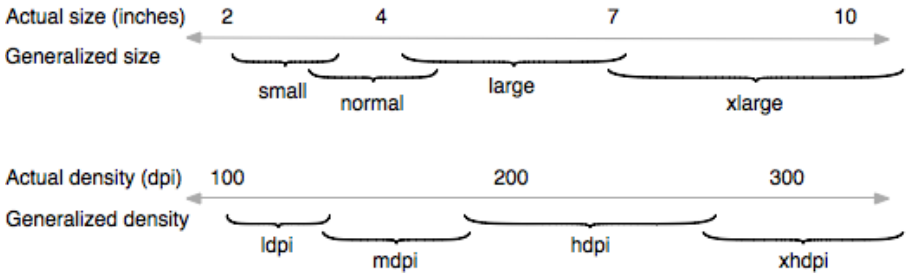
```
<receiver android:name=".services.KPIAlarm_AlarmReceiver">
    <intent-filter>
        <action android:name="si.occapi.services.ACTION_REFRESH_KPI_ALARM"/>
    </intent-filter>
</receiver>
```

4.2.3 Podpora različnih naprav

Android je trenutno najbolj razširjen operacijski sistem med pametnimi telefoni, kar mu omogoča predvsem možnost namestitve na zmogljivostno in fizično različne naprave. Pri razvoju smo se odločili podpreti vse naprave, ki imajo nameščen Android od verzije 2.1 do verzije 3.x, ki je v večini primerov nameščena na tablicah. Po podatkih iz tabele 4.1 smo tako pokrili 94,4 % vseh naprav, ki so prisotne na trgu.

Zaradi podpore različnih naprav je bilo potrebno veliko časa posvetiti načrtovanju in oblikovanju uporabniškega vmesnika. Naš cilj je bil izdelati preprosto in uporabniku intuitivno aplikacijo. Pri pripravi dizajna smo se držali minimalističnih načel, v ospredje pa smo postavili vsebino, ki je za uporabnika ključnega pomena. Pri oblikovanju je zelo pomembno upoštevati to, da se zasloni naprav med seboj razlikujejo tako v velikosti, kot tudi v resoluciji ter posledično gostoti pik na enoto. Različne velikosti in gostote, ki nastopajo pri Android napravah, so predstavljene na sliki 4.6. Celoten dizajn smo pripravili s programi Adobe in kasneje opravili razrez za potrebe naše aplikacije. Slike, stili, postavitve elementov ipd. so zbrane v mapi *res/*. Končni dizajn in

uporabniški vmesnik aplikacije sta predstavljena v nadaljevanju pri predstavitvi aktivnosti.



Slika 4.6: Dejanska velikost in gostota zaslona Android naprav in njihove posplošene vrednosti

Kot smo že omenili, Android omogoča enoten pristop pri razvoju aplikacij, kar razvijalcem omogoča preprost razvoj za različne naprave. V nadaljevanju je predstavljenih nekaj stvari, na katere je potrebno biti pozoren pri podpori različnih naprav:

- V manifest datoteki je potrebno definirati katere velikosti zaslonov naša aplikacija podpira. To storimo z vključitvijo elementa *supports-screens* v manifest datoteko. Spodaj je primer iz manifesta aplikacije, ki podpira *small*, *medium* in *large* tipe zaslonov.

```
<supports-screens
        android:smallScreens="true"
        android:normalScreens="true"
        android:largeScreens="true" />
```

- Potrebno je zagotoviti različne postavitve elementov, za različne velikosti zaslonov. V večini primerov to ni potrebno, saj Android, če seveda razvijalec to ustrezno upošteva, pri definiranju postavitve elementov na zaslonu sam prilagodi velikost aplikacije velikosti zaslona. V primerih,

ko to ne zadostuje in so potrebne večje spremembe, to storimo z definiranjem postavitve elementov za posamezno velikost zaslona. V primeru velikih zaslonov bi tako xml datoteke, ki opisujejo postavitev elementov, shranili v mapo *layout-large/* znotraj mape, z resursi *res/*. Za ostale velikosti zaslonov bi datoteke, ki določajo postavitev elementov, shranili v mape, kjer bi namesto *large* uporabili *small*, *normal*, ali *xlarge* in sicer glede na velikost zaslona za katerega je prirejena postavitev.

- Za različne gostote zaslona je potrebno zagotoviti različne bitne slike (angl. "bitmap drawables"). Privzeto Android uporabi slike, ki so mu na voljo in jih ustrezno poveča tako, da ustrezajo gostoti trenutnega zaslona. Za najboljši rezultat lahko uporabimo različne slike za različne gostote zaslonov. Tako bi, na primer, morali za zaslone z visoko gostoto slike shraniti v mapi *drawable-hdpi/* znotraj mape z resursi *res/*. Za ostale gostote zaslonov bi slike shranili v mape, kjer bi namesto *hdpi* uporabili *ldpi*, *mdpi* ali *xhdpi* in sicer glede na gostoto zaslona, za katerega so prilagojene slike.

- Pri definiranju postavitve elementov in določanju njihovih velikosti so na voljo različne enote.

  - **dp** predstavlja velikost elementa v pikah (angl. "pixels"), katerih število se prilagaja glede na gostoto zaslona in njegovo velikost.
  - **sp** predstavlja velikost elementa v pikah, katerih število je odvisno od gostote zaslona in uporabniških nastavitev glede velikosti pisave. Uporaba te enote je še posebej primerna pri določanju velikosti pisave.

4.2.4 RESTful spletna storitev

Večina aktivnosti za svoje pravilno delovanje od platforme Occapi zahteva ustrezne podatke. Do podatkov aplikacija dostopa preko v ta namen razvi-

tega vmesnika, ki je pripravljen v skladu z načeli REST arhitekture. Posebno pozornost je potrebno posvetiti varni povezavi med strežnikom in aplikacijo ter ohranjanju seje tekom uporabe aplikacije. Glede nato, da REST arhitektura ne predvideva ohranjanja stanja povezave, smo sejo oblikovali na osnovi piškotkov, kjer strežnik za vsako sejo pripravi nov unikaten ključ. V času izvajanja aplikacije podatek o ključu hranimo kot globalno spremenljivko znotraj razreda *OccapiApplication*, ki razširja razred *Application* in je dostopen vsem aktivnostim. V primeru potekle seje se sproži izjema in se uporabnika preusmeri na prijavno stran.

Android zahteva, da se dolgotrajne operacije, kot so komunikacija in prenos podatkov preko interneta, izvajajo v posebni niti, ki je ločena od niti uporabniškega vmesnika. S tem se doseže večja odzivnost aplikacije, ki je zelo pomembna z vidika dobre uporabniške izkušnje. Spodaj je del kode, ki skrbi za povezavo s spletno storitvijo in prejem podatkov. Ko iz JSON niza dobimo želene podatke, izvajanje nadaljujemo v niti uporabniškega vmesnika, saj je le v tej niti možno spreminjati elemente oziroma vrednosti elementov na zaslonu.

```
String jsonString = "";
try {
    URL url = new URL('URL naslov do resursa spletne storitve');

    BufferedReader in = new BufferedReader(
        new InputStreamReader(url.openStream()));
    String str;
    while ((str = in.readLine()) != null) {
        jsonString = jsonString + str.trim();
    }
    in.close();

    ... iz JSON niza dobimo podatke, ki nas zanimajo.

    runOnUiThread(setResults);
} catch (MalformedURLException e) {
} catch (IOException e) {
```

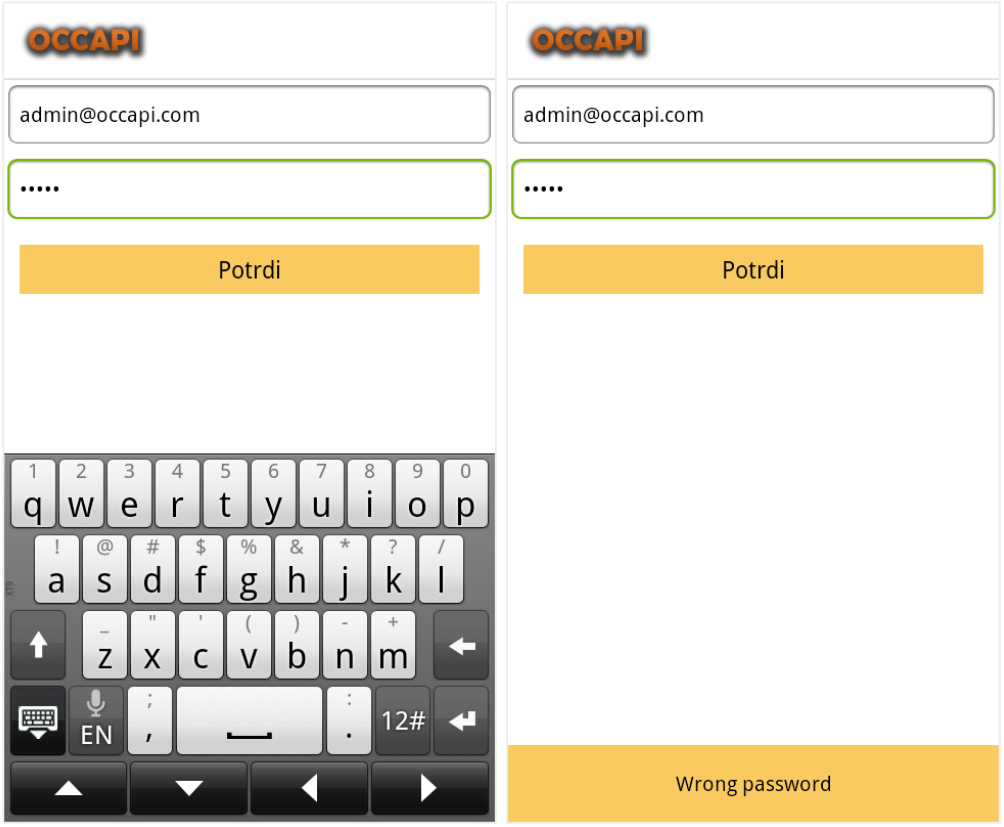
```
}
```

Spletna storitev za pravilno delovanje potrebuje pravice za uporabo interneta, kar omogočimo tako, da dodamo naslednjo vrstico v manifest datoteko.

```
<uses-permission android:name="android.permission.INTERNET" />
```

4.2.5 Predstavitev aktivnosti in uporabe aplikacije

Prijava



Slika 4.7: Zasloni aktivnost Prijava

Uporabniku se ob zagonu aplikacije prikaže aktivnost *Prijava* (razred *Login.java*). Gre za relativno preprosto aktivnost, kjer se od uporabnika zahteva,

da vnese podatke o svojem uporabniškem imenu in geslu. V primeru napačno vnesenih podatkov oziroma neke druge napake, ki se je zgodila na strežniku, se uporabniku prikaže obvestilo z opisom napake. Ob začetku izvajanja aktivnosti se tudi preveri, ali ima uporabnik dostop do svetovnega spleta, saj je to pogoj za uporabo aplikacije. V primeru, da ga nima, se uporabniku prikaže obvestilo. Identifikacija uporabnika poteka preko spletne storitve, kjer strežniku pošljemo podatke o uporabniku, strežnik pa potrdi njihovo ustreznost in s tem omogoči nadaljnjo uporabo aplikacije.

Slika 4.7 prikazuje posnetke zaslonov aktivnosti *Prijava*.

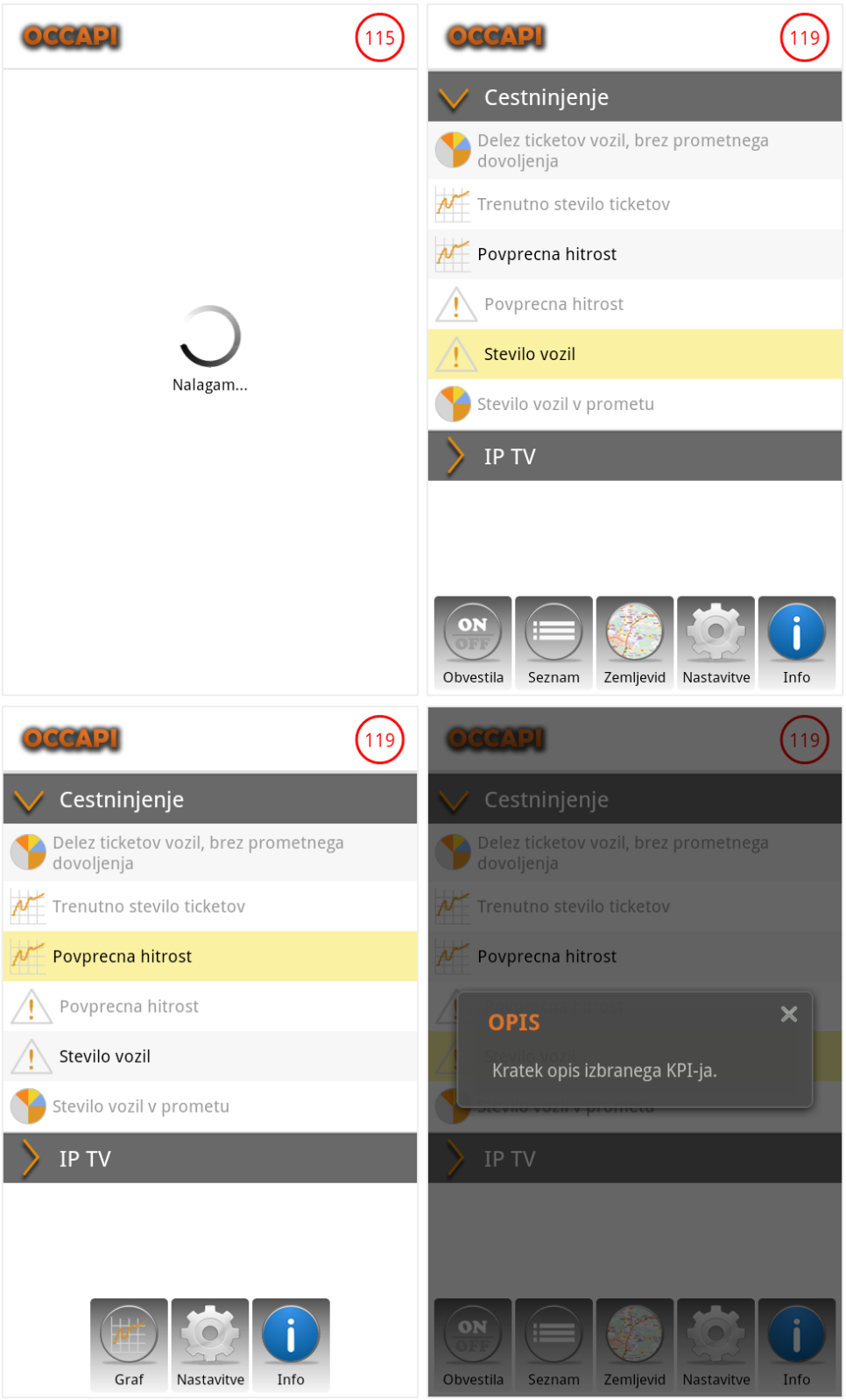
KPI knjižnice

Po uspešni prijavi se uporabniku odpre aktivnost *KPI knjižnice* (razred *KPILibrariesList.java*), ki predstavlja glavno aktivnost aplikacije. Tukaj so uporabniku na pregled vsi KPI, ki jih spremlja. Ob kliku na posamezni KPI se uporabniku spodaj prikažejo možne akcije, ki jih lahko izvede nad tem KPI. S klikom na akcijo se uporabnika preusmeri na ustrezno aktivnost. KPI so prikazani v dvonivojskem seznamu, (*ExpandableListView*) v katerem so razdeljeni po skupini, kateri pripadajo. Za delo s seznamom skrbi vmesnik *ExpandableKPIGroupsListAdapter*, ki razširja *BaseExpandableListAdapter* in smo ga prilagodili za naše potrebe. Podatke o KPI dobimo preko spletne storitve, zato je potreben dostop do interneta. O prenosu podatkov se uporabnika obvesti z ustreznim obvestilom.

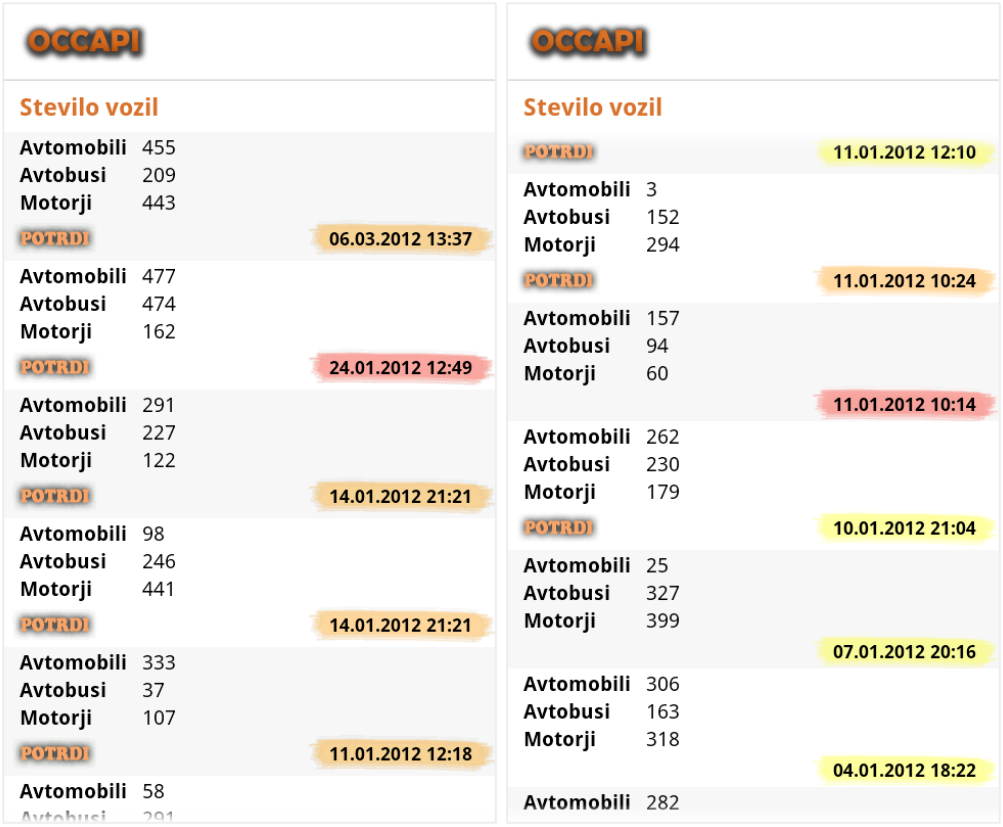
Slika 4.8 prikazuje zaslone aktivnosti.

Seznam alarmov

Poseben tip KPI predstavljajo alarmi. Aktivnost *Seznam alarmov* (razred *AlarmList.java*) omogoča pregled seznama alarmov urejenih po datumu nastanka. Posamezni element iz seznama hrani podatke o nastanku in nivoju alarma, vsebinske podatke o alarmu ter status, ali je uporabnik alarm že prebral. Uporabnik ima tudi možnost potrjevanja alarmov, s čimer pove, da je



Slika 4.8: Zasloni aktivnosti KPI knjižnice

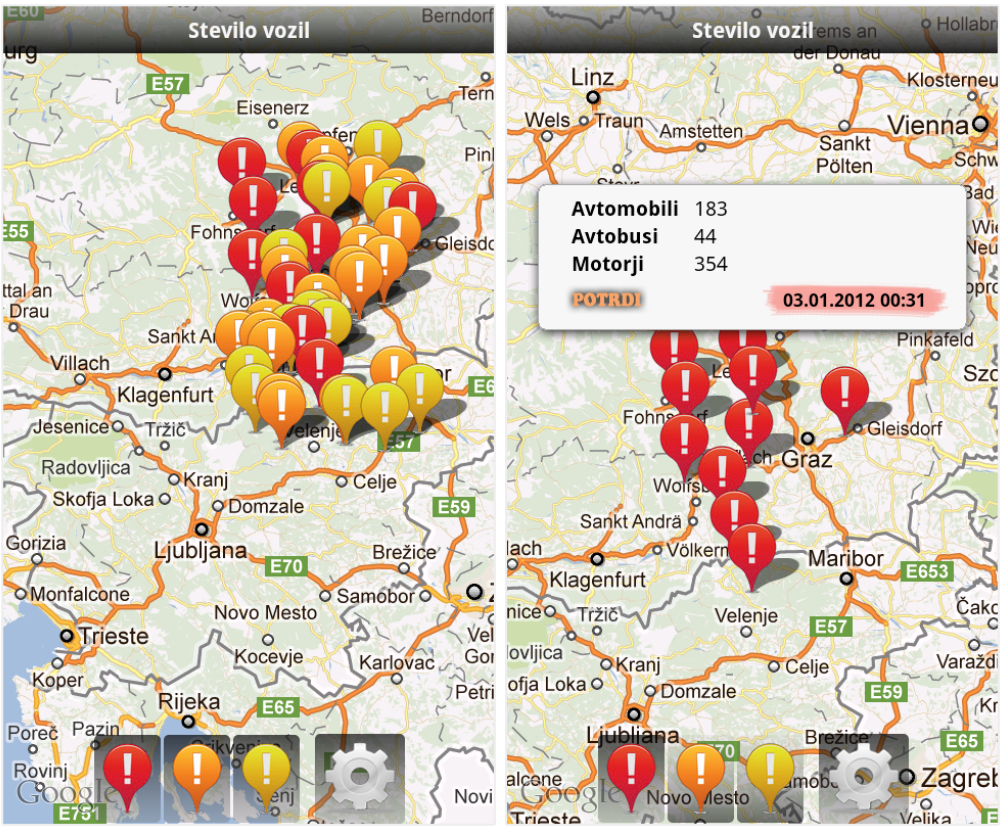


Slika 4.9: Zasloni aktivnosti Seznam alarmov

bil o alarmu obveščen in ga tako umakne s seznama aktivnih alarmov. Za delo s seznamom skrbi vmesnik *AlarmAdapter*, ki razširja razred *ArrayAdapter*. Podatke, ki se jih prikaže v seznamu, se dobi iz lokalne podatkovne baze, s katero se upravlja preko razreda *DBAdapter*. Na sliki 4.9 so prikazani posnetki zaslonov aktivnosti.

Prikaz alarmov na zemljevidu

Prikaz KPI tipa alarm je možen tudi na zemljevidu. To seveda velja samo za alarme, ki hranijo podatke o zemljepisni širini in dolžini. Kot pri seznamu se tudi v tem primeru podatki berejo iz lokalne podatkovne baze. Aktivnost



Slika 4.10: Zasloni aktivnosti Prikaz alarmov na zemljevidu

Prikaz alarmov na zemljevidu (razred *AlarmMap.java*) razširja razred *MapActivity*, ki poskrbi za vse, kar je potrebno za vključitev Google zemljevida v aplikacijo. Na zemljevidu se z ustreznimi oznakami, ki so različne glede na nivo alarma, prikaže vse alarme, ki ustrezajo filtru, ki ga je določil uporabnik. Privzeto se prikažejo vsi alarmi. Poleg tega je za pravilno delovanje potrebno v manifest datoteki dodati knjižnico (kot otrok elementa *application*) in pravice za dostop do interneta. To je prikazano v spodnji kodi.

```
<uses-permission android:name="android.permission.INTERNET" />
<application ... >
    <uses-library android:name="com.google.android.maps" />
```

```
    ...
</application>
```

Poleg tega je potrebno v datoteko, ki opisuje postavitev elementov vključiti *MapView* element, ki je namenjen prikazu zemljevida.

```
<si.occapi.widget.OccapiMapView
    android:id="@+id/mapview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="[API KEY]"/>
```

V našem primeru smo vključili element *OccapiMapView*, ki razširja razred *MapView* in smo ga prilagodili našim potrebam. Prav tako lahko iz kode razberemo, da je potreben API ključ, ki ga dobimo z registracijo certifikata, s katerim je podpisana naša aplikacija, na Google Maps[1] strani. Pri registraciji se zahteva MD5 prstni odtis certifikata.

Del razreda *AlarmMap* predstavlja tudi privatni razred *MapsAlarmMarker*, ki razširja razred *ItemizedOverlay* in skrbi za prikaz oznak na zemljevidu ter definiranje ustreznih akcij ob kliku le-teh.

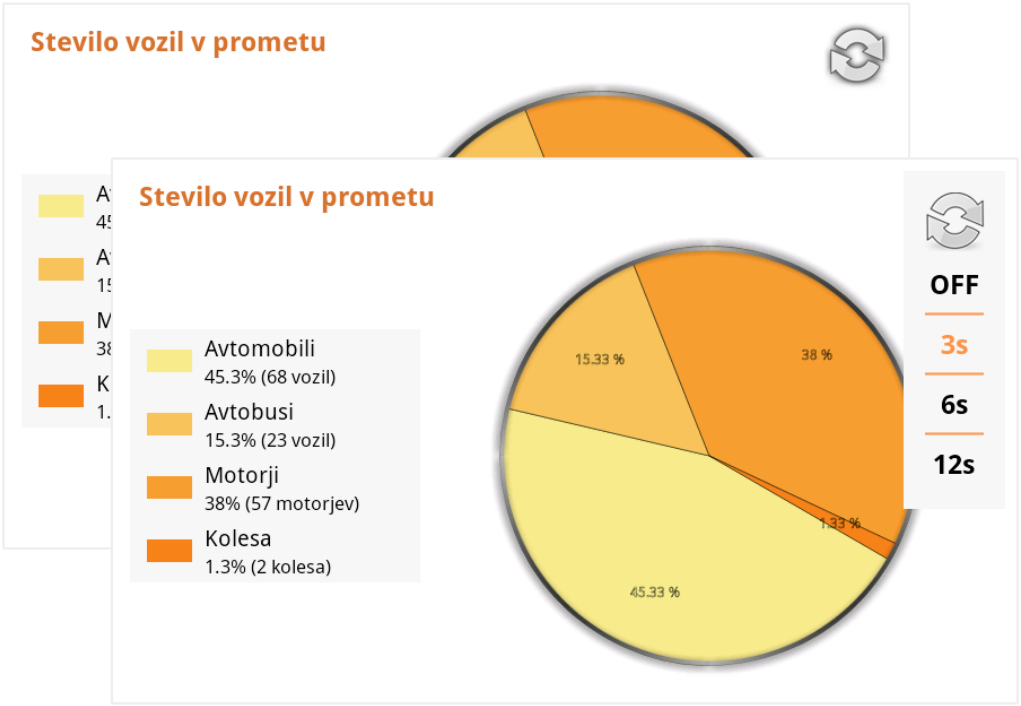
Ob kliku na neko oznako se prikaže pojavno okno, ki hrani podrobnejše podatke o alarmu. Za prikaz in definiranje tega okna skrbi razred *PopupPanel*. Vsebina pojavnega okna je enaka posameznemu elementu iz prikaza alarmov v seznamu.

Slika 4.10 prikazuje zaslone aktivnosti.

Tortni graf

Aktivnost *Tortni graf* (razred *DisplayGraph.java*) je eden od načinov prikaza KPI podatkov. Prikazani podatki odražajo trenutno stanje v sistemu in se periodično osvežujejo. Za izris tortnega grafikona skrbi razred *PieChartView*, znotraj katerega so definirana pravila in metode, potrebne za ustrezen izris grafa.

[1] http://code.google.com/android/add-ons/google-apis/maps-api-signup.html



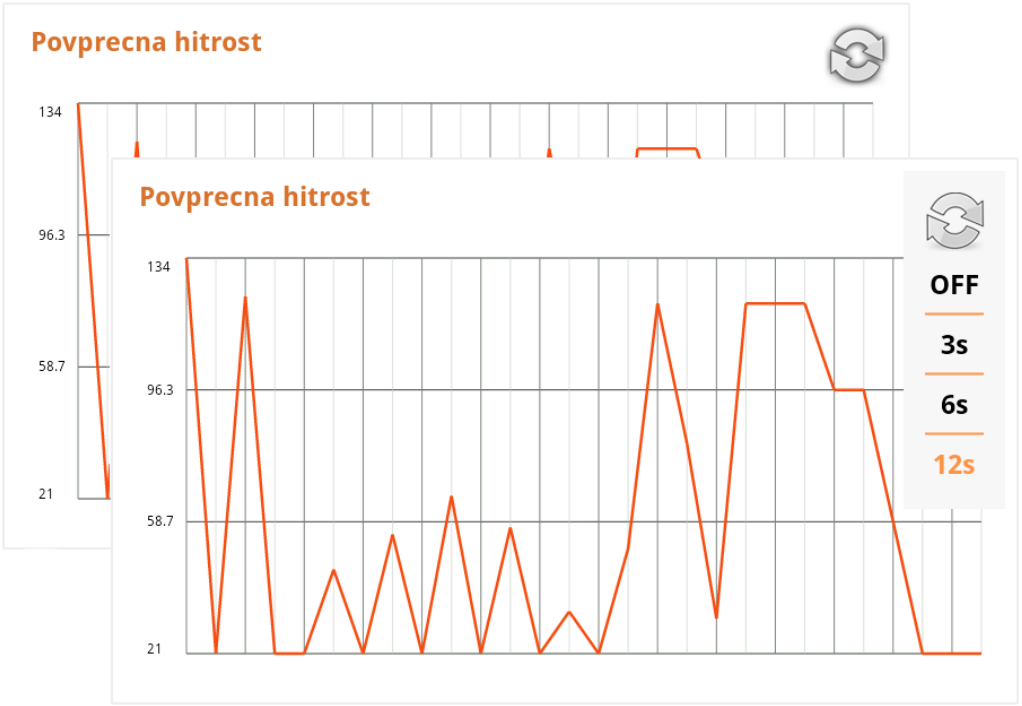
Slika 4.11: Zasloni aktivnosti Tortni graf

V legendi se uporabniku poleg vrednosti v odstotkih prikaže tudi številčna vrednost z ustreznimi enotami. Podatke aktivnost pridobi preko spletne storitve, zato je za pravilno delovanje potrebna internetna povezava. Med opazovanjem podatkov na tortnem grafu lahko uporabnik nastavi frekvenco s katero želi, da se podatki osvežujejo. Za to skrbi razred *RefreshRateCallout*, ki razširja razred *PopupWindows* in katerega glavna naloga je prikaz ustreznega pojavnega okna in posodobitev nastavitev ob novo izbrani vrednosti.

Slika 4.11 prikazuje zaslone aktivnosti.

Črtni graf

Aktivnost *Črtni graf* (razred *DisplayGraph.java*) je eden od načinov prikaza KPI podatkov. Prikazani podatki odražajo trenutno stanje v sistemu in se



Slika 4.12: Zasloni aktivnosti Črtni graf

periodično osvežujejo. Za izris črtnega grafikona skrbi razred *LineChartView*, znotraj katerega so definirana pravila in metode, potrebne za ustrezen izris grafa. Podatke aktivnost pridobi preko spletne storitve, zato je za pravilno delovanje potrebna internetna povezava. Med opazovanjem podatkov na črtnem grafu lahko uporabnik nastavi frekvenco s katero želi, da se podatki osvežujejo. Za to skrbi razred *RefreshRateCallout*, ki razširja razred *PopupWindows* in katerega glavna naloga je prikaz ustreznega pojavnega okna in posodobitev nastavitev ob novo izbrani vrednosti.

Slika 4.12 prikazuje zaslone aktivnosti.

Poglavje 5

Sklep

Nadzorna plošča za platformo Occapi, ki smo jo razvili v sklopu diplomske naloge, uporabnikom omogoča spremljanje dogajanja v vsakem trenutku. Glavna prednost oz. doprinos razvite rešitve je obveščanje uporabnikov o pomembnih dogodkih direktno na njihov pametni telefon tudi takrat, ko same aplikacije ne uporabljajo. Poleg tega je možen pregled zgodovine obvestil in spremljanje trenutnega dogajanja. S tem se uporabniku omogoči hitro in učinkovito odzivanje na nepredvidene situacije, v katerih se lahko znajde sistem in sprejemanje ustreznih ukrepov.

Za enkrat smo podprli samo mobilne naprave, ki jih poganja operacijski sistem Android, v nadaljevanju pa bi bilo potrebno ustrezno rešitev razviti tudi za ostale mobilne platforme. Veliko prostora je tudi na področju funkcionalnih izboljšav, kjer bi lahko omogočili različne funkcionalnosti glede na tip uporabnika, ki se prijavi v aplikacijo. Tako bi imeli uredniki večji nadzor in možnost osnovnega upravljanja s sistemom. V primeru, da se omogoči upravljanje s sistemom, je potrebno še dodatno posvetiti pozornost varni ter zanesljivi komunikaciji med platformo Occapi in aplikacijo. Zanimivo bi bilo tudi uvesti večje število različnih načinov prikazovanja podatkov posameznega KPI in preklapljanje med posameznimi pogledi, kjer je to seveda mogoče. V nekaterih primerih bi bilo smiselno uvesti lokacijsko obveščanje o alarmih, kjer

bi bil uporabnik obveščen samo o alarmih, ki so se zgodili v njegovi bližini. To bi bilo primerno predvsem za večje, geografsko porazdeljene sisteme.

Slike

Literatura

[1] A. Yochem, H. Taylor, L. Phillips, F. Martinez, "Event-driven architecture: how SOA enables the real-time enterprise", Addison-Wesley, 2009.

[2] O. Etzion, P. Niblett, D. Luckham, "Event Processing in Action", Manning Pubs Co Series, Manning Publications, 2010.

[3] B. M. Michelson, "Event-driven architecture overview", *Patricia Seybold Group*, 2006.

[4] J. van Hoof, "How EDA extends SOA and why it is important", 2006.

[5] K. Mani Chady, "What is Different About Event Driven Architecture?", *Gartner*, 2006.
Dostopno na: http://www.gartner.com/it/content/616700/616710/mani_chandy_research.pdf

[6] Wikipedia, "Event-Driven Architecture", 2012.
Dostopno na: http://en.wikipedia.org/wiki/Event-driven_architecture

[7] Wikipedia, "Event-Driven SOA", 2012.
Dostopno na: http://en.wikipedia.org/wiki/Event-driven_SOA

[8] A. Mar, "Event Driven Architecture Not Your Grandmother's SOA", Simplicable, 2011.
Dostopno na: http://simplicable.com/new/event-driven-architecture-not-your-grandmothers-soa

[9] Opcomm, 2012.
Dostopno na: http://www.opcomm.eu/

[10] M. Bajec, D. Lavbič, T. Vidonja, A. Kos, "OCCAPI - odprta inteligentna komunikacijska platforma", v zborniku: Informatika v javni upravi 2011, 2011, str. 1–9.

[11] iDigi, "Enabling the Internet of Things", 2012.
Dostopno na: http://www.idigi.com/the-internet-of-anything

[12] Axeda, "Axeda Application and Data Integration Platform", 2012.
Dostopno na: http://www.axeda.com/products/platform

[13] ioBridge, 2012.
Dostopno na: http://iobridge.com/technology/

[14] "Platform Versions", Android developers, 2012.
Dostopno na: http://developer.android.com/resources/dashboard/platform-versions.html