UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Aljaž Čukajne

KONTEKSTNI STREŽNIK NA MOBILNI NAPRAVI

MAGISTRSKO DELO

ŠTUDIJSKI PROGRAM DRUGE STOPNJE

RAČUNALNIŠTVO IN INFORMATIKA

Mentor: doc. dr. Rok Rupnik

Ljubljana, 2014

IZJAVA O AVTORSTVU MAGISTRSKEGA DELA

Spodaj podpisani Aljaž Čukajne, z vpisno številko 63070064, sem avtor magistrskega dela z naslovom:

Kontekstni strežnik na mobilni napravi

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom doc. dr. Roka Rupnika,
- so elektronska oblika magistrskega dela, naslov (slov., ang.), povzetek (slov., ang.) ter ključne besede (slov., ang.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 26.9.2014 Podpis avtorja: _____

Za pomoč in nasvete pri izdelavi diplomske naloge se zahvaljujem mentorju
doc. dr. Roku Rupniku.

Zahvaljujem se svojim staršem, ki so mi omogočili študij in mi vedno stali ob strani.

Zahvala gre tudi vsem sošolcem in prijateljem, za vso pomoč in moralno podporo.

KAZALO

POVZETEK

Da bi mobilne aplikacije lahko učinkovito nudile vse svoje storitev in jih prilagodile uporabniku, bi morale pri tem uporabiti ves kontekst uporabnika in njegove okolice. V obstoječih raziskavah to nalogo prevzema poseben kontekstni strežnik, ki skrbi za zbiranje podatkov ter posredovanje informacij aplikaciji. V primeru, da bi tak kontekstni strežnik deloval na mobilni napravi, bi bila postopoma odpravljena potreba po omenjenem sistemu. V sklopu naloge smo predlagani model kontekstnega strežnika uspešno implementirali in njegovo delovanje preizkusili s pomočjo kontekstno odvisne aplikacije, ki deluje na principu tekstovne klasifikacije uporabnikovih vnosov v spletnih iskalnikih in njegovi trenutni lokaciji. Aplikaciji omenjenih podatkov ni potrebno zajeti ročno, saj so ti ob pojavitvi novega podatka avtomatično posredovani preko kontekstnega strežnika.

Ključne besede

Kontekst, kontekstni strežnik, Android, Weka.

ABSTRACT

In order for mobile applications to efficiently offer all of their services and adjust them according to the end user, they would need to monitor all of his context and surroundings. In the existing researches this is done by a special context server which is responsible for data retrieval and transmission of information application. In the event, when the context server would work on a mobile device, the before mentioned system would gradually not be needed anymore. In the context of the thesis we have successfully implemented the proposed model of context server and tested it with a context application, which operates in the principle of text classification of user entries in the search engines and its current location. The application of this data does not need to be captured manually, since they are automatically transmitted to the server when the new data appears.

Keywords

Context, context server, Android, Weka.

1. UVOD

1.1. Motivacija

Informacije so v današnjem času postale temeljni dejavnik obstoja in razvoja različnih panog, ki so ključne za delovanje združb na različnih nivojih populacije. Pomembne so delovanje vladnih, izobraževalnih, sodnih in ostalih ustanov na nivoju držav, kot tudi podjetij, borz, bank in ostalih institucij. Zanemariti ne smemo tudi informacijskih potreb posameznih oseb. Informacije, ki jih potrebujemo so lahko poslovne ali zasebne narave. Obveščajo nas o dogodkih in novicah v svetu, trenutni gospodarsko situaciji na trgih, stanju bližnjih, s katerimi smo v stiku, naravnih razmerah v okolju in ostalih dejavnikih. Kljub temu, da v nekem trenutku nek podatek obravnavamo kot informacijo, pa lahko ta hitro izgubi svoj pomen, če ni dostavljena pravemu naslovniku v pravem času.

Zaradi teh razlogov je pridobivanje in procesiranje informacij ena od glavnih lastnosti slehernega računalniškega sistema v današnjem času. Na tem področju so se sčasoma razvijale nove tehnologije in metode, ene od njih so tudi tako imenovani kontekstni stroji (ang. *context engine*), ki jih lahko umestimo v večje celote, kontekstno odvisne sisteme. Ti sistemi so zgrajeni iz več delov, kjer ključno vlogo igra kontekstni strežnik (ang. *context server*).

Ni naključje, da so prve omembe kontekstnega procesiranja sovpadale z razvojem različnih tipov brezžičnih komunikacij in prenosnih naprav. V preteklosti so bile te naprave programsko in predvsem strojno zelo omejene, zato je procesiranje konteksta prevzel ločen kontekstni stroj. Sočasno z razvojem strežniških arhitektur se je tako pojavilo tudi nekaj novih sistemov za distribuirano pridobivanje konteksta, njegovo procesiranje, hranjenje ter distribucijo procesiranih informacij ciljnim uporabnikom ali skupinam.

Zelo pomemben vidik je tudi uporaba že procesiranih podatkov in učinkovita dostava teh podatkov v obliki informacij do uporabnika. Aplikacije morajo s svojimi uporabniškimi vmesniki uporabnikom olajšati pregled informacij, omogočati enostavno manipulacijo z aplikacijo ter prikazati le tiste podatke, ki jih ti v nekem trenutku potrebujejo.

1.2. Cilji magistrske naloge

Cilj sodobnega kontekstnega stroja je zmanjšanje števila ročnega vnosa podatkov in uporabnikovih potrditev, s čimer težimo k avtomatizaciji in proaktivnemu avtonomnemu pridobivanja podatkov o kontekstu. Da bi lahko prišli do kvalitetnih rezultatov je potrebno

uporabnika spremljati vseskozi. Zaradi teh razlogov so za zajem konteksta najbolj primerne razne prenosne mobilne naprave, ki so vseskozi v kontaktu z uporabnikom. Standardi prenosnih naprav in njihovih operacijskih sistemov so se v zadnjem času že občutno razvili saj lahko, v določenih pogledih, že konkurirajo klasičnim računalnikom. V preteklosti so bile te naprave programsko in predvsem strojno zelo omejene, zato je zbiranje podatkov ter kasnejše procesiranje konteksta prevzel ločen kontekstni stroj.

V magistrskem delu skušamo lastnosti klasičnih kontekstnih strojev prenesti na mobilno platformo in doseči njihovo optimalno delovanje. Takšen pristop omogoča pridobivanje podatkov iz bogatega nabora kontekstnih virov, ki se nahajajo na mobilni napravi. Kontekstni podatki in informacije se z uporabo takšnega kontekstnega stroja ne bodo več prenašale po omrežju, ampak se bodo hranile izključno na mobilni napravi. Poleg samega kontekstnega stroja zelo veliko vlogo v takem principu odigrajo tudi kontekstno odvisne aplikacije, ki za svoje delovanje uporabljajo module mobilnega kontekstnega strežnika. Poleg enostavnih podatkov iz senzorjev, ki jih nato združimo v bolj kompleksne kontekstne informacije, bodo v tem principu veliko vlogo odigrali tudi podatki o različnih uporabnikovih interakcijah, ki bodo uporabljeni v praktični aplikaciji.

Zaradi boljše preglednosti bo razvoj razdeljen v več različnih sklopov. Prvi bo obsegal zajem konteksta trenutne situacije (kot je lokacija, fizične razmere iz okolja, socialna situacija uporabnika, ipd.) na mobilni napravi, drugi obravnaval hranjenje zbranih podatkov v lokalni podatkovni bazi in procesiranje teh podatkov. Zadnji, tretji sklop, bo obravnaval deljenje procesiranih informacij končnim aplikacijam.

Končni rezultat zasnove kontekstnega strežnika bo tako storitev (ang. *service*), ki bo preko razpoložljivih virov v ozadju zajemala kontekst uporabnika, njegovega okolja in trenutne situacije, ter nato po postopkih kontekstnega procesiranja v aplikaciji uporabila te informacije. Namen takšne aplikacije je, da svojo vsebino bolj prilagodi željam in zahtevam uporabnika in mu tako nudi boljšo uporabniško izkušnjo. S tem je izražen tudi izvirni doprinos tega koncepta; običajne računalniške arhitekture zaradi omejene mobilnosti oz. prenosljivosti od uporabnika ne morejo pridobiti vseh podatkov, s katerimi bi lahko izboljšale svoje delovanje, ga prilagodile uporabniku ali vplivale na izgled svojega vmesnika.

Implementacijo omenjene arhitekture bomo opravili v predzadnjem poglavju. Pri tem bomo preučili in opisali vsa opažanja, ki smo jih zasledili med definicijo osnovne sheme arhitekture, ter opravili tudi osnovno zmogljivostno testiranje naše implementacije kontekstnega procesiranja na mobilni napravi, ki jih bomo v temu delu tudi primerjali z rezultati na običajnem računalniškem sistemu. Vsi ti podatki bodo uporabljeni v zadnjem delu evalvacije, kjer bomo primernost celotno strukture strežnika in pripadajoče aplikacije ocenili z analizo SWOT.

2. KONTEKST IN KONTEKSTNA ODVISNOST

2.1. Definicija konteksta

Pojem konteksta (ang. *context*) in kontekstne odvisnosti (ang. *context dependency*) obsega dostavo relevantnih podatkov napravam in ciljnim aplikacijam za uporabo posameznih uporabnikov. Kontekst lahko glede na tip razdelimo v tri dele:

- **Kontekst (trenutne) situacije –** obravnava situacijo, v kateri se nahaja uporabnik ali obravnavana entiteta. V primeru uporabnika so senzorji in drugi viri podatkov lahko že vključeni v napravi sami ali bližnji okolici, lahko pa so to podatki pridobljeni preko inteligentnih naprav v okolici entitete, ki jo opazujemo. Sistem mora biti poleg opazovanja dejanskega stanja sposoben tudi shranjevati podatke preteklih stanj za potrebe njihovih primerjav in klasifikacij njihovih razlik.
- **Kontekst informacijskih potreb uporabnika in njegovega osebnega stanja –** obsega celovit pogled na potrebe uporabnika v stanju mobilnosti, kot so: tip in količina željenih tipov informacij, njegove pretekle navade, sedanja in pretekla fizična interakcija z okoljem in socialna interakcija z drugimi entitetami. Omenjeni kontekst se lahko zelo razlikuje glede na tip osebe, ki uporablja aplikacijo (primer primerjave poslovneža in turista), se lahko v določenih situacijah hipoma spreminja in je lahko odvisna od lastnosti trenutnega okolja.
- **Kontekst tehnološkega opis –** definira tehnološke zmožnosti in specifikacije posamezne naprave, na kateri se izvajajo aplikacije in storitve, ki kontekst uporabljajo. Od vseh naštetih se ta v realnosti najmanj spreminja, vseeno pa mora sistem spremljati morebitne spremembe na napravah. V primeru deljenega konteksta in interakcije med napravami v okolici se morajo upoštevati tudi lastnosti sosednjih naprav.

2.2. Zaznavanje in zajem konteksta

Kontekst lahko zaznavamo in zajemamo na različne načine in na različnih mestih. Opazujemo lahko uporabnikovo interakcijo z nekim informacijskim sistemom ter opazujemo njegove akcije in odločitve, njegova plačila s kreditno kartico in naročili v spletni trgovini. Tudi, če bi lahko zajeli vse sisteme s katerimi uporabnik komunicira in sodeluje, ne bi mogli zaseči celotnega konteksta uporabnika. Pri tem bi lahko izpustili morebitne dejavnike okolja, socialnih vplivov in ostalih dejavnikov, ki jih v določenih situacijah težko obravnavamo. Takšen

kontekst lahko označimo kot parcialnega. Da bi lahko posamezni ponudniki storitev in izdelkov lažje prišli do podatkov o uporabnikih, njihovih navadah, željah in mnenjih, so v ta namen vzpostavili bolj ali manj kompleksne ter razširjene sisteme ter uporabnikom za določene ugodnosti ponudili različne identifikacijske kartice, s katerim lahko beležijo njihovo interakcijo s sistemom. Manjkajoče podatke so skušali pridobiti z anketami, intervjuji in ostalimi metodami. Še bolj napredno zajemanje uporabnikovih aktivnosti poteka na internetu in na mobilnih platformah. Številni ponudniki različnih storitev vlagajo veliko energije in sredstev v to, da bi lahko uporabnikom ponudili ravno tisto, kar v tistem trenutku želijo. To morajo opraviti hitro in za uporabnika čim bolj transparentno. Uporabniki so namreč tisti, ki diktirajo uporabo storitev in od nje pričakujejo, da deluje v skladu z njihovimi željami. Podobno kot ponudniki klasičnih storitev v trgovinah tudi ti iščejo različne načine za pridobivanje podatkov, ki morajo biti, poleg mnogih tehnoloških omejitev, tudi v skladu z veljavno zakonodajo.

2.3. Uporaba konteksta v aplikacijah

Mnoge današnje spletne in mobilne aplikacije uporabljajo različne aspekte konteksta, da bi lahko uporabniku podali določene informacije. Pri nekaterih ti podatki ne predstavljajo samo možnost obogatitve uporabniške izkušnje ampak so njena gonilna sila za delovanje. Nekaj primerov obstoječih aplikacij, ki za svojo delovanje uporabljajo kontekstne podatke lahko najdemo v razdelku 4.3.

Da bi lahko kontekstno procesiranje hitreje, bolj učinkovito ter natančno izvajali, so bili s strani omenjenih ponudnikov storitev definirani in razviti različni tipi posebnih strojev, ki so zmožni procesiranja in hranjenja velikih količin konteksta. Ti so bili običajno implementirani v večje med seboj povezane celote, ki jih imenujemo kontekstni strežniki. Kontekst, ki ga pridobimo preko kontekstnega strežnika, lahko v aplikacijah ali spletnih storitvah uporabimo na več različnih načinov. Z njim lahko obogatimo že prej obstoječe implementacije in jim s tem dodamo nove funkcionalnosti za uporabnike. Primer takšne situacije je razširitev aplikacije za turizem, ki bi omogočala prikaz najbližjih turističnih znamenitosti v okolici turista. Omenjeno funkcionalnost bi lahko vključevala že aplikacija sama, v kolikor pa ob tem upoštevamo tudi osebne preference samega uporabnika (naklonjenost določenem umetnostnem slogu, tipu znamenitosti itd.), pa je to veliko enostavneje implementirati preko kontekstnega strežnika. Takšna avtonomna aplikacija bi morala vseskozi teči v ozadju in sama poskrbeti za logiko pridobivanja podatkov ter njihovega procesiranja.

Drugi tip oz. način uporabe konteksta pa je definiranje popolnoma novih tipov aplikacij in storitev za uporabo konteksta. V teh so podatki, ki se pridobijo preko kontekstnega strežnika, glavni viri podatkov za vse operacije, ki jih ta izvaja. Spekter takšnih aplikacij je lahko zelo

velik, odvisen je samo od zmožnosti samega kontekstnega strežnika. Prednosti takšnih aplikacij proti klasičnim avtonomnim aplikacijah, ki same pridobivajo in procesirajo podatke, so v tem, da lahko določen podatek v nekem intervalu pridobimo samo enkrat, ter ga nato le delimo med več aplikacij hkrati, kar vodi v manjšo porabo virov na napravi. Tudi razvijalci imajo lahko zaradi tega manj težav, saj se jim ni treba toliko posvetiti samemu pridobivanju podatkov v ozadju, ampak se lahko bolj osredotočijo na samo predstavitev pridobljenih podatkov in dobri strukturiranosti aplikacije.

3. UPORABLJENE TEHNOLOGIJE

3.1. Android SDK

Mobilno računalništvo je danes dostopno bolj kot kdajkoli prej. Običajne mobilne naprave so se sčasoma vse bolj razvijale in poleg svojega osnovnega namena, telefoniranja in pošiljanja kratkih sporočil, pridobivale vse več funkcij in dodatkov. Postopoma so te naprave prevzemale lastnosti žepnih računalnikov in jih zaradi vse večje popularnosti v celoti nadomestile. Da bi lahko takšne naprave bolj enostavno ločili od njihovih predhodnikov, jih imenujemo pametni telefoni (ang. *smartphone*).

Omenjena transformacije mobilnih naprav ne bi bila mogoča brez pripadajočega operacijskega sistema in njegovih aplikacij. To dejstvo so v zgodnjih letih razvoja pametnih telefonov, poleg proizvajalcev običajnih mobilnih naprav, prepoznala tudi nekatera podjetja, ki pa na omenjenem področju še niso bila prisotna. Da bi lahko konkurirala omenjenim proizvajalcem pametnih telefonov so svoje operacijske sisteme razvile in približale končnemu uporabniku, hkrati pa so se osredotočili tudi na bodoče razvijalce mobilnih aplikacij in jim izpostavili napredna ter dobro dokumentirana razvijalna ogrodja, s pomočjo katerih so ti lahko razvijali različne aplikacije za lastno ali komercialno uporabo. Omenjeni pristop je tem podjetnim sčasoma prinesel prevlado na trgu sistemov pametnih naprav, medtem ko so proizvajalci mobilnih naprav svoje lastne sisteme počasi ukinjali. Nekateri proizvajalci so zaradi vztrajanja na lastnih sistemih izgubili velik tržni delež in bili celo prisiljeni opustiti omenjeni segment.

3.1.1. Lastnosti operacijskega sistema

Android je operacijski sistem za različne tipe mobilnih naprav, kot so pametne mobilne naprave in tablični računalniki. Zasnovan je na prirejeni verziji operacijskega sistema Linux skupaj z njegovim jedrom, ki skrbi za upravljanje naprav, pomnilnika in procesov. Poleg omenjenega jedra ima vgrajene še različne knjižnice, ki podpirajo delovanje različnih storitev, ko so telefonija, video, govor, povezovanje, programiranje uporabniškega vmesnika in ostalih osnovnih aspektov sistema.

Ob izidu operacijskega sistema je Google želel, da bi ta ostal odprt in brezplačen. Izvorno kodo je zato izdal pod odprtokodno licenco Apache s čimer jo lahko vsakdo prenese, uporablja in v skladu s svojimi željami tudi spreminja [1]. Na spletu se tako najde veliko uporabniško spremenjenih različic sistema, ki se osredotočajo na boljše in hitrejše delovanje ter

implementacijo lastnega uporabniškega vmesnika. Še bolj korenito so to lastnost izkoristili nekateri proizvajalci pametnih naprav, ki so, v želji po boljši prepoznavi njihovega sistema in lastne blagovne znamke, prilagodili uporabniški vmesnik, dodali nekatere lastne aplikacije in orodja ter ga prilagodili za uporabo z novimi tehnologijami, ki so bili predstavljene z izdajami novih telefonov.

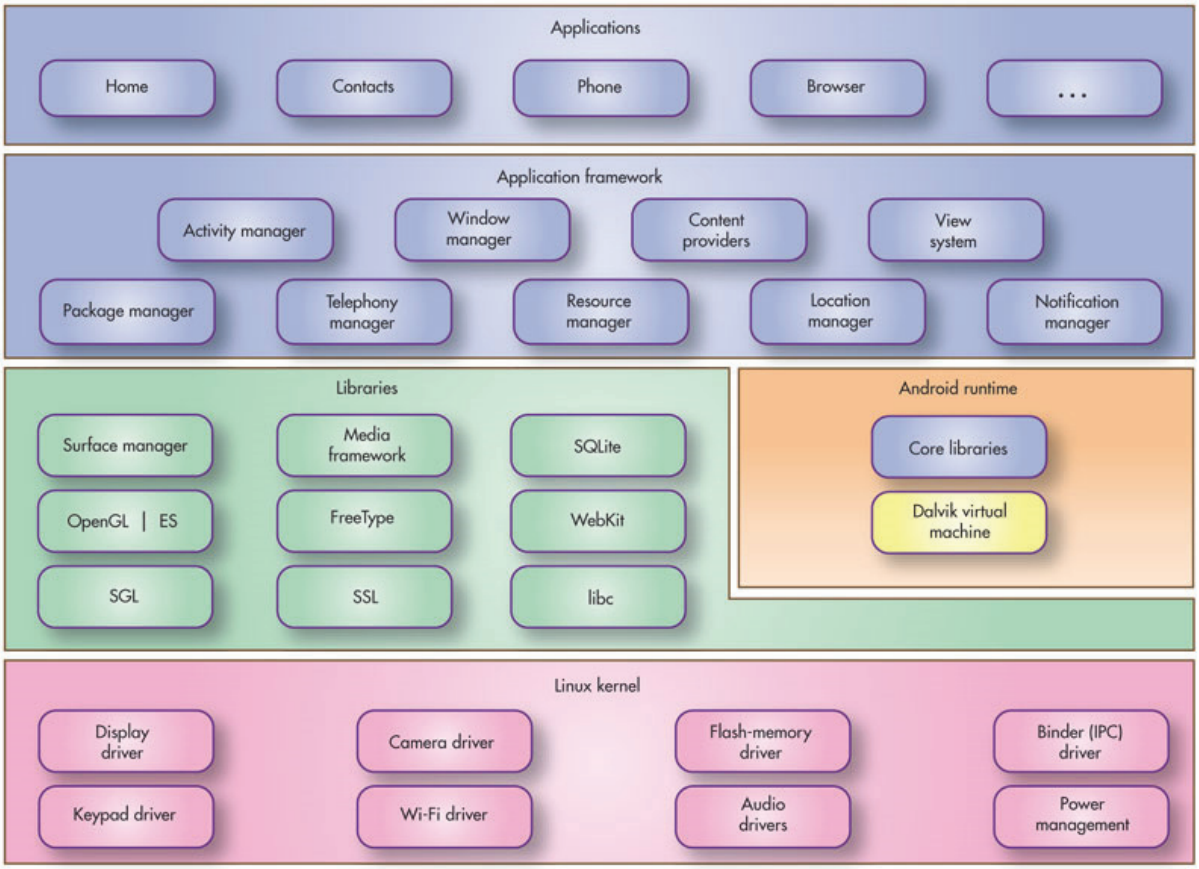
Kljub temu, da je ob njegovem izidu kar nekaj obstoječih mobilnih sistemov že omogočalo večino njegovih specifikacij, pa je ta sistem v eno celoto združil vse omenjene lastnosti:

- **Odprtokoden, brezplačen sistem –** Proizvajalci naprav so se namesto razvoja lastnih sistemov lažje osredotočili na izdelovanje in razvoj boljših naprav. Razvijalci pa so lahko izbrali sistem, ki ni omejen samo na enega proizvajalca. Njihova aplikacija je tako lahko delovala na več različnih napravah.
- **Komponento odvisna arhitektura –** Posameznim vgrajenim komponentam lahko spremenimo izgled in s pomočjo metod spremenimo njihovo izvajanje ali obnašanje. Omenjene spremembe se lahko uveljavijo tudi med delovanjem aplikacije ali na uporabnikovo zahtevo.
- **Velik nabor vgrajenih storitev –** Sistem za vzpostavitev različnih storitev na potrebuje uvažanje zunanjih knjižnic, saj so te že vgrajene v sistem, dobro preizkušene in vsebujejo dodelan nabor dokumentacije. Primer je storitev za določanje lokacije, ki je zmožna lokacijo določiti s uporabo mobilnega omrežja, naslova IP trenutnega brezžičnega omrežja ali uporabo sistema pozicioniranja s pomočjo satelitov.
- **Upravljanje življenjskega cikla aplikacij –** Posamezne aplikacije so izolirane drug od druge z implementacijo različnih varnostnih nivojev. Kljub temu lahko komunicirajo med seboj s pomočjo internega sporočanja in namer (ang. *Intents*). Aplikacije imajo že vnaprej definirana stanja, v katerih se lahko nahajajo, prav tako so določeni tudi prehodi med posameznimi stanji.
- **Uporaba naprednih knjižnic za multimedijo –** Sistem omogoča delovanje bogatih multimedijskih aplikacij in animacij, podprtih 3D pospeševalnikom OpenGL. Hkrati sistem vsebuje večino industrijskih kodekov za predvajanje različnih glasbenih in filmskih formatov, kot so MP4, H.264, AMR AAC, MP3, ipd.

3.1.2. **Arhitektura**

Platformo Android lahko v grobem opišemo kot programsko skladovno arhitekturo, predstavljeno na spodnji shemi (Slika 1). Vsak nivo te arhitekture združuje različne programe, razrede in knjižnice, ki skupaj tvorijo celoten operacijski sistem. Na najnižjem nivoju se nahaja jedro Linux, ki podpira osnovne funkcije delovanja operacijskega sistema. Na naslednjem

nivoju najdemo nabor knjižnic, implementiranih v jeziku C/C++ ter osrednje izvajalno okolje. Ta vsebuje izvajalni stroj Dalvik, ki se deluje kot samostojna naprava. Ta lahko namreč izvaja posamezne ukaze ne glede na kateri arhitekturi se nahaja. Za svoje delovanje potrebuje samo nabor jedrnih knjižnic (ang. *core libaries*), ki so vključene v ta nivo. Sledi nivo definicij osnovnih razredov razvijalnega ogrodja, ki se uporabljajo za podporo delovanja posameznih aplikacij. Omenjene aplikacije se nahajajo na najvišjem nivoju.



Slika 1: Shema arhitekture operacijskega sistema Android po posameznih slojih.

Za namene razvoja aplikacij Android vsebuje paket za razvoj programske opreme (ang. *Software Development Kit, SDK*), ki podpira večino funkcionalnosti platforme Java SE (Java Standard Edition), razen njene razširitve za ustvarjanje uporabniškega vmesnika AWT/Swing. Android namreč ponuja lastno razširitev definiranja uporabniškega vmesnika, ki omogoča definiranje gradnikov v jeziku XML in njihovo enostavno manipulacijo v metodah posameznih aktivnosti. S pomočjo omenjenega paketa lahko aplikacije programiramo v jeziku Java na enak način kot pri javanskih aplikacijah za klasične računalniške sisteme. Pri tem se omenjeni razredi pretvorijo v bitno kodo in izvajajo na mobilni napravi.

Edina razlika med njima je, da ta platforma Android ne vsebuje standardnega javanskega navideznega stroja (ang. *Java Virtual Machine, JVM*). Omenjena arhitektura mora namreč med

pretvorbo omenjene razrede še dodatno optimizirati, da lahko ti bolj optimalno deluje na pomnilniško in procesno bolj omejeni platformi.

Za izvajanje omenjen kode se tako uporablja virtualni stroj Dalvik (ang. *Dalvik Virtual Machine*). Ta uporablja nižje ležeče jedro za izvajanje osnovnih funkcionalnosti, kot so: nitenje, zagotavljanje varnosti aplikacij ter upravljanje s procesi in pomnilnikom. Omenjeni stroj deluje tako, da najprej združi vse uporabljene datoteke z razredi v eno ali več izvršljivih datotek navideznega stroja, imenovane tudi DEX (ang. *Dalvik EXecutable*). Po omenjenih postopkih optimizacije kode, je takšna koda pripravljena za izvršitev.

3.2. Baza SQLite

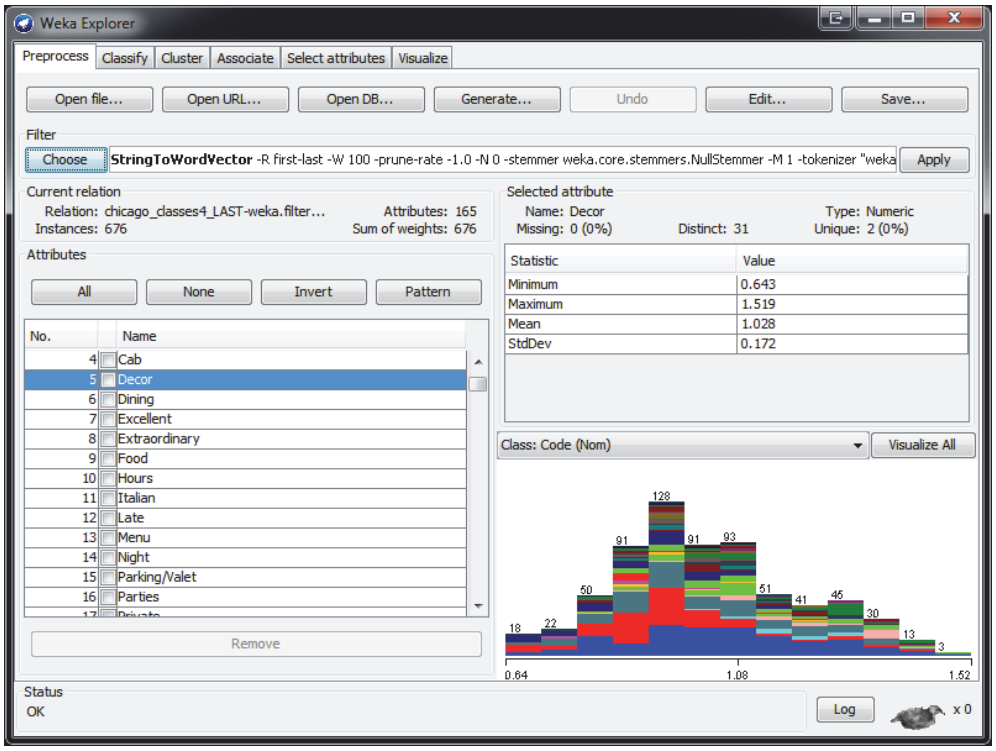
SQLite je relacijska podatkovna baza, katere osnovna različica je bila bazirana na programskem jeziku C. Njena izvorna koda je v obliki javne domene, zato se lahko prosto uporablja in distribuira . Za razliko od nekaterih ostalih podatkovnih baz ta ne potrebuje posebnega strežnika z ločenim procesom za dostop do baze, ampak je po navadi že del same aplikacije ali operacijskega sistema. Različne implementacije za operacijske sisteme so prostorsko in spominsko zelo malo zahtevne, prav tako pa je z uporabo niti mogoče sočasno branje podatkov iz baze. Zaradi teh lastnosti se večinoma uporablja kot rešitev za shranjevanje podatkov internetnih brskalnikov (zgodovina brskanja), operacijske sisteme za prenosne in druge naprave ter razne posebno namenske (ang. *embedded*) sisteme.

SQLlite je tako prisotna tudi v sistemu Android in kot knjižnica vgrajena v osrednji del. Zaradi tega je programerjem ni potrebno vključevati v njihove aplikacije. To je sicer velika prednost, vseeno pa je potrebno upoštevati dejstvo, da je inačica izdaje te knjižnice pogojena s samo verzijo sistema Android na posamezni napravi. Primerek baze je predstavljen z zapisi, shranjenimi kot običajna samostojna datoteka na disku.

3.3. Weka

Weka (*Waikato Environment for Knowledge Analysis*) je odprtokodno programsko orodje za podatkovno rudarjenje in strojno učenje, ki je bilo razvito na univerzi Waikato na Novi Zelandiji in izdano pod odprtokodno licenco GNU GPL (ang. *General Public Licence*). Orodje je bilo v osnovni različici (leta 1993) izdelano na osnovi programskega jezika C in TCL/TK, kasneje, leta 1997, pa je bilo skupaj z implementacijami posameznih algoritmov v celoti predelano v jeziku Java. Vsebuje več tipov orodij in algoritmov za identifikacijo informacij iz

osnovnih oz. surovih podatkov in podpira večino standardnih metod za obdelavo teh podatkov, kot so predpriprava in predobdelava podatkov, klasifikacija, razvrščanje, regresija, vizualizacija podatkov in izbira značilk oz. atributov. Vse te operacije se lahko v orodju vršijo neposredno preko grafičnega vmesnika, kjer lahko spremljamo tudi izpis morebitnih rezultatov, ki je dodan analizi izvedbe posameznega koraka. Dodatno so v grafični vmesnik vključena tudi orodja za vizualizacijo podatkov ter vmesnih ali končnih rezultatov. Poleg grafičnega vmesnika lahko z orodjem manipuliramo tudi preko ukazne vrstice (ang. *Command Line Interface, CLI*), za razvijalce zelo uporabna pa je predvsem uradna javno dostopna knjižnica v obliki javanskih razredov in pripadajoče izvorne kode ter že prevedenih in arhiviranih razredov v obliki JAR datoteke, ki se lahko uporablja v lastnih programih.



Slika 2: Okno grafičnega vmesnika orodja Weka.

Podprtih je več tipov formata vhodnih podatkov, vendar se ti pred uporabo v orodju pretvorijo v ASCII tekstovne datoteke, predstavljene s kratico ARFF (ang. *Attribute-Relation File Format*), ki predstavljajo posamezne instance opisane z atributi. Takšna datoteka je razdeljena na dva dela: glavo, ki opisuje tip ali možne vrednosti posameznih atributov in relacij med njimi, ter podatkovni del, kjer so predstavljene instance podatkov (Programska koda 1). Takšna pretvorba se izvede samodejno, pogosto pa je potrebno zaradi lastnosti posameznih algoritmov tipe podatkov ročno pretvoriti v različne oblike. S tem tipom datoteke je pogosta predstavljena tudi končna oblika rezultatov.

```
@relation weather
@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

Programska koda 1: Primer zgradbe preproste datoteke v formatu ARFF.

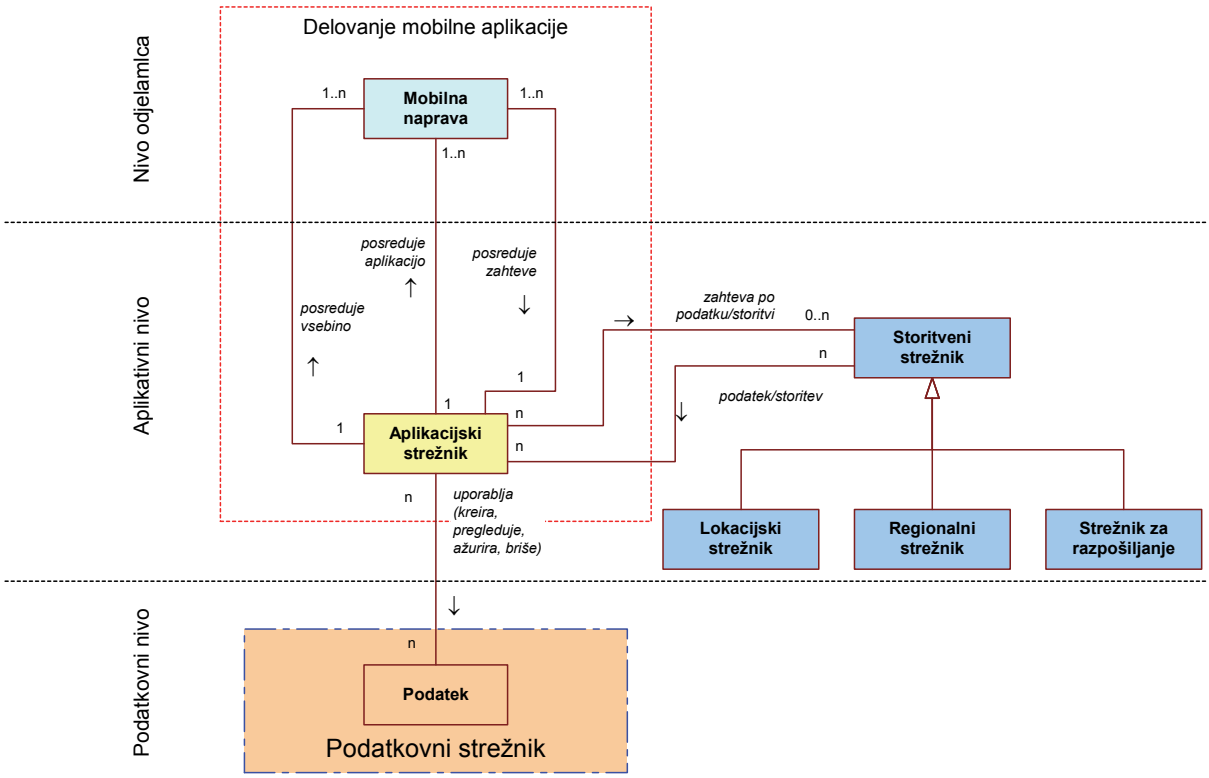
4. KLASIČEN MODEL OBRAVNAVANJA KONTEKSTA

Razne storitve in aplikacije so v zadnjem času postale stalnica za veliko število končnih uporabnikov, ki te jih uporabljajo vsakodnevno. Težnja po obravnavanju konteksta v teh aplikacijah se ni pojavila sama od sebe, ampak so jo z vse večjimi potrebami po informacijah diktirali uporabniki. Te morajo biti vseskozi dostopne in delovati v skladu z uporabnikovimi željami, izraženimi preko nastavitev v aplikacijah, izbranih filtrih, vnesenih virih za dostop do novičarskih skupin, definiciji urnika osveževanja vsebine in ostalih ročno nastavljenih ali vnesenih vnosov, s katerimi si lahko uporabniki sami prilagodijo format in tip vsebine, ki jo želijo prejemati. Slednji način zagotavlja, da si uporabnik sam zagotovi vsebino, ki jo želi prejemati in s tem tudi kontrolira tok dogodkov.

Zgoraj omenjeni pristop pa ima tudi nekaj pomanjkljivosti. Zaradi velikega obsega storitev in možnosti, ki jih te ponujajo, bi bilo takšno vnašanje (vseskozi spreminjajočih) podatkov za uporabnika preobsežno, storitev pa bi s tem postala za uporabnika preveč obremenjujoča in sčasoma tudi neizkoriščena. Zaradi tega so ponudniki teh storitev iskali različne načine, kako bi lahko poenostavili uporabo svojih storitev, jih prilagodili uporabnikovi situaciji in njegovim spreminjajočim zahtevam, ter mu hkrati ponudili tudi informacije, ki bi lahko bile relevantne za uporabnika. Primeri takšnih vsebin so sorodne informacije iz nekega interesnega področja, dogodki v bližini uporabnika, in predvsem storitve s področja oglaševanja, reklamnih akcij, priporočila izdelkov in storitev glede na potrebe, pretekle oglede in nakupe ter priporočila sorodnih akcij uporabnikov, ki imajo podobne zahteve. Z razvojem interneta, spletne prodaje in nudenje spletnih storitev uporabnikom so se zaradi ohranjanja konkurenčne prednosti takšni sistemi razvijali in prilagajali posameznim naročnikom. Zaradi tega v preteklosti ni bilo veliko celovitih odprtokodnih in standardiziranih rešitev, ki bi lahko takšne sisteme približala raziskovalcem in običajnim uporabnikom.

Kljub temu lahko v literaturah najdemo kar nekaj raziskav in konceptov na to temo. Njihov primarni cilj je bil integrirati kontekst v aplikacijo in ga obravnavati kot ključni del za delovanje in obnašanje določene storitve. Skladno z razvojem strežniških arhitektur se v teh raziskavah uporablja porazdeljeni sistem kontekstnega procesiranja, pri katerem ima glavna vlogo pri procesiranje kontekstni strežnik. Običajno je takšen model predstavljen v obliki trinivojske arhitekture (Slika 3). Strežnik v omenjeni vlogi pridobiva podatke od odjemalcev ter jim posreduje kontekstne informacije. Odjemalec je v tem primeru vir podatkov, ki preko različnih komunikacijskih kanalov komunicira s kontekstnim strežnikom in je ob prejemu informacij preko aplikacije zmožen prikazati informacije uporabniku v ustrezni obliki. Poleg procesiranja konteksta se mora strežnik tudi odzivati na morebitne spremembe v kontekstu in ga prilagoditi odjemalcu. Zaradi tega se mora del preteklega konteksta tudi shranjevati v podatkovni shrambi

podatkovnega strežnika in se v rednih intervalih tudi ažurirati, za kar skrbi strežnik preko sistema za upravljanje podatkovnih baz.



Slika 3: Meta-model trinivojske arhitekture kontekstnega strežnika [6].

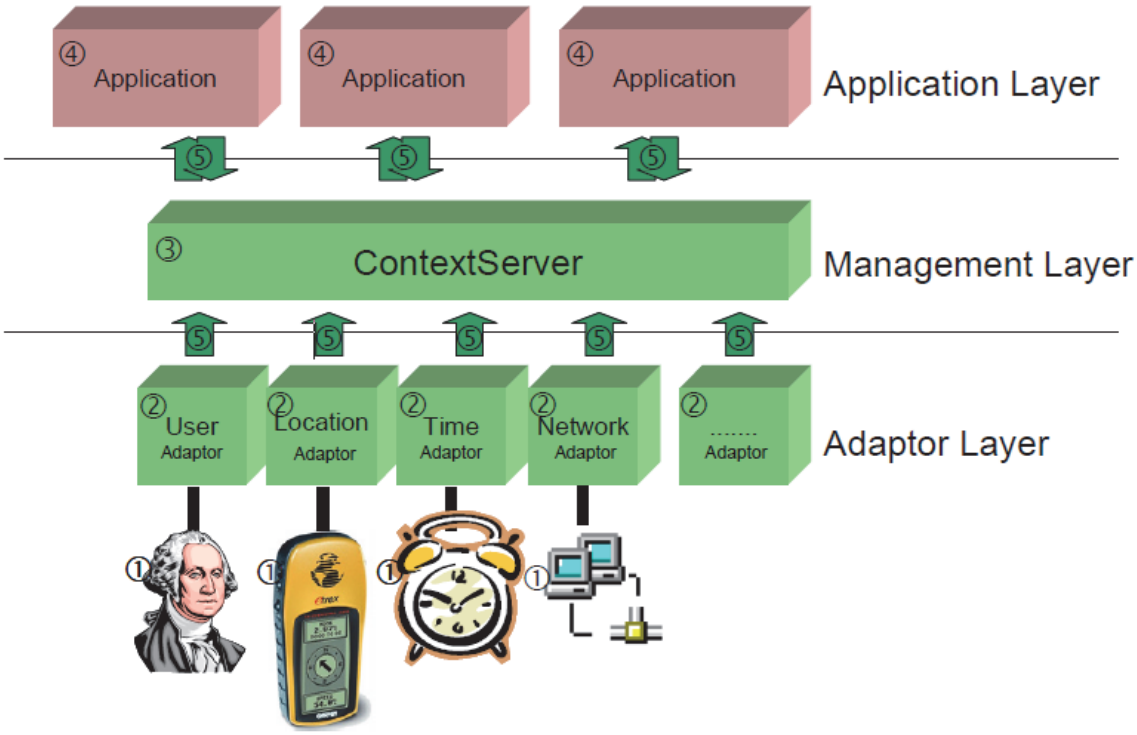
4.1. Pregled definicij obstoječih standardov in implementacij kontekstnega strežnika

Prve omembe kontekstnega procesiranja so močno sovpadale z razvojem različnih tipov brezžičnih komunikacij in prenosnih naprav. Dve desetletji nazaj so tako nastali prvotni opisi uporabe konteksta skupaj z nekaterimi predlogi za bodočo implementacijo različnih vidikohv uporabnikovega konteksta, od fizičnih dejavnikov do osebnostnih in socialnih. V svoji doktorski dizertaciji B.N. Schilit tako omenja nekatere primere prenosljivih računalniških sistemov, njihovo dinamičnost glede na vplive iz okolja ter predstavlja idejo vseprisotnega računalništva (ang. *ubiquitous computing*) [2]. Pri tem se ozira na preteklost in spremembe v tehnologiji računalniških sistemov, od strukture osrednjih računalniških sistemov ter terminalov, enouporabniških delovnih postaj in večuporabniških operacijskih sistemov, do prenosnih računalnikov in prenosljivih mobilnih naprav. V svojem delu zagovarja tezo o definiciji arhitekture, ki bi osveščala aplikacije o spremenljivem računalniškem okolju. Ta bi

bil morala biti preprosta, tolerantna do napak, učinkovita, skalabilna in omogočati implementacijo na različnih računalniških platformah. Konkretno se v delu obravnava predvsem kontekst lokacije, opis entitete, ki sodeluje v kontekstnem okolju, ter interakcija med temi entitetami oz. agenti. Veliko pozornosti je namenjeno tudi komunikaciji s strežnikom in med samimi odjemalci, saj v tistem času mobilna in brezžična omrežja še niso bila razvita. Dejanska implementacija je bila razvita s pomočjo naprav PARCTAB, ki so omogočala vnos osnovnih podatkov o uporabniku in lastnostih naprave, vsa komunikacija pa je potekala preko vgrajenega IR vmesnika do IR sprejemnikov povezanih preko serijske povezave do posebnih prehodov. Ta podatke preda posebni aplikaciji, ki sproži neko akcijo in podatke v obratni smeri preko IR oddajnikov pošlje ciljni napravi.

Medtem ko Schilit v svoji doktorski dizertaciji [2] obravnava kontekst v smislu točno določenih dejavnikov, ki vplivajo na stanje neke entitete, pa ga A.K. Dey opisuje v veliko bolj širšem smislu. Konteksta se ne da poenostaviti in posploševati, ampak ga je potrebno v vsaki situaciji obravnavati kot celoto, to je lahko katerakoli informacija, ki v nekem trenutku opisuje stanje neke entitete [3]. Entiteta je lahko oseba, objekt ali prostor, ki ga lahko štejemo kot relevantnega za interakcijo med uporabnikom in aplikacijo, vključujoč uporabnika in omenjeno aplikacijo.

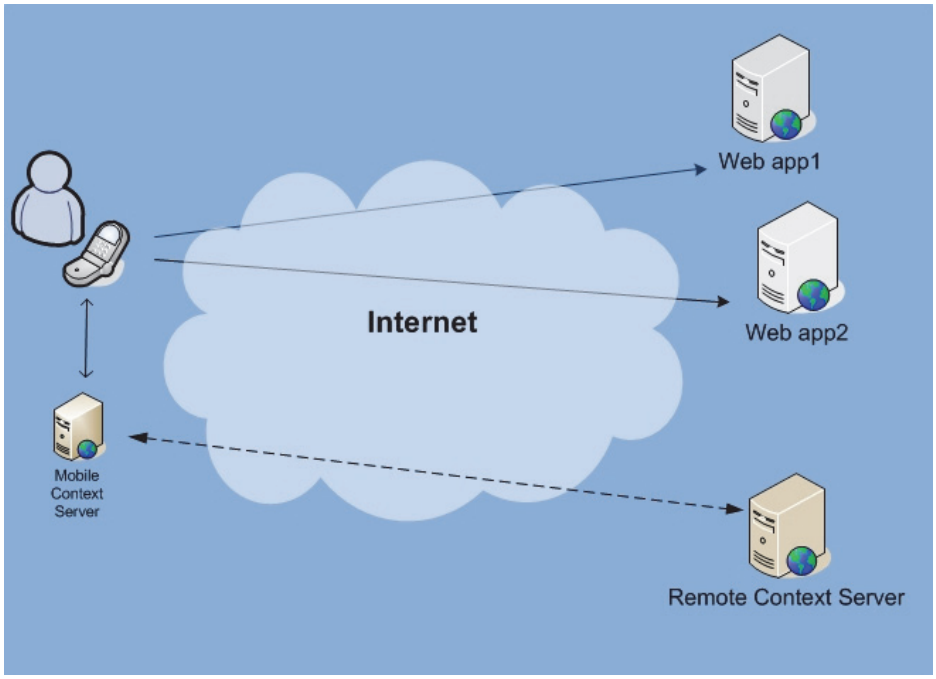
Hydrogen je predlagana tri-nivojska arhitektura, ki skuša zgoraj opisane vidike konteksta uveljaviti v obliki modela, ki bi lahko izboljšal slabosti obstoječih modelov kontekstnega procesiranja ter bil s tem boljše prilagojen mobilnim napravam [4]. V uvodu je namreč poudarjeno, da je bilo do sedaj predlaganih arhitektur kar nekaj, vendar nobena od njih ni prilagojena posebnim zahtevam mobilnih naprav, kot so: omejena povezljivost v omrežje, omejene zmogljivosti računskih zmožnosti ter zajem karakteristik uporabnikov v stanju mobilnosti. Podobno velja tudi za aplikacije, ki osnovne kontekstne podatke zajamejo le takrat, ko jih uporabnik eksplicitno zahteva, pri tem pa je celotno procesiranje podatkov potrebno opraviti v aplikaciji sami. Arhitektura namreč zagovarja stališče, da z vidika razvoja aplikacij zajem, shranjevanje in procesiranje takšnih podatkov ne bi smelo biti v breme razvijalcu ampak bi se ti morali osredotočiti na razvoj same aplikacije in predstavitve podatkov (Slika 4).



Slika 4: Shema strukture arhitekture Hydrogen po slojih.

Med različnimi tipi kontekstno odvisnih arhitektur najdemo tudi nekaj konceptov in idej o mobilnih kontekstnih arhitekturah. V eni od njih avtorji ugotavljajo prednosti prenosnih naprav predvsem v smislu njihove prenosljivosti in možnostih za hranjenju konteksta. Po njihovem mnenju glavni problem za implementacijo predstavlja definicije ustrezne arhitekture, ki bi aplikacijam dovoljevala dostop do kontekstnih podatkov in njihovo uporabo. V ta namen v delu predlagajo mobilno kontekstno ogrodje, imenovano Mobile Contextual Framework (MCF) [5]. Pred definicijo samega ogrodja se dotaknejo tudi vidika platformske neodvisnosti. Za rešitev omenjenega problema, poleg definiranje aplikacije v spletnem brskalniku, omenjajo tako imenovani hibridni način, ki vključuje vzajemno delovanje nižje ležeče platforme in vmesnika, ki je osnovan na eni izmed spletnih tehnologij. Omenjena arhitektura naj bi imela tri ključne lastnosti: univerzalnost (aplikacija mora delovati neodvisno od strojnih in programskih specifikacij), konstantno kontekstno delovanje (aplikacija mora biti sposobna zajema konteksta, ki se lahko vseskozi spreminja), nizkostroškovno delovanje (vzpostavitev, implementacija in vzdrževanja takšnega sistema ne sme zahtevati velikih sredstev). Predlagana arhitektura je tako sestavljena in strežnika na mobilni napravi, ki je zmožen preko HTTP zahtevkov v formatu JSON (ang. *JavaScript Object Notation*) komunicirati z spletnimi storitvami, jim posredovati podatke o kontekstu in prejemati rezultate (Slika 5). Dodatno je bil v arhitekturo vključen še oddaljeni spletni strežnik, ki bi bil namenjen obdelavi bolj kompleksnih procesiranj. Največjo

težavo po avtorjevem mnenju predstavlja predvsem platformsko neodvisno pridobivanje podatkov iz naprave.



Slika 5: Predstavitev strukture mobilne arhitekture MCF.

4.2. Pregled lastnosti mobilnih naprav in njihova klasifikacija

Mobilne naprave so naprave, ki jih predstavljajo običajni mobilni telefoni in pametni telefoni, zraven pa lahko uvrstimo tudi nekatere tipe tabličnih računalnikov. Razlika med njimi se skozi čas spreminjala, vsi so sčasoma postajali bolj tehnološko in programski napredni, zato jih danes že težko kategoriziramo. Glede na njihove lastnosti jih lahko razdelimo v več kategorij:

- **Mobilni telefoni** so naprave, ki podpirajo različne govorne in sporočilne storitve, vključno s povezavo v internet. Naložene imajo tovarniške mobilne operacijske sisteme z omejenimi možnostmi razširitve. Znani so predvsem po svoji enostavnosti uporabe in dolgi avtonomiji baterije.
- **Pametni telefoni** predstavljajo nadgradnjo teh naprav, imajo večje zaslone z boljšimi ločljivosti, boljše strojne karakteristike in ponujajo obsežen pomnilniški prostor za shranjevanje datotek. Znani so predvsem po bolj napredenih operacijskih sistemih in aplikacijah, ki jih uporabniki nameščajo ter konfigurirajo glede na svoje zahteve. Odvisno od karakteristik in odprtosti takšnega operacijskega sistema imajo uporabniki možnost gradnje in razvoja lastnih mobilnih aplikacij, ki jih lahko pod določenimi pogoji tudi delijo z drugimi uporabniki.

- **Tablični računalniki** so naprave, ki so po specifikacijah in programskih lastnosti zelo podobne pametnim telefonom. Običajno ne podpirajo telefonskih govornih storitev, lahko pa preko SIM kartice mobilnega operaterja dostopajo do mobilnega interneta. Za razliko od pametnih telefonov imajo zaradi lažje manipulacije večji zaslon in zaradi tega tudi večjo baterijo, a so vseeno veliko bolj prenosljivi kot klasični prenosni računalniki.
- **Nosljiva tehnologija** predstavlja nove tipe prenosnih naprav, ki zaradi svojih majhnih dimenzij in nizke teže omogočajo nošenje na različnih mestih telesa, oblačil ali modnih dodatkov. Vključujejo pametne ure, očala in ostale podobne naprave. Te so bile v različnih oblikah, dimenzijah in zmogljivostih predstavljene že veliko prej v preteklosti, a do sedaj s strani proizvajalcev prenosnih naprav in operacijskih sistemov še ni bilo toliko poudarka na razvoju teh tehnologij. Po njihovih besedah naj bi bilo prav leto 2014 na tem področju prelomno.

4.3. Zasnove in implementacije kontekstno odvisnih aplikacij

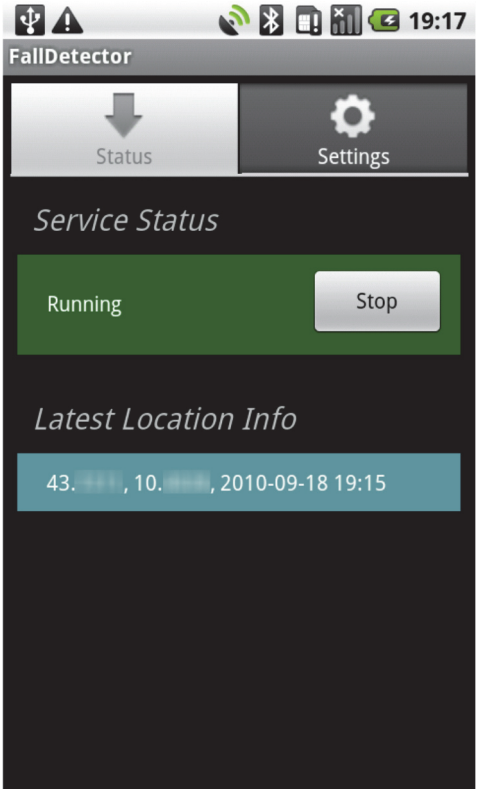
Skladno z razvojem kontekstnih strežnikov so se kot demonstracija njihove uporabnosti pojavile tudi različne kontekstno odvisne aplikacije. Glede na predlagano arhitekturo so imele različno vlogo, večina od njih pa je služila za prikaz rezultatov kontekstnega procesiranja ter tudi za zajem konteksta preko vgrajenih senzorjev. Nekatere od njih, predvsem tiste, ki so bile implementirane na novejših tipih naprav, so bile programirane tako, da same prevzamejo vse funkcije zajema konteksta, njegovega procesiranja in predstavitve rezultatov.

Običajne kontekstno odvisne aplikacije imajo tako naslednje lastnosti in karakteristike:

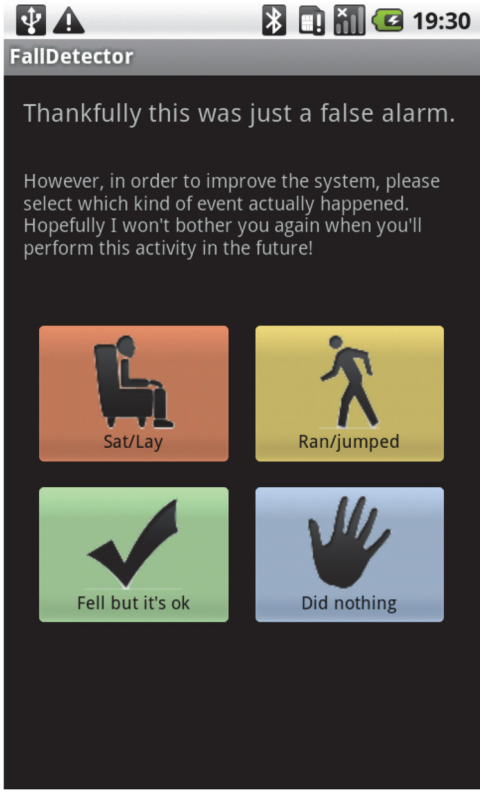
- Predstavitev informacij in storitev uporabniku,
- Samodejna izvedba storitve za uporabnika,
- Označevanje konteksta za omogočanje kasnejšega ponovnega pridobivanja.

Primer samostojne kontekstno odvisne aplikacije je mobilna aplikacija za avtomatično zaznavanje padcev oseb [10]. Namenjena je ljudem pri katerih je zaradi starosti, bolezni ali različnih telesnih napak povečana možnost nenadzorovanega padca. Takšna aplikacija seveda teh padcev ne more preprečiti, lahko pa jih zazna ter v primeru neodziva osebe zaradi poškodbe ali začasne izgube zavesti sproži ene od vnaprej določenih akcij. Aplikacija je zanimiva predvsem zato, saj za delovanje poleg osnovnih podatkov o opazovani osebi potrebuje le napravo z vgrajenim merilnikom pospeška. Pri razvijanju aplikacije je bila metoda z uporabo vgrajenega merilnika pospeška primerjana z uporabo eksternega merilnika pospeška, ki naj bi v večini primerov zagotavljal boljšo natančnost in občutljivost zaznave. Na začetku testiranja aplikacije so določil testni nabor ciljnih uporabnikov, 6 moških in 4 ženske, starih med 60 in

82 let, ki so jim dali aplikacijo v test in jih po koncu prosili za izpolnitev pred pripravljenega vprašalnika, kjer so ti ocenili primernost aplikacije. V test so vključili tudi veliko primerov napačne klasifikacije dogodkov, ki ne bi smeli biti obravnavani kot padec (dogodki, ko se oseba uleže v posteljo, sede na stol, hodi navzdol po stopnicah, izstopi iz avta, se z roko dotakne senzorja in ostale). Rezultati vprašalnika so avtorjem pomagali pri boljši organiziranosti aplikacije in študiji primernosti uporabe ločenega senzorja za detekcijo, medtem ko so meritve vrednosti pri posameznih dogodkih pripomogle k izgradnji bolj natančnega modela za detekcijo tipa padcev (padec na mestu, padec nazaj, padec naprej) ter prepoznava lažnih dogodkov (Slika 7). Pri izgradnji modela so raziskovalci upoštevali več različni tehnik prepoznavanja padcev ter nato te modele ocenili z oceno senzitivnosti (ang. *sensitivity*) ter specifičnosti (ang. s*pecificity*) in točnosti (ang. *accuracy*). Pri eni izmed metod so se avtorji pri oceni senzitivnosti približali vrednosti 1 (oz. 100%). To pomeni, da bi lahko z izgradnjo takšnega modela uspešno klasificirali skoraj vsak padec.



Slika 6: Samostojna kontekstna odvisna aplikacija za detekcijo padcev FallDetector [10].

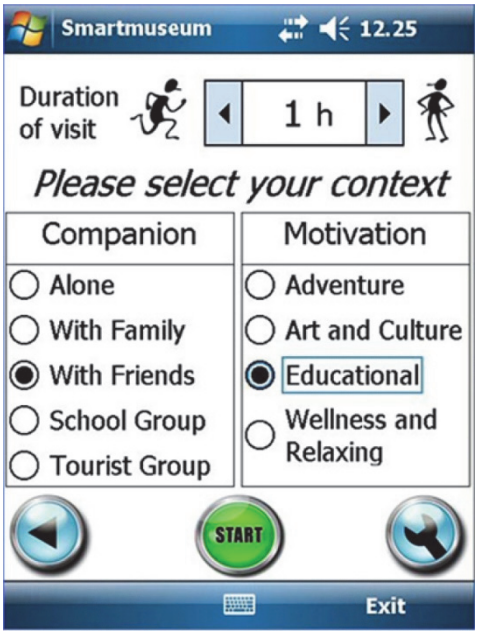


Slika 7: Okno aplikacije za klasifikacijo dogodka, ki v bodoče ne bi smel biti obravnavan kot padec.

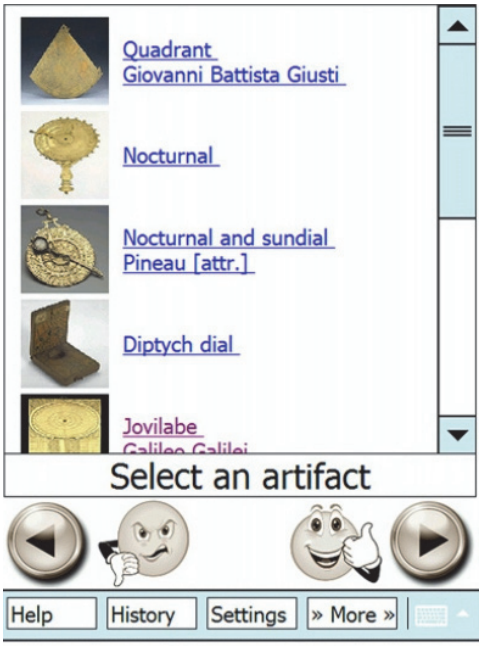
Zelo interesantne so tudi aplikacije, ki bazirajo na osnovi delovanja priporočilnih sistemov. Te lahko delujejo na različne načine, prilagodijo se lahko uporabnikovi interakciji s sistemom v

preteklosti, mu priporočijo vsebino glede na interakcijo ostalih (sorodnih) uporabnikov sistema, upoštevajo njegov osebnostni profil in njegovo trenutno stanje, in ostale. Da bi lahko uporabnika obravnavali kot aktivni del kontekstnega okolja je potrebno nekako zajeti njegovo stanje. Ena izmed metod zajema takšnih podatkov so bili elektronski dnevniki. Vanje so opazovani uporabniki, študenti, dnevno vpisovali podatke o svojih dejavnostih, ki so jih ti sistemi uporabljali za napovedovanje bodočih aktivnosti uporabnika. Dodatno so bili s kandidatov tedensko opravljeni še razgovori o poteku teh aktivnosti in njihovih mnenjih, na podlagi katerih so izvedli osnovno profiliranje [11]. S pomočjo teh podatkov so lahko raziskovalci prilagodili kontekstno delovanje njihove aplikacije na način, ki bi študentom omogočal lažjo organizacijo njihovega dela in glede na njihov način in kraj učenja tudi predlagal uporabo ustreznih učnih materialov.

Omenimo lahko tudi študijo *SMARTMUSEUM*, ki opisuje zasnovo priporočilnega sistema za priporočanje različnih znamenitosti, kot so muzeji ali stavbe z zgodovinsko arhitekturno vrednostjo, skupaj z vsemi umetniškimi stvaritvami na tej lokaciji [9]. Priporočilni sistem je predlagan z namenom boljše predstave in učinkovitejše dostave te informacije končnim uporabnikom. Za svoje delovanje predvideva več različnih scenarijev, ki obsegajo zajem različnega konteksta in predstavljenih rezultatov. Ko je tak uporabnik zunaj, mu sistem prikaže znamenitosti, ki so v bližini. Pri tem kombinira podatke o uporabnikovi lokaciji, njegovemu osebnostnemu profilu in interesih ter trenutnemu času. Njegov kontekst se nato primerja s lokacijami znamenitosti, njihovim značilnostim ter morebitnemu omejenemu delovnemu času. Ob obisku znamenitosti sistem preklopi v notranji način, ki uporabnika predlaga posamezne objekte znotraj omenjene stavbe, s pomočjo tehnologije RFID pa uporabnik lahko pridobi dodatne informacije o izbranem objektu. Pred samim obiskom mora tak uporabnik preko obrazca na prenosni napravi vnesti podatke o predvidenem trajanju obiska, ali je prišel sam ali v skupini, njegovo trenutno motivacijo in želje, ter ostale (Slika 8). Med samim obiskom lahko posamezne objekte uporabnik tudi komentira in poda oceno njihove predstavitve (Slika 9). Tretji scenarij obsega ogled in določitev osebnostnega profila ter interesov preko spletne aplikacije na osebnem računalniku. Ta je veliko bolj prijazna za uporabo kot enostavni mobilni obrazec, hkrati pa se lahko preko nje vnese še več podrobnejših podatkov.



Slika 8: Okno aplikacije Smartmuseum za vnos podatkov o trenutnem kontekstu uporabnika.



Slika 9: Prikaz objektov znotraj kulturne znamenitosti v aplikaciji.

Pri implementaciji omenjenega sistema se avtorji zanašajo na uporabo večnivojske arhitekture za zaznavo konteksta, profiliranje uporabnikov, filtriranje rezultatov in iskanje sorodnih elementov. Pri zaznavi konteksta se avtorji osredotočajo na zajem podatkov, ki je možen preko senzorjev ter pridobivanju podatkov, ki jih uporabniki morajo vnesti sami. To so prej omenjeni podatki o dolžini obiska, oceni posameznih znamenitosti in njihove predstavitve in ostali. Ostali nivoji arhitekture pri svojem delovanju uporabljajo tehniko ontologij v obliki shem RDF. Podatki so namreč opisani v obliki tehnike *triplestore*, kjer se za opis neke entitete uporabljajo trije elementi, ki vsebujejo fizični opis entitete z osnovnimi lastnostmi elementa, vsebinski opis z informacijami o avtorju, kraju izdelave in uporabljenimi tehnikami ter kontekstni opis entitete, ki vsebuje podatke o morebitni ciljni publiki in razlogih za ogled tega subjekta.

5. PREDLOG IMPLEMENTACIJE STREŽNIKA NA MOBILNI NAPRAVI

5.1. Razlogi za implementacijo in primerjava s klasičnim modelom

Na začetku snovanja predlagane arhitekture smo najprej pregledali literaturo, članke in ostale vire o kontekstnem procesiranju, da bi si lahko ustvarili sliko o arhitekturi in delovanju kontekstnih strežnikov ter njihovih osnovnih karakteristikah. Večina te literature opisuje kontekstne strežnika, ki so nameščeni na posebej za to namenjenem računalniškem sistemu in služijo točno določenemu namenu. Večina teh opisov se konča pri opisih delovanja in predstavitve rezultatov, medtem ko dejanskih implementacij, namenjenih širšim množicam, ni mogoče najti, ali pa so te tehnološko preveč zastarele in ozko usmerjene. Po analizi klasičnega modela kontekstnega strežnika lahko hitro opazimo, da se ta drži smernic več-nivojskih šibko sklopljenih arhitektur. Kontekstni strežnik je neodvisna entiteta, ki preko povezanih/nepovezanih standardov pridobiva podatke in rezultate posreduje končnim aplikacijam. Kot je bilo že prej omenjeno je cilj kontekstnega sistema zmanjšanje ročnega vnosa podatkov in uporabnikovih potrditev, s čimer težimo k večji avtomatizaciji procesov in proaktivnemu avtonomnemu pridobivanju podatkov o kontekstu. Da bi lahko prišli do takšnih kvalitetnih podatkov bi bilo potrebno takšnega uporabnika spremljati v ustreznih časovnih intervalih (idealno bi bilo, če bi ga lahko spremljali vseskozi) ter te dogodke tudi hitro in ustrezno klasificirati.

Zaradi teh razlogov so za zajem konteksta zelo primerne razne prenosne naprave, ki vseskozi spremljajo uporabnika v stanju mobilnosti. Večina teh naprav vsebuje tudi nabor osnovnih senzorjev za pridobivanje nekaterih podatkov o kontekstu, zaradi česar odpade potreba po zunanjih senzorjih, ki bi jih bilo potrebno konfigurirati ter namestiti za zajem konteksta. Kljub tem lastnostim pa so bile te naprave v preteklosti, gledano z vidika zmožnosti in razširljivosti programske opreme ter specifikacij strojne opreme, relativno zelo omejene. Navkljub tem razlogom pa so v relativno kratkem času razvili standardi brezžičnih in mobilnih komunikacij, ki so omogočili razvoj vseprisotnega računalništva (ang. *ubiquitous computing*). V tem principu je dostop do informacij in podatkov možen kjerkoli in kadarkoli in je delno ali v celoti podprt tudi z internetnim dostopom. Zaradi teh lastnosti so te naprave lahko majhne, prenosljive in poceni, služijo namreč le za komunikacijo med uporabnikom in nekim večjim sistemom, ki namesto njih v ozadju skrbi za pridobivanje in obdelavo podatkov ter končno predstavitvijo in dostavo podatkov do teh odjemalcev.

Neodvisno od razvoja programske in strojne opreme na osebnih računalnikih ter strežnikih pa se v zadnjem času pospešeno razvijejo standardi prenosnih naprav. Zmogljivosti teh naprav so

se občutno izboljšale z novimi izdajami integriranih sistemov na enem čipu (ang. *system on chip, SoC*), ki so postali zmogljivejši, vsebujejo več pomnilnika, podpirajo večnitnost procesov, a hkrati ne potrebujejo znatnejših virov energije od njihovih predhodnikov. Temu trendu so na različnih področjih sledili tudi operacijski sistemi za mobilne telefone in tablične računalnike, ki so te sisteme prilagodili trenutnim tehnologijam in za razvijalce v različnih obsegih izdali tudi ustrezne pakete za razvoj programske opreme (ang. *software development kit, SDK*).

Zaradi prej omenjenih specifikacij in lastnosti prenosnih naprav bi lahko del kontekstnega procesiranja opravili kar na napravi sami. To pomeni, da bi glavni del kontekstnega procesiranja opravljala kar naprava sama. Tak koncept bi tako delno ali v celoti odpravil potrebo po posebnem kontekstnemu strežniku. Tega je potrebno za lastno uporabo potrebno vzpostaviti na lastnem računalniku in skrbeti za njegovo nemoteno delovanje ali ga najeti pri enem od ponudnikov takšne storitve. Tudi podjetja morajo tak strežnik kupiti in ga sami vzdrževati ali pa za najem plačevati. Posebno vlogo pri tem konceptu ima tudi varnost in zasebnost podatkov, saj se ti v predlaganem konceptu ne pošiljajo skozi omrežje in se ne hranijo na podatkovnih nosilcih dislociranega kontekstnega strežnika. Na tak način je mogoče aplikacije zasnovati tako, da se izvajanje procesiranja izvaja tudi ob nepovezanosti v omrežje, uporabnik pa ima tako možnost hranjenja konteksta samo na njegovi napravi. Zaradi neuporabe uporabnikovega omrežja je tak način bolj neodvisen od morebitnih izpadov v tem omrežju, prav tako pa se v preko njega prenaša tudi manj podatkov.

Zelo zanimiv je članek z naslovom *Context-aware service engineering: A survey* [7], ki opisuje več različnih pristopov za implementacijo kontekstnega strežnika, ki sledijo nekaterim obstoječim raziskavam in predlaganim arhitekturam. Ti se med seboj ločijo glede na uporabljene programske jezike in njihove paradigme, postopke in načine razvoja teh implementacij ter tehnologij, ki povezujejo medsebojne module v celoto. V kolikor bi želeli večino opisanih konceptov v tem dokumentu enostavno prenesti na mobilno platformo bi prav gotovo naleteli na veliko problemov in neskladij.

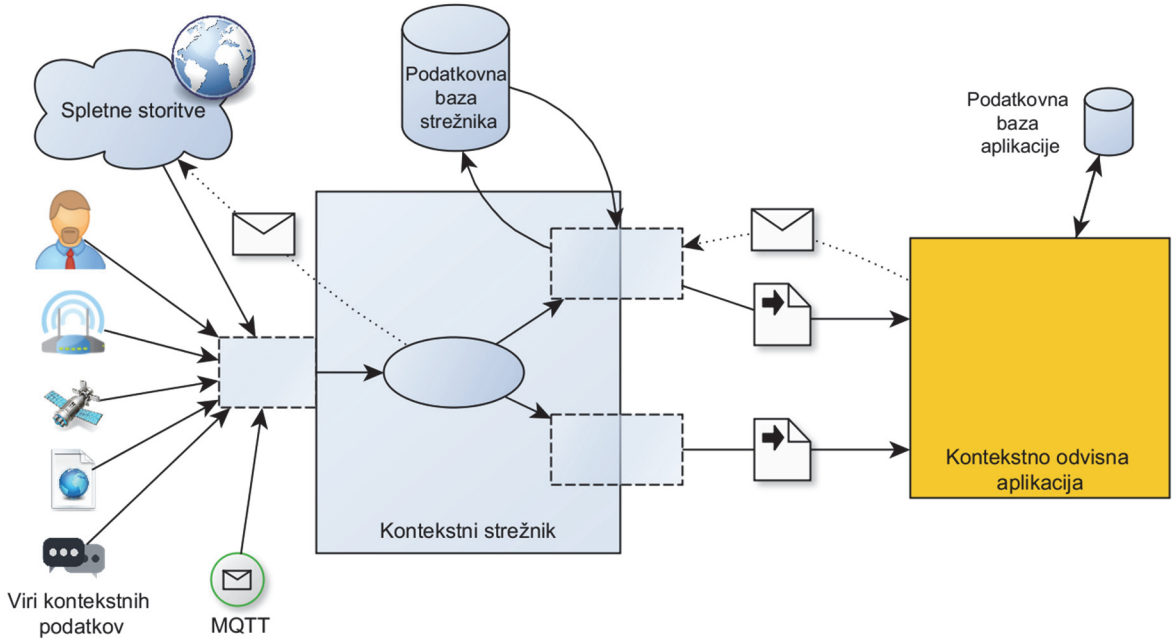
Zaradi navedenih razlogov se za implementacijo kontekstnega strežnika na napravi zdi najprimernejša oblika implementacija aplikacije oz. storitve v okviru ciljnega operacijskega sistema. V delu je tako predstavljen primer kontekstnega strežnika v operacijskem sistemu Android (v nadaljevanju Android), ki je predstavljen v poglavju 3.1. Ta nudi dobro podporo pri pridobivanju različnih podatkov iz vgrajenih senzorjev, nudi podporo pri pridobivanju podatkov preko spletnih programskih vmesnikov (ang. *Web API*) in ostalih storitev, omogoča enostavno deljenje podatkov med aplikacijami, izvajanje storitev in aplikacij v ozadju ter večnitnost znotraj posamezne aplikacije, zelo zanimiva pa je tudi bogata izbira knjižnic s strani neodvisnih razvijalcev. Nekatere od teh so namreč obstajale že v programskem jeziku Jave in so bile z njihove strani predelane in izpopolnjene za uporabo v jeziku Android.

5.2. Zasnova kontekstnega strežnika na mobilni napravi

Pregled zasnove kontekstnega strežnika na mobilni platformi Android začenjamo s pregledom osnovne strukture, ki poleg kontekstnega strežnika obravnava tudi kontekstno (ne)odvisno aplikacijo. Obe entiteti namreč sicer delujeta neodvisno druga od druge, vendar takšnega delovanja, vsaj s strani aplikacije, ne moremo obravnavati v smislu kontekstnega delovanja.

Da bi kontekstni strežnik lahko vseskozi spremljal kontekst bo moral biti implementiran kot storitev (ang. *service*), ki deluje v ozadju in glede na ta kontekst izvaja določene akcije. Ko je tak kontekstni strežnik prvič nameščen na ciljno napravo, s pomočjo zahtevkov do standardnih ponudnikov vsebin (ang. *Content Providers*) ugotovi tip in nabor standardnih senzorjev ter drugih ponudnikov informacij, ki jih naprava omogoča. V kolikor določenega senzorja ni na voljo, ga kontekstne aplikacije ne morejo uporabiti.

Kljub temu, da je definiran kot storitev, ima lasten uporabniški vmesnik, preko katerega lahko uporabnik eksplicitno vklopi ali izklopi delovanje strežnika, spremlja osnovne parametre delovanja spreminja nastavitve. V kolikor uporabnik iz katerikoli razloga izklopi delovanje kontekstnega strežnika, ciljne aplikacije ne bodo obveščene o novih podatkih. Kontekstno odvisne aplikacije morejo tak dogodek predvideti in svoje delovanje prilagoditi omenjeni situaciji.



Slika 10: Shema kontekstnega strežnika in kontekstno odvisne aplikacije. Nekateri vmesniki so na omenjeni shemi še neoznačeni, saj se lahko razlikujejo glede na uporabljene tehnologije.

Shema zasnove takšnega kontekstnega strežnika je prikazana na zgornji sliki (Slika 10). Teoretični opisi posameznih delov zajema konteksta in njihova uporaba je opisana spodaj, podrobnosti dejanske implementacije pa so opisane v naslednjem poglavju.

5.2.1. **Pridobivanje kontekstnih podatkov**

Že v uvodu je bilo omenjeno, da mora biti kontekstni strežnik zmožen sam poizvedovati po naboru že vgrajenih senzorjev in drugih virih podatkov. Pri tem v sistemu tudi poizve in zabeleži osnovne lastnosti senzorjev. Ti se hranijo v kontekstnem strežniku ter so na voljo morebitnim aplikacijam, ki bi želele omenjeni senzor uporabiti. Težava bi lahko namreč nastala ob tem, ko bi aplikacija želela dostopati do podatka senzorja, ki sploh ne bi bil na voljo, ali pa bi izbran senzor, glede na zahteve aplikacije, ponujal premajhno natančnost, frekvenco osveževanja podatkov ali razpon merljivega območja (Programska koda 2). Takšne primeri se morajo obravnavati že ob prvi inicializaciji aplikacije, ko se poda zahteva za uporabo vira podatka kontekstnega strežnika. Slednji mora v tem primeru prilagoditi zahtevane nastavitve ali vrniti odgovor neuspešne inicializacije v obliki napake oz. izjeme (ang. *Exception*). Odgovornost je nato na strani programerja, ki mora takšne dogodke tudi predvideti in v takih primerih prilagoditi tok izvajanja aplikacije.

```
D/Sensors(26123): {Sensor name="BMP280 Barometer", vendor="BOSCH",
version=1, type=6, maxRange=1100.0, resolution=0.009994507, power=0.004,
minDelay=33333}
```

Programska koda 2: Primer izpisa podatkov senzorja za merjenje zračnega pritiska, vgrajenega v mobilno napravo.

Za namene učinkovitega sledenja in zbiranja kontekstnih podatkov je potrebno tok teh podatkov najprej zajeti iz ciljnega vira ter jih, glede na različnih definirane mejne vrednosti, učinkovito speljati do ustreznega porabnika. Implementacija sistema, ki bo mogel takšne podatke zajemati v realnem času, bi bila sicer mogoča, težave pa bi nastale v primerih, ko bi želeli hkrati zajemati in obdelovati podatke iz več senzorjev naenkrat v realnem času. Ena od možnih rešitev tega bi bila implementacija medpomnilnika s pomočjo ene od standardnih struktur za hranjenje podatkov v glavnem pomnilniku, kot sta sklad (ang. *stack*) ali vrsta (ang. *queue*). To bi premostilo težave z medpomnjenjem, vendar bi s tem po obdelavi izgubili posredovani podatek. Rešitev tega problema je uporaba knjižnice za shranjevanje podatkov v podatkovni bazi SQLite, opisane v razdelku 3.2. Na takšen način lahko še vedno učinkovito obdelujemo meritve v realnem času in jih kot strukturirane podatke hranimo v podatkovni bazi. Posamezne dele meritev iz senzorja lahko namreč združimo v en kontekstni objekt, ki mu poleg teh dodamo še časovni žig, identifikacijsko oznake naprave, natančnost senzorja in ostale možne parametre.

Tak princip nam že v fazi zajema omogoča lažjo predstavitev stanja trenutno obravnavanega konteksta, enostavnejše kasnejše pridobivanje iskanega podatka in lažjo identifikacijo konteksta iz večje skupine povezanih naprav.

Na shemi predlaganega kontekstnega procesiranje vidimo tudi možnost prejema podatkov iz določenih spletnih storitev, ki jih je mogoče uporabljati v primeru uporabnikove povezanosti v internetno omrežje. Tudi spletne storitve so lahko bogat vir kontekstnih podatkov, ki jih lahko kombiniramo s lastnimi kontekstnimi podatki. Nekateri izmed teh kontekstnih podatkov so lahko tudi ključni elementi za prejem novih kontekstnih podatkov. Kot primer lahko navedemo zahtevo za pridobitev podatkov o vremenu v naši okolici, ki jo pošljemo spletni storitvi za opazovanje vremena. Ključni člen za uspešno generiranje zahtevka in prejem ustrezne informacije je tukaj podatek o trenutni lokaciji, ki smo jo preko kontekstnega strežnika prejeli še pred ustvarjanjem zahtevka.

Dodatno lahko na omenjeni shemi opazimo tudi prejem sporočil od določenega spletnega vira s pomočjo protokola MQTT (ang. *Message Queue Telemetry Transport*). Omenjeni protokol je bil v začetku naših zgodnjih načrtih kontekstne arhitekture na mobilni platformi namenjen prenosu poslovnih pravil in ostalih konfiguracij za delovanje kontekstnega strežnika. Ker se je že v zgodnji fazi implementacije izkazalo, da bomo princip izvajanja kontekstnih pravil zelo težko implementirali na mobilni platformi, smo ta postopek kasneje opustili. Ker je bil ta postopek v veliki celoti že implementiran, smo ta postopek ohranili v sami shemi in mu v končni implementaciji tudi našli koristen namen.

5.2.2. **Priprava podatkov in njihovo hranjenje**

Predpriprava podatkov je postopek, ki ga izvedemo na zbranih podatkih po postopku zajema podatkov. Lahko se izvede sočasno z zajemom, kjer vsak surov podatek spremenimo v strukturirano ali uporabniku bolj razumljivo obliko, lahko pa se izvede pred samim procesiranjem podatkov. V slednjem primeru se zelo pogosto uporablja pri postopkih rudarjenja po podatkih in strojnem učenja ter vključuje različne stopnje oz. korake:

- Čiščenje podatkov – Podatki se analizirajo tako, da se ugotovijo možne manjkajoče ali neobstoječe vrednosti. Pri neobstoječih vrednostih vnaprej predvidevamo, da bo ta v dotičnem primeru neizmerljiva ali ne relevantna za procesiranja, medtem ko lahko manjkajoča vrednost vpliva na potek in rezultate procesiranja podatkov. Metode za rešitev te situacije so različne, od najbolj enostavnih, ko tak podatek zavržemo ali ga nadomestimo s privzeto vrednostjo, do različno kompleksnih metod. Dodatno v

nekaterih primerih analiziramo tudi morebitne izstopajoče vrednosti, ki lahko predstavljajo potencialno napako v merjenju.

- Integracija podatkov – Obravnava integracijo oz. združevanje podatkov v enotno strukturo, nad katero se izvajajo operacije. Podatki namreč po navadi prihajajo iz različnih virov, kjer so ti lahko predstavljeni v različnih strukturah in oblikah. Primeri takšnih zapisov so: tekstovni zapisi, podatkovne zbirke, objekti v serializirani obliki, strukturirani dokumenti v različnih zapisih označevalnih jezikov in ostali. Za potrebe končnega procesiranja te podatke združimo v enoten zapis podatkov, ki jo lahko imenujemo tudi tabela. V tej točki posamezne primerke podatke začnemo obravnavati kot celoto, zato lahko pri tem pride do različnih nekonsistentnosti, kot je redundanca v podatkih različnih virov, podvajanje zapisov, različni zapisi in enote vrednosti pri posameznih virih podatkov ter problem enolične identifikacije entitet med različnimi podatkovnimi zbirkami.
- Redukcija podatkov – Je korak, ki obsega različne metode za definicijo nove predstavitve obstoječe tabele podatkov v obliki, kjer zmanjšamo količino teh podatkov a vseeno ohranimo njihovo integriteto. Uporablja se v postopku predprocesiranja in glede na izbran algoritem, zaradi manjšega nabora vhodnih podatkov, vpliva na hitrost in porabo virov pri tem zagotavlja skoraj enake rezultate in kvaliteto procesiranja kot osnovna predstavitev podatkov. V ta namen se uporabljajo različne metode, kot so, iz podatkovnih baz že predhodno znani, postopki agregacija celotne strukture podatkov na določeni ravni – kuboidu (ang. *data cube aggregation*), zmanjšanje dimenzionalnosti podatkov z odstranjevanjem za procesiranje nerelevantnih ali manj relevantnih atributov, redukcija možnih vrednosti posameznih (numeričnih) atributov, kompresija podatkov in ostali. Proces redukcije podatkov se lahko uporablja tudi za namene shranjevanje rezultatov procesiranja v podatkovno bazo in namene predstavitve rezultatov.
- Diskretizacija podatkov – Posamezni atributi v tabeli podatkov so lahko predstavljeni z različnimi podatkovnimi tipi, ki označujejo njegovo vrednost. V primeru atributov z nezveznimi kategoričnimi vrednostmi je vrednost tega atributa natančno določena z naborom možnih stanj, medtem ko lahko pri numeričnih zveznih atributih predstavimo katerikoli vrednost. Tem atributom v postopku diskretizacije zmanjšamo nabor možnih vrednosti tako, da jih razdelimo v intervale. Posamezne oznake teh intervalov se nato uporabijo za predstavitev vrednosti tega atributa. S tem postopkom dosežemo, da so podatki bolj preprosto predstavljeni in jih je pri nekaterih postopkih obdelave podatkov veliko lažje obravnavati kot običajne zvezne atribute. Določene metode diskretizacije podatkov se uporabljajo tudi pri kategoričnimi atributih s končnim, a vseeno zelo obširnim, naborom možnih vrednosti. V tem primeru se uporabljajo metode za kategorizacijo posameznih vrednosti v skupine, s čimer imamo na višjem nivoju manjši nabor kategoričnih vrednosti.

Struktura kontekstnega strežnika na mobilni napravi ima pri postopku obdelave podatke deljeno vlogo s ciljnimi aplikacijami. Večina priprave in morebitne obdelave podatkov se namreč dogaja v sami aplikaciji, medtem ko kontekstni strežnik večinoma operira z enostavnimi podatki. Primer takšnega procesiranja bomo podrobno preučili v poglavju 6.4, kjer bomo predstavili delovanje kontekstno odvisne aplikacije in podrobnosti njenega kontekstnega procesiranja.

Na strani samega kontekstnega strežnika kakšnih večjih postopkov predprocesiranja ne bomo izvajali, ti postopki bodo večinoma usmerjeni v razčlenjevanje prejetih podatkov in njihovo agregacijo za hranjenje v podatkovni bazi. Dodatno se bodo na kontekstnem strežniku ustvarjale zahteve za dostop do spletnih storitev, katerih odgovori se bodo ravno tako hranili v bazi. Ostale operacije so kvečjemu namenjene podpori delovanja samega kontekstnega strežnika in pa postopkov deljenju podatkov aplikacijam, ki sledijo v naslednjem poglavju.

Pri postopkih hranjenja podatkov bomo uporabili standardne razrede operacijskega sistema za hranjenje v lastni podatkovni bazi. Ta omogoča strukturirano hranjenje podatkov in definiranje lastnih kontrol za dostop do teh podatkov. Po pregledu nekaterih definicij kontekstnih strežnikov smo tudi sami preučevali možnosti hranjenja podatkov v obliki ontologij. Takšna uporaba bi lahko prinesla boljšo povezljivost podatkov in iskanje sorodnosti med njimi. Vseeno so ontologije bolj primerne za zasnovo končne aplikacije, kjer bi te lahko bolje opisale pomen svojih podatkov in njihove medsebojne relacije.

5.2.3. **Deljenje podatkov ciljnim aplikacijah**

Deljenje podatkov končnim kontekstnim aplikacijam v osnovi poteka preko t.i. ponudnikov virov, vendar je celoten postopek pridobivanje podatkov lahko različen. V kolikor v kontekstno odvisnem sistemu za določen vir ni nobenega aktivnega naročnika, se proces zajema kontekstnih podatkov iz tega vira ne bo izvajal. Ob definiranju nove kontekstno odvisne aplikacije ima programer tako na voljo več različnih virov kontekstnih podatkov, ki jih lahko uporabi. Pri tem mora vir ustrezno nasloviti ter vnesti morebitne spremembe privzetih vrednosti za zajem. V tem trenutku lahko ubere eno dveh možnosti za zajem podatkov.

1. Uporaba poslušalca – Preko poslušalca se aplikacija lahko naroči na spremembe v izbranem kontekstu ter ob teh spremembah udejanji določeno akcijo. Dober primer tega je sprememba lokacije. Kontekstni strežnik bo v tem primeru opazoval za spremembami v lokaciji in ob dovolj visoki natančnosti opozoril vse aplikacije, ki imajo registrirane poslušalce. V tem trenutku bo aplikacija iz baze kontekstnih podatkov pridobila ustrezne informacije in jih uporabila. V kolikor aplikaciji ni

potrebno vseskozi spremljati stanja tega vira, ga mora na zahtevo ali ob odhodu iz aplikacije ustrezno deaktivirati.

2. Ročno pridobivanje podatkov – Kontekstno odvisna aplikacija lahko ročno, na svojo zahtevo pridobiva podatke iz ustrezne tabele v bazi kontekstnega strežnika, v kateri so podatki kontekstnega strežnika. Podatke o tabeli aplikacija pridobi preko ustreznega ponudnika podatkovne baze za določen vir in s pomočjo poizvedovalca pošlje zahtevo. V odgovor dobi kazalec na polja z zahtevano vsebino, ki jo nato lahko pridobi.

Zgoraj opisana postopka sta temelja za pridobivanje podatkov iz procesa kontekstnega strežnika. Zaradi uporabe podatkovne baze SQLite, bo urejena tudi sočasnost bralnih dostopov in pridobivanje večjega števila podatkov naenkrat, tudi iz strani več različnih aplikacij. Podatke bi bilo namreč možno pošiljati tudi preko sporočil ob spremembah v kontekstu, vendar ta metoda za več hkratnih poslanih sporočil ne zagotavlja njihovega prejema saj obstaja možnost, da lahko posamezna sporočila tudi izgubijo.

5.2.4. **Procesiranje podatkov**

Veliko pozornosti pri implementaciji je bilo namenjeno predvsem v definiranje ideje, kako naj bi mobilni kontekstni strežnik procesiral zbrane podatke. V sklopu je bilo tako že omenjeno, da smo se skušali pri tem zgledovati pri obstoječih idejah in opisih delovanja strežniških kontekstnih strojev. Večina teh se med seboj loči glede na namen zbiranja in procesiranje kontekstnih podatkov in zaradi tega uporabljajo zelo različne načine in metode.

Pri določitvi lastne arhitekture kontekstnega strežnika smo tako preučili možnost uporabe ene od tehnik definiranja poslovnih procesov z uporabo poslovnih pravil. V ta namen se uporabljajo različne sistemi in programske aplikacije, ki jih imenujejo sistemi za izvajanje poslovnih pravil (ang. *Business Rules Engine, BRE*). Takšni sistemi, večinoma so to programske komponente, so zmožne s pomočjo enostavno določljivih poslovnih pravil, ki se v postopku obdelave pretvorijo v določeno izvršljivo kodo, izvesti neke določene akcije nad nekimi podatki. Poslovna pravila so enostavno določljiva in razumljiva tudi običajnim (poslovnim) uporabnikom. Ta pravila se lahko definirajo ročno, še enostavneje pa jih je modelirati s pomočjo okolja, vključenega v sistem za upravljanje poslovnih pravil (ang. *Business Rules Management System, BRMS*). Ti se uporabljajo za definiranje, upravljanje, izvajanje in nadziranje celotne poslovne logike, ki se uporablja v kompleksnih sistemih. Z naprednimi zmožnostmi takšnega okolja lahko definiranje kompleksnih poslovnih pravil pretvorimo v enostavnejše in bolj pregledno modeliranje procesnega toka, ki mu dodamo vhode, iz nabora gradnikov izgradimo in testiramo procesni tok, prilagodimo njegovo izvajanje glede na različne notranje iz zunanje dogodke in na koncu določimo njegov izhod – rezultat končnega procesiranja.

Po pregledu lastnosti omenjenih sistemov smo preiskali možne implementacije, ki bi jih lahko uporabili na mobilni napravi s sistemom Android. Ideja je bila, da bi lahko poslovna pravila definirali na osebnem računalniku in jih nato prenesli na mobilni kontekstni strežnik. Mobilni strežnik bi ta pravila nato tolmačil ter izvajal določene akcije nad kontekstnimi podatki, ki bi jih pridobival preko lastnih senzorjev. Še bolj zanimivo in poenostavljeno bi bilo modeliranje celotnega poslovnega procesa preko BRMS sistema in nato prenos tega (serializiranega) modela na mobilno napravo. Ker po obširnem iskanju nismo našli ustrezne rešitve za sistem BRE, ki bi se lahko izvajal na mobilni napravi, smo začeli preučevati možne implementacije sistemov, ki bazirajo na programskem jeziku Java in so odprtokodne. Pri teh bi lahko ključne javanske razrede vgradili v kodo projekta kontekstnega strežnika, jih preko privzetega prevajalnika pretvorili in izvajali na mobilni napravi. Osnovnim zahtevam sta ustrezala BRMS sistema IBM ILOG JRules in Drools. Žal pa smo pri implementaciji ključnih razredov obeh sistemov naleteli na probleme pri prevajanju razredov. Ti so namreč vsebovali določene razrede in razčlenjevalnike, kateri na sistemu Android zaradi različnih razlogov niso podprti. Podobno so prej preizkušali tudi že nekateri ostali uporabniki tega koncepta, ki jim je uspelo le osnovno tolmačenje poslovnih pravil in njihovo definiranje v tekstovni obliki na mobilni napravi [8]. Zaradi tega razloga smo ocenili, da bi implementacija takšnega modula v mobilnem kontekstnem strežniku težko opravila vse osnovne naloge pri procesiranju surovih podatkov. V primerjavi z izvajanjem programske kode na nivoju operacijskega sistema bi takšna implementacija, na račun prevajanja poslovnih pravil ali modelov, prav gotovo imela tudi slabše zmogljivosti in večjo porabo virov. Zaradi teh razlogov smo se odločili, da procesiranja v kontekstno odvisni mobilni arhitekturi zasnujemo na nivoju programske kode operacijskega sistema z uporabo možnih vtičnikov.

Takšna implementacija mobilnega strežnika v primerjavi z vlogo klasičnega kontekstnega strežnika bi potrebovala novo razmejitev vlog pri procesiranju podatkov. Medtem ko klasični kontekstni strežnik poleg distribuiranja podatkov in informacij skrbi predvsem za celoten proces procesiranja podatkov, je pri mobilnem kontekstnem strežniku vloga procesiranja deljena med strežnikom in aplikacijo. Takšno delitev zagovarja kar nekaj konkretnih dejstev:

- Kontekstni strežnik in vse kontekstno odvisne aplikacije se izvajajo na isti napravi. V primeru uporabe kontekstnega strežnika s strani ene aplikacije ni razlik med tem, ali se določeno kontekstno procesiranja izvede v procesu kontekstnega strežnika ali v delu mobilne aplikacije.
- Vsaka aplikacija ali proces ima v Androidu odmerjen točno določen del pomnilnika (ang. *heap space*), ki ga lahko uporablja. V preteklosti je bil ta določen glede na inačico operacijskega sistema Android in se ga zato ni dalo spreminjati. V primeru, ko bi večino procesiranja izvajali v procesu kontekstnega strežnika, bi ta, v primeru več sočasnih zahtev za procesiranja, lahko zasedel ves razpoložljiv prostor. V tem trenutku bi v sistemu začel izvajati proces sproščanja pomnilnika (ang. *garbage collection*), ki

močno zmanjša zmogljivosti omenjenega procesa in vpliva na delovanje celotnega sistema. V najslabšem primeru bi lahko operacijski sistem trenutni proces tudi zaustavil, s čimer bi se lahko začasno ustavilo tudi delovanje kontekstnega strežnika.

- Večina kontekstno odvisnih aplikacij za svoje delovanje potrebuje podatke, ki jih hrani v lastni podatkovni bazi. To pomeni, da bi se morali ti podatki za namene skupnega procesiranja prenašati v proces kontekstnega strežnika. Prenos teh informacij in njihovo prejemanje je spet dodatna obremenitev za napravo, prav tako pa bi teoretično lahko bila ogrožena zasebnost teh podatkov. Večini avtorjev aplikacij zagotovo ne bi bilo po godu, da bi ti podatki lahko odtekali iz aplikacije.
- V operacijskem sistemu Android imajo uporabniki natančen vpogled v porabo energije, pomnilnika in trenutne zmogljivosti posameznih komponent. Za lažjo skupno predstavitev jih, v primeru porabe energije, Android loči glede na porabo posameznih aplikacij in storitev. Če bi se večino procesiranja izvajalo v storitvi kontekstnega strežnika, bi poraba takšnega strežnika občutno prednjačila pred porabo energije aplikacij, za katere strežnik procesira podatke. To bi lahko zmotilo nekatere uporabnike, ki se trudijo za čim boljšo avtonomijo svoje naprave in bi lahko v tem primeru storitev kontekstnega strežnika odstranili.

Seveda pa se na kontekstnem strežniku izvedejo osnovne obdelave surovih podatkov, ki jih predelajo, združijo ali kako drugače obdelajo v določeno obliko informacije, s katero lahko nato oskrbijo aplikacije. Nekatere izmed teh operacij smo omenili že v poglavju 5.2.2. Ostali del kontekstnega procesiranja se nato po predlaganem scenariju izvaja v sami aplikaciji in je prilagojen namembnosti aplikacije, njenim nastavitvam in podatkom, ki jih uporabljajo.

5.3. Lastnosti mobilnih kontekstno odvisnih aplikacij

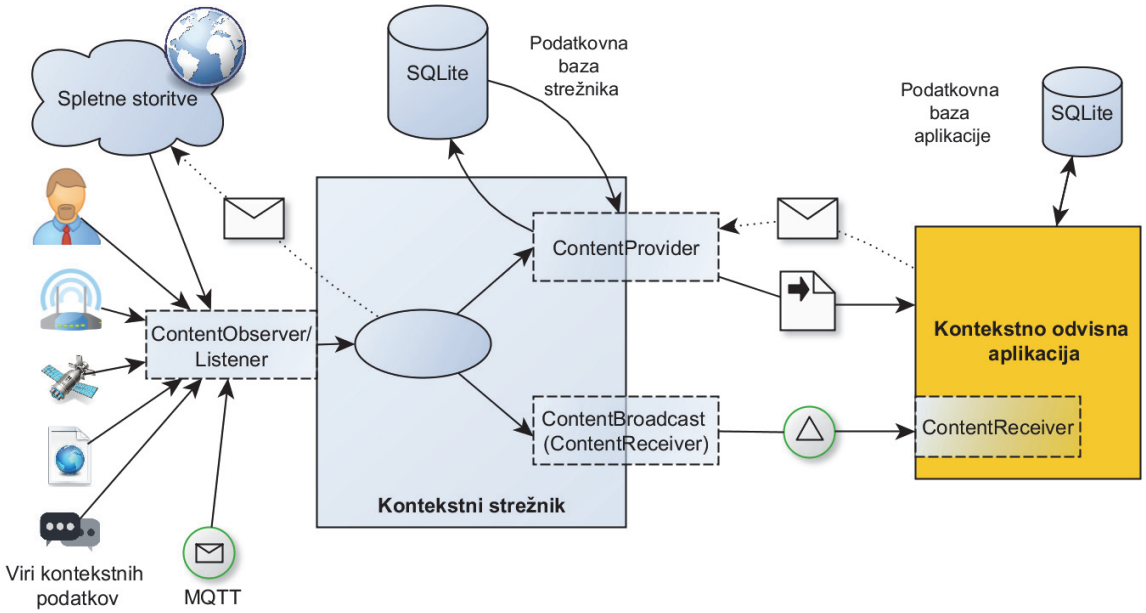
Skozi opise principov delovanja posameznih delov kontekstnega strežnika je bilo veliko povedanega tudi o lastnostih pripadajočih kontekstno odvisnih aplikacij. Prav gotovo se ta na prvi ogled ne bo razlikovala od preostalih aplikacij ampak se njena vrednost pokaže šele z določenim časom uporabe. Med razvojem arhitekture smo v namene testiranja kontekstnega strežnika izvedli več manjših aplikacij, ki so uporabljale različne tipe konteksta. Primer ene od takih aplikacij je aplikacija za ugotavljanje, ali je uporabnik v dvigalu. Takšno aplikacijo bi se lahko implementiralo s pomočjo merilnika pospeška, v naši poskusni implementaciji pa smo uporabili barometer in glede na vrednosti in spremembe zračnega pritiska ugotavljali, ali se uporabnik spušča ali dviga. Ker smo ob razvoju kontekstnega strežnika uporabljali mobilno napravo z vgrajenim nizkoenergijskim senzorjem za zaznavo korakov, smo pri razvoju implementirali osnovno mobilno aplikacijo za zaznavo korakov in štetje le teh. V tej aplikaciji

smo tudi prvotno uporabili tehnike obveščanja o obstoju novih podatkov, ki jih je pošiljal senzor za zaznavo korakov. Šele ob tem dogodku smo namreč aktivirali števec korakov, ki je od prej omenjenega veliko bolj energijsko potraten.

Ob koncu implementacije kontekstnega strežnika pa smo za njegovo delovanje zasnovali osnovno idejo o aplikaciji, ki bi za svoje delovanje lahko izkoristila podatke kontekstnega strežnika in se glede na njegove podatke bolj prilagodila uporabnikovemu kontekstu. Da uporabniku ne bi bilo potrebno ročno vnašati nekaterih nastavitev ali mu postavljati ponavljajočih vprašanj o njegovih željah, smo definirali aplikacijo, ki uporabnikove iskalne vnose primerja s svojimi podatke ter mu v primeru ugodnih zadetkov samodejno priporoči tip restavracije, ki bi se uporabniku lahko zdela interesantna. V kolikor se iskane besede ne ujemajo z dovolj veliko natančnostjo, mu aplikacija morebitnih zadetkov ne prikaže. Dodatno mu aplikacijo pokaže tiste rezultate, ki so glede na njegovo lokacijo najbližji. V naslednjem poglavju implementacijo takšne aplikacije tudi podrobno opišemo.

6. OPIS IMPLEMENTACIJE PREDLAGANE ARHITEKTURE KONTEKSTNEGA STREŽNIKA

Idejo predlagane arhitekture smo v tem poglavju prenesli na mobilno platformo Android in opisali njene osnovne gradnike ter uporabljene tehnologije. Tako so podrobno opisane vse tri osnovne lastnosti od zajema do hranjenja in procesiranja konteksta ter povezovanje s kontekstno odvisno aplikacijo. Pri implementaciji smo uporabili temeljne gradnike sistema Android, ki so v tem okolju že standardizirani. Pri določenih rešitvah smo sicer morali uporabiti tudi nekatere primerke zunanjih knjižnic, ki so večinoma odprtokodne.



Slika 11: Shema implementacije kontekstnega strežnika v sistemu Android.

Na zgornji shemi lahko vidimo posamezne razrede implementirane rešitve in okviren tok zahtev ter podatkov. Podobno kot smo v prejšnjem poglavju opisali idejo arhitekture, je tudi v tem poglavju pregled implementacije strežnika razdeljen na več delov. Na koncu je opisan tudi razvoj kontekstno odvisne aplikacije, ki je bila predlagana v poglavju 5.3.

6.1. Zbiranje podatkov

Začetek implementacije zajema podatkov iz mobilne naprave se začne z definiranjem dovoljenj za dostop do nekaterih delov sistema Android, ki bi lahko dostopali do uporabnikovih datotek, trenutno izvajajočih aplikacij, podatkov o uporabnikovih omrežjih, vplivali na delovanje naprave in ostale. Ta dovoljenja se morajo definirati v posebni datoteki imenovani *AndroidManifest* in jih tekom izvajanja aplikacije ni mogoče spreminjati. Uporabnik ima tako ob instalaciji natančen vpogled v vsa dovoljenja, ki jih aplikacija potrebuje. Da bi lahko kontekstni strežnik zbral čim več različnih podatkov ter jih posredoval aplikacijam, je bilo potrebno v prej omenjeni datoteki vnesti več kot 25 dovoljenj za dostop do različnih podatkov in delov sistema.

Zajem podatkov običajno na napravi poteka preko klicev do osrednje knjižnice za dostop do senzorjev in preko različnih podanih parametrov omogoča pridobivanje podatkov neposredno od senzorja v obliki neobdelanih podatkov (ang. *raw data*). Ker se tipi senzorjev in njihove specifikacije od naprave do naprave razlikujejo, je mogoče v Androidu ob inicializaciji naprave pridobiti osnovne podatke o senzorju, kot so: ime senzorja, njegov tip (v obliki interne oznake), proizvajalec, maksimalno merljivo območje in ločljivost senzorja - predstavljena z osnovno enoto merjenja, potrebna moč za delovanja senzorja v mA, ter minimalni časovni interval med dvema zajemoma novega podatka. Končna implementacija zajema podatkov iz senzorjev je bila realizirana s pomočjo odprtokodne programske knjižnice Aware. Njena osnovna različica je sicer namenjena zajemu podatkov na napravah Android, shranjevanju podatkov v lokalno podatkovno bazo ter občasno sinhronizacijo lokalne baze z glavnim strežnikom in njegovo podatkovno bazo. Ti podatki so nato na voljo raziskovalcem, ki lahko na teh podatkih opravljajo različne raziskave.

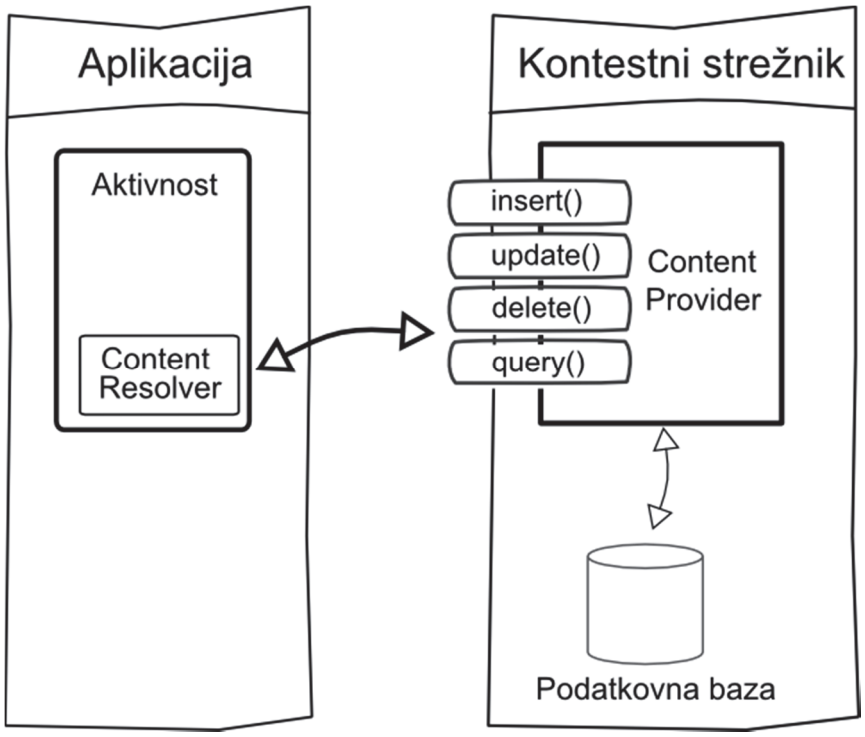
V naši implementaciji je bil zajem podatkov odvisen od tipa vira podatkov. Pri podatkih iz senzorjev, ki jih nadzoruje razred *SensorManager*, zadostuje že razširitev osnovnega razreda z vmesnikom *SensorEventListener* in definiranje razreda poslušalca, ki spremlja spremembe v omenjenem senzorju. Pri nekaterih preostalih virih so bili uporabljeni poslušalci drugih tipov, medtem ko smo morali pri preostalih virih podatkov uporabiti drugačen način. Nekateri izmed takšnih virov hrani svoje podatke v podatkovni bazi (zgodovina in zaznamki brskalnika, instalirane aplikacije), zato smo na tabele teh podatkov registrirali poslušalca iz razreda *ContentObserver*, ki reagira ob spremembah v teh tabelah. V kolikor pri določenih virih sistem teh podatkov ni shranjeval avtomatično in jih zato ni spremljal, smo podatke o virih iz pripadajočih modulov periodično pridobivali preko ustreznih upravljalcev ter jih kasneje sami hranili v lastnem delu podatkovne baze.

Pri implementaciji omenjenih metod smo imeli težave predvsem s prenehanjem zajema podatkov v primerih, ko je naprava prehajala v stanje mirovanja. Razlog je v tem, da poleg izključitve zaslona tudi procesor sčasoma preide v stanje mirovanje. Ker večina vgrajenih senzorjev ne vsebuje lastnih koprocesorjev, ki bi lahko sami kontrolirali delovanje, je potrebno glavni procesor ohraniti pri delovanju tudi v primerih, ko želimo telefon postaviti v mirovanje. To se doseže s pomočjo standardnega mehanizma *WakeLock* razreda *PowerManager*, ki poskrbi, da se določeni aplikaciji ali storitvi omogoči delovanje procesorja tudi v fazi mirovanja. Takšen pristop je v običajnih problemih, ko bi želeli aplikacijo pustiti da vseskozi deluje v ozadju, odsvetovan, saj procesor med mirovanjem porabi manj energije.

6.2. Hranjenje in posredovanje podatkov

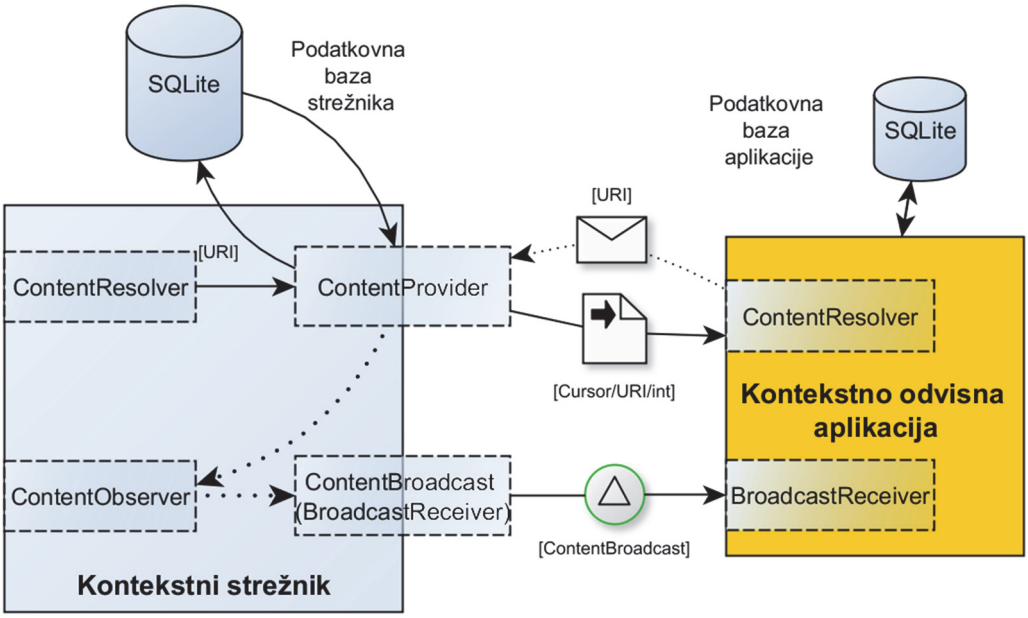
Postopek hranjenja podatkov smo delno omenili že v prejšnjem poglavju, saj je bilo potrebno iz določenih tipov virov podatkov implementirati periodičen zajem podatkov, njihovo osnovno obdelavo in agregacijo v poenoteno obliko ter shranjevanje v bazo kontekstnega strežnika. Shranjevanje podatkov v bazo je bilo podprto z standardnim razredom *ContentProvider*, ki ima več različnih funkcij in prednosti v primerjavi z ročnim definiranjem SQL sintakse ter njenega izvajanja. Zmožnosti omenjenega razreda so naslednje:

- Posamična aplikacija lahko preko osnovnih ukazov ustvari in dostopa le do lastne podatkovne baze. Z uporabo razreda *ContentProvider* lahko določena aplikacija oz. storitev, v našem primeru je to kontekstni strežnik, s tem razredom izpostavi posamezne tabele v svojem delu podatkovne baze in omogoči tudi drugim aplikacijam dostop do teh podatkov (Slika 12).
- Razred se uporablja za definicijo posameznih tabel ter njihovih stolpcev skupaj z njihovimi podatkovnimi tipi, začetnimi vrednostmi, definicijo različnih tipov ključev in ostalih možnostih. Omenjene definicije se nato uporabijo ob inicialni ustvaritvi tabel.
- V omenjenem razredu lahko definiramo vse osnovne CRUD operacije (*Create, Read, Update, Delete*) za poizvedovanje, vstavljanje, spreminjanje in brisanje podatkov.
- V razredu je potrebno posameznim tabelam določiti oznako oziroma pot, ki je v obliki enotnega označevalnika vira (ang. *URI, Uniform resource identifier*). Ta se določi glede na oznako aplikacije (*authority*) in ciljne tabele (*path*) in v primeru omenjenega razreda vključuje predpono "content://".
- Da bi omenjene aplikacije lahko dostopale do teh podatkov, morajo implementirati razred *ContentResolver*. Poizvedovanje poteka preko instance tega razreda, ki mu moramo vnesti zgoraj omenjeni URI, zahtevano operacijo ter glede na slednjo tudi morebitne podatke za uspešno izvedbo te operacije.



Slika 12: Prikaz dostopa do podatkov iz aplikacije do podatkovne baze kontekstnega strežnika preko razreda *ContentProvider*.

Podroben opis hranjenja podatkov skupaj s uporabljenim razredom *ContentProvider* je podrobneje opisan tudi zato, ker ima tudi pri posredovanju podatkov veliko vlogo. Na spodnji sliki lahko vidimo, da ima kontekstni strežnik več opcij za posredovanje podatkov. V prvem načinu ima inicialno vlogo prav kontekstno odvisna aplikacija, ki na lastno ali uporabnikovo zahtevo uporabi lasten razred *ContentResover* in preko njega pošlje ustrezno zahtevo s pripadajočim označevalnikom vira URI in morebitne druge podatke (koda selekcij, projekcij, razvrščanja in ostalih operatorjev v primeru poizvedovanju ali brisanju podatkov, objekt *ContentValues* z novimi vrednostmi pri vnosu in morebitni spremembi podatkov). Omenjeni razred nato sam dostopa do omenjene tabele, izvede ustrezno operacijo in glede na zahtevani podatek vrne ustrezen odgovor. V primeru poizvedovanja po tabeli je to kazalec (ang. *Cursor*), ki kaže za ciljni nabor podatkov, omogoča iteracijo po teh podatkih ter njihovo pridobivanje.



Slika 13: Podrobnejši pregled shranjevanja in posredovanja podatkov.

Drugi način pridobivanja podatkov je omenjeno naročanje na spremembe v kontekstu določena vira podatkov. Omenjeno delovanje je mogoče doseči s pomočjo razredov *ContentObserver* in *BroadcastReceiver*. Prvi omenjeni razred je namenjen definiranju posebnih poslušalcev, ki opazujejo spremembe v določeni podatkovni bazi in ob tem izvedejo določeno akcijo. Eden od razlogov takšne uporabe razreda je ta, da lahko s tem enostavno povežemo prvi del zajema podatkov, ko se podatki iz senzorjev zajamejo in vnesejo v bazo. V kolikor bi skušali ob teh dogodkih hkrati shraniti podatek v bazo in obenem obvestiti aplikacijo o obstoju novih podatkov, bi lahko zahteva aplikacija o prejemu podatkov iz kontekstnega strežnika prehitela zaključek operacije vnašanja teh podatkov, kar bi lahko vodilo v prejem neažurnih podatkov.

Ko razred *ContentObserver* zazna obstoj novih podatkov preko razreda *BroadcastReceiver* ustvari in pošlje ustrezno sporočilo aplikacije, kjer vključi identifikacijo vira konteksta in morebitne dodatne podatke. Aplikacija mora le uporabiti ustrezen razred za prejem teh sporočil ter se na ta sporočila naročiti. Naročanje na obvestila poteka preko Androidovega manifesta ali se definira dinamično v aplikaciji. Ker sporočanje poteka preko namer (ang. *Intent*), je potrebno pri registraciji naročanja definirati ustrezen filter z ustrezno oznako namere, ki jo želimo prejeti. Ob prejemu takšne namere se nato v metodi *onReceive()* izvede ustrezna operacija. Prednost uporabe takšnega razreda je predvsem v tem, da lahko z njegovo pomočjo prejemamo obvestila o novih podatkih, te podatke nato preko že omenjenega *ContentResolverja* preberemo in v primeru ustreznih pogojev zaženemo glavno aktivnost aplikacije, ki ponavadi vključuje uporabniški vmesnik. To se stori s pomočjo ustvarjanje nove namere, ki ji je potrebno nastavitvi ustrezne zastavice in dodati morebitne podatke. Primer takšne je

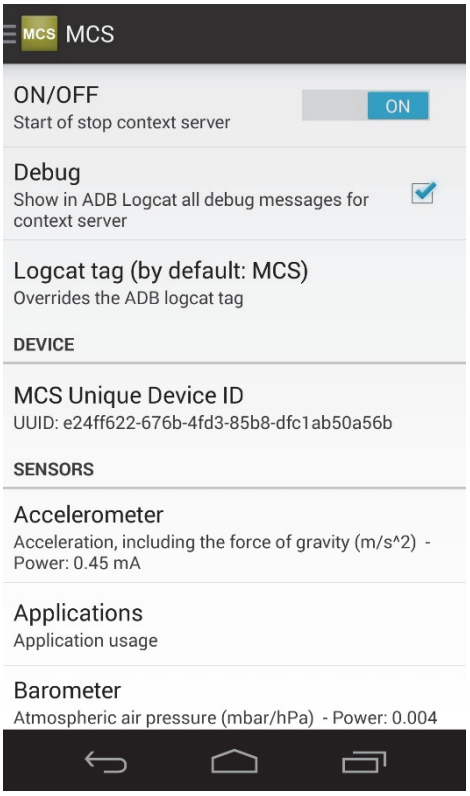
FLAG_ACTIVITY_NEW_TASK. Poleg tega lahko v namero vstavimo tudi dodatne podatke. Primer uporabe je namera, ki zažene aplikacije z uporabniškim vmesnikom s štirimi zavihki, ob zagonu takšne namere se bo ob prejemu podatkov kontekstnega strežnika aplikacija samodejno zagnala ter glede na prejete podatke v nameri uporabniku prikazala uporabniški vmesnik tretjega zavihka.

6.3. Procesiranje podatkov

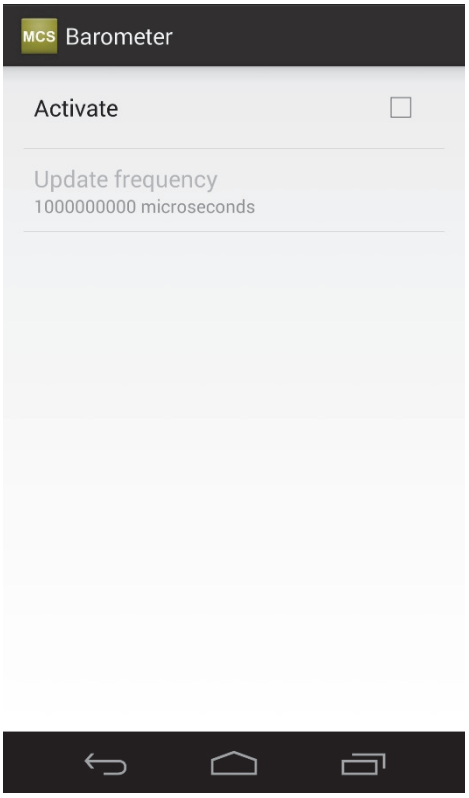
V prejšnjem poglavju smo opisali idejo o delitvi kontekstnega procesiranja med kontekstnim strežnikom in aplikacijo ter razlogih za omenjeno delitev. To idejo smo uporabili tudi pri naši implementaciji kontekstnega strežnika. Zaradi tega v naslednjem razdelku tudi opišemo kontekstno procesiranja v zato ustvarjeni aplikaciji ter tehnike pridobivanja podatkov od kontekstnega strežnika.

Seveda se tudi v kontekstnem strežniku izvaja določen del procesiranja podatkov, vendar je ta pri virih manj kompleksnih virih konteksta dokaj osnoven in obsega različne funkcije za predobdelavo podatkov, ki so namenjeni shranjevanju v podatkovno bazo. V kolikor bi več različnih aplikacij te podatke procesiralo na tak način, da bi se lahko na osnovi teh procesov ustvarilo splošno generalizacijo podatkov in podatkovnega toka, bi bilo bolj primerno te podatke procesirati v samem kontekstnem strežniku ter informacije ponuditi izbranemu naboru aplikacij. Izbrano razširitev bi si lahko uporabnik kontekstnega strežnika ustvaril sam, ali pa bila ta razširitev že vgrajena v same metode kontekstnega strežnika.

Uporabniški vmesnik kontekstnega strežnika je namenjen osnovnemu pregledu nastavitev saj se večina storitev kontekstnega strežnika opravi v ozadju. Preko omenjenih nastavitev lahko trajno izklopimo vse njegove module ali jih vklopimo (Slika 14). Preko vmesnika lahko vnesemo tudi nekaj najbolj osnovnih podatkov o kontekstnem strežniku, kot je identifikacijska številka strežnika in možnost izpisa podatkov v konzoli. Uporabniški vmesnik ponuja tudi ogled nekaterih lastnosti virov podatkov, ter njihovo ročno aktivacijo. Slednja možnost je vključena za namene testiranja omenjenih virov podatkov ne da bi bilo za to potrebno definirati ločeno aplikacijo.



Slika 14: Uporabniški vmesnik kontekstnega strežnika s pregledom osnovnih nastavitev.



Slika 15: Testiranje delovanja senzorja lahko opravimo preko vmesnika kontekstnega strežnika.

6.4. Kontekstno odvisna aplikacija

6.4.1. Motivacija in zasnova aplikacije

Kontekstno odvisna aplikacija za iskanje restavracij omogoča iskanja restavracij po različnih kriterijih. Eden od glavnih kriterijev je izbira tipa restavracija, ki se odraža v različnih vrstah hrane, ceni, ambientu v restavraciji, parkirnih mestih na voljo in ostalih. Da uporabniku ne bi bilo potrebno vnašati parametrov je aplikacija v odvisnosti od kontekstnega strežnika sama zmožna sklepati, kakšen tip restavracija išče uporabnik ter mu hkrati izpostavi tiste, ki so najbližje njegovi lokaciji. V kolikor se uporabniku zdi ena od restavracij zanimiva, si jo lahko ogleda. Pri tem mu aplikacija predlaga restavracije, ki so bile ogledana s strani ostalih uporabnikov, ki imajo enake interese kot opazovani uporabnik.

6.4.2. **Pridobivanje podatkov s pomočjo kontekstnega strežnika**

Da bi lahko realizirali takšno aplikacijo je potrebno najprej od kontekstnega strežnika pridobiti podatke o uporabnikovih interesih. Ena od možnosti bi bilo pridobivanje uporabnikove zgodovine brskanja in njegovo iskanje po priljubljenih spletnih brskalnikih. V ta namen je bil implementiran vtičnik kontekstnega strežnika, ki preko standardnega ponudnika vsebine (ang. *Content Provider*) dostopa do podatkov vgrajenega internetnega brskalnika. Ta ob prvem zagonu preko ponudnika vsebine preveri podatke, jim doda nekatere parametre ter jih shrani v tabelo obiskanih strani kontekstnega strežnika. Dodatno se v vtičniku nastavi opazovalca vsebine, ki ob novo obiskani strani določenega iskalnika doda iskani vnos v tabelo. Ker je ta namenjen le shranjevanju spletnih brskalnikov, je potrebno v njem definirati filter, ki iz obiskanih strani izloči tiste, ki tem stranem ne pripadajo.

id	timestamp	device_id	browser_title	browser_url	browser_visited_time
1	1410775673626.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	Google	https://www.google.com/	1410773996666
2	1410775673664.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	Google	https://www.google.si/?gws_rd=cr&ei=7LMWVNDINoTMyAOhqILQCA	1410773996666
3	1410775673692.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	chinese restaurant - Iskanje Google	https://www.google.si/search?safe=off&site=&source=hp&ei=7bMWVIYthILMA7XVgNgK&q=chinese+restaurant&oq=chinese+restaurant&gs_l=mobile-gws-hp.3..0l5.18617.38897.0.41366.38.22.7.9.9.0.156.2789.0j22.22.0....0...1c.1.53.mobile-gws-hp..0.38.2963.3.5wlG-br2jvU	1410774039100
4	1410775673733.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	italian restaurant business meeting - Iskanje Google	https://www.google.si/search?safe=off&site=&source=hp&ei=7bMWVIYthILMA7XVgNgK&q=chinese+restaurant&oq=chinese+restaurant&gs_l=mobile-gws-hp.3..0l5.18617.38897.0.41366.38.22.7.9.9.0.156.2789.0j22.22.0....0...1c.1.53.mobile-gws-hp..0.38.2963.3.5wlG-br2jvU#safe=off&q=chinese+nice+decor	1410774081620
5	1410775673825.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	italian restaurant kids - Iskanje Google	https://www.google.si/search?safe=off&site=&source=hp&ei=7bMWVIYthILMA7XVgNgK&q=chinese+restaurant&oq=chinese+restaurant&gs_l=mobile-gws-hp.3..0l5.18617.38897.0.41366.38.22.7.9.9.0.156.2789.0j22.22.0....0...1c.1.53.mobile-gws-hp..0.38.2963.3.5wlG-br2jvU#safe=off&q=italian+restaurant+business+meeting	1410774116221
6	1410775673857.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b		https://www.google.si/search?safe=off&site=&source=hp&ei=7bMWVIYthILMA7XVgNgK&q=chinese+restaurant&oq=chinese+restaurant&gs_l=mobile-gws-hp.3..0l5.18617.38897.0.41366.38.22.7.9.9.0.156.2789.0j22.22.0....0...1c.1.53.mobile-gws-hp..0.38.2963.3.5wlG-br2jvU#safe=off&q=italian+restaurant+kids	1410774127403
7	1410775673887.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	Bing	http://www.bing.com/	1410775198442
8	1410775673917.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	american traditional breakfast - Bing	http://www.bing.com/search?q=american+traditional+breakfast&qs=AS&form=QBLH&pq=american+traditional+brea&sc=1-25&sp=1&sk=	1410775217677
9	1410775673945.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	thai seafood - Bing	http://www.bing.com/search?q=thai+seafood&qs=n&form=QBRE&pq=thai+seafood&sc=11-12&sp=-1&sk=	1410775230333
10	1410775673970.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	thai seafood restaurant - Bing	http://www.bing.com/search?q=thai+seafood+restaurant&qs=n&form=QBRE&pq=thai+seafood+restaurant&sc=4-23&sp=-1&sk=	1410775238210
11	1410775673993.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	yahoo - Iskanje Google	https://www.google.si/search?q=yahoo&oq=yahoo&aqs=chrome..69i57j0j5j0.3129j0j4&client=ms-android-google&sourceid=chrome-mobile&espv=1&ie=UTF-8	1410775246988
12	1410775674120.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	Yahoo	https://www.yahoo.com/	1410775325005
13	1410775674148.0	e24ff622-676b-4fd3-85b8-dfc1ab50a56b	mexican tortillas kids - Yahoo Search Results	https://search.yahoo.com/mobile/s?p=mexican+tortillas+kids&.tsrc=yfp-hrmob-815	1410775316294

Tabela 1: Primer tabele kontekstnega strežnika s podatki obiskanih spletnih strani, ki pripadajo spletnim iskalnikom.

Da bi lahko te podatke uporabila ciljna aplikacija, v vtičniku s pomočjo lastnega razreda, dedovanega iz razreda *BroadcastReceiver*, definiramo možnost obveščanja aplikacij o novih podatkih v aplikaciji. Aplikacija bo obvestilo prejela za vsak novi vnos v tabeli, od programerja pa je odvisno ali bo te vnose takoj procesiral, jih shranil v svojo tabelo in kasneje obdelal ali pa ta obvestila prezrl. V slednjem primeru bi bilo v takšni aplikaciji bolj smiselno definirati razred za zajem podatkov neposredno iz tabele kontekstnega strežnika.

Na podoben način se v aplikaciji pridobiva tudi podatke o lokaciji. Aplikacija se s pomočjo poslušalca naroči na podatke o lokaciji uporabnika ter hkrati poda željene parametre. Ti parametri zajemamo vir pridobivanja lokacije (triangulacija mobilnih omrežij, ugotavljanje lokacija vira internetnega dostopa ali uporaba globalnega pozicioniranja preko satelitov), željeno minimalno natančnost rezultatov, frekvenco osveževanja meritev in ostale.

6.4.3. **Pogoji za implementacijo in rešitev**

Po definiranju osnovnih metod za pridobivanje podatkov je potrebno določiti še osnovne metode procesiranja prejetih podatkov v kombinaciji s podatki v aplikaciji. Za implementacijo ugotavljanja morebitnih interesantnih restavracij glede na uporabnikovo predhodno interakcijo z iskalniki si lahko pomagamo z metodami klasifikacij dokumentov in tekstovnim rudarjenjem. Klasifikacija ali uvrščanje je primer ene najpogostejših uporab strojnega učenja. Naloga klasifikacije je za nek določen objekt, opisan z množico različnih atributov (lastnosti, značilk) določiti, kateri izmed podmnožic možnih razredov pripada [12].. Takšni atributi so lahko različnih tipov, v osnovi pa ločimo neodvisne zvezne (višina, starost, temperatura…) ter diskretne spremenljivke (spol, krvna skupina, kategorija vozila, …), medtem ko je razred odvisna diskretna spremenljivka in se jo določi glede na vrednosti prej omenjenih atributov. Da bi lahko klasificirali nov, še neznan primerek, potrebujemo ustrezno diskretno funkcijo oz. model, ki lahko preslika množico atributov novega primerka v ustrezen razred. Takšen model je lahko določen že vnaprej, lahko pa je naučen iz predhodno določenega nabora testnih primerov.

Za klasifikacijo dokumentov imamo na voljo več različni rešitev, ki uporabljajo različne klasifikacijske metode. Te so na voljo za večino platform osebnih in strežniških računalnikov, vsebujejo različno število metod, algoritmov in orodij za omenjeno klasifikacijo in vizualizacijo podatkov ter bazirajo na različnih programskih jezikih (Python, R, Java, C, …). Ko pa se ozremo po podpori različnih mobilnih platform je situacija, vsaj v tem trenutku, precej drugačna. V primeru sistema Android v tem trenutki nismo našli uradno podprte ali vsaj s strani kakšne skupnosti trajno vzdrževane rešitve, ki bi bila primerna za uporabo na mobilni platformi.

Zaradi tega smo preučili nekatere rešitve, ki so odprtokodne in bazirajo na programskem jeziku Java, s čimer bi jih lahko pogojno lahko uporabili na platformi Android. Glede na obseg dokumentacije, nabor možnih funkcij in orodij ter ostalih zmožnosti, smo se odločili za orodje Weka, ki je že predhodno omenjeno v razdelku 3.3. Ta poleg namestitvene ali izvršljive datoteke za različne operacijske sisteme vsebuje tudi izvorne datoteke z razredi, ki so namenjeni uporabi v lastnih javanskih aplikacijah [14]. Po pregledu teh razredov in njihovem uvozu v razvojno okolje in ustvaritvi novega primerka knjižnice za sistem Android pa je bilo očitno, da nekatere javanske knjižnice na mobilni platformi niso podprte. Potrebne bi bile korenite in relativno zelo dolgotrajne spremembe posameznih, a med seboj zelo močno odvisnih razredov, kjer bi te module odstranili ali jih nadomestili s podprtimi. Na priljubljenem spletnem repozitoriju za hranjenje projektov smo kasneje odkrili knjižnico, kjer je avtor v testne namene sam preuredil razrede ene od starejših verzij orodja Weka za Android in jih dal v javno rabo [15]. Gre za različico 3.7.3, kjer pa je večina osnovnih orodij in metod enako funkcionalnih, kot v novejših različicah. Takšna preurejena knjižnica tako deluje tudi na operacijskem sistemu Android.

6.4.4. **Priprava podatkov in koraki implementacije**

Preden smo začeli z implementacijo klasifikacije na mobilni napravi, smo posamezne korake načrtovali in preizkusili skozi grafični vmesnik orodja v sistemu Windows. Prednosti uporabe tega vmesnika je predvsem prijaznost do začetnikov, dober povraten odziv ob napah, uporabi nepodprtih tipov atributov s strani posameznih metod, celovitega pregled vseh možnih opcij in parametrov posameznih metod on njihova enostavna izbira ter dober prikaz rezultatov in statističnih analiz posameznih metod, ki jih lahko po potrebi tudi izvozimo v datoteko. Po uspešni implementaciji vseh operacij smo te posamezne korake preizkusili še v javanski aplikaciji in primerjali rezultate s samostojno aplikacijo. Prenos te kode v projekt mobilne aplikacije namreč ne bo obsegal prevelikih sprememb, poleg tega pa bomo lahko s tem v koraku evalvacije primerjali rezultate in zmogljivosti obeh platform.

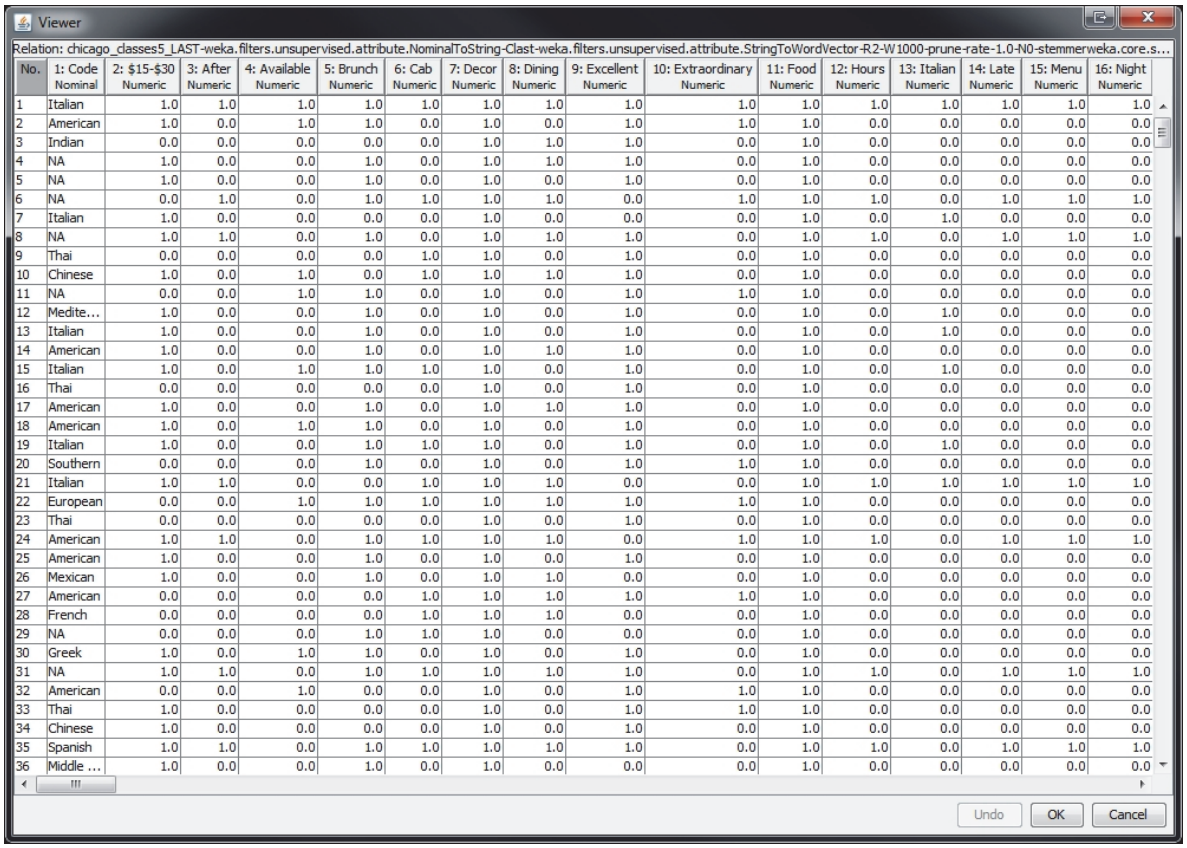
Hkrati je potrebno na strani aplikacije pripraviti podatke na procesiranje. V ta namen smo spisali metode za branje podatkov iz tabele restavracij v podatkovni baze in ustvarjanje datoteke ARFF (opisane v poglavju 3.3). Ker so v podatkovni bazi pod opisi restavracij zabeležene samo posamezna opisne kode, je te za namene tekstovne klasifikacije pretvoriti v zapis posameznih nizov.

Naslednji korak je uporaba pripravljene datoteke ARFF in pretvorba posameznih delov besedila v vektorsko obliko. Večina klasifikatorjev v Weki namreč ne more operirati z atributi, ki so predstavljeni v obliki nizov. Za ta postopek se uporablja vgrajen filter *StringToWordVector*, ki

opravi želeno transformacijo v vektorsko obliko [16]. Transformacija se lahko opravi na različnih tipih besedilnih dokumentov: različni besedilnih korpusi, arhivi, dnevne izdaje časopisov v elektronski obliki, spletne strani in ostali. Rezultat tega filtra, pri uporabi privzetih nastavitev, je nova datoteka ARFF, ki poleg razreda vsebuje posamezne besede izražene kot atribute, njihova vrednost pa odraža vsebovanost ali odsotnost posamezne besede v posameznem primerku razreda. Omenjen filter vsebuje še nekaj dodatnih parametrov, s katerimi lahko natančneje določimo pomembnost določenih pojavitev besed in zaporedij teh besed v besedilu [12]. Najpomembnejši filtri so naslednji:

- Korenjenje, lematizacija (ang. *stemming, lemmatizaion*) – Večina besed v besedilu, ki vsebujejo isti koren, ima zelo verjetno enak ali soroden pomen. S pomočjo postopka morfološke analize odstranimo predpone in konce besed v taki meri, da nam ostanejo le posamezni koreni. Posamezne besede z istim pomenom tako predstavimo v enotni obliki korena. Lematizacija se za razliko od korenjenja ukvarja tudi z lingvističnim pomenom besede. V primer lahko omenimo angleški besedi *car* in *automobile*, ki nimata nobenega skupnega korena, a imata v besedilu vseeno soroden pomen in ju moramo tako tudi obravnavati.
- *Stopwords* – S tem izrazom opisujemo besede, ki se pogosto uporabljajo v besedilih, vendar v osnovi nimajo nobenega pomena saj le povezujejo stavke med seboj. Zaradi tega so za namene klasifikacije neuporabne in jih je zaradi tega najbolj pametni ignorirati. Vsak jezik ima svoj nabor tako opisanih besed.
- Ocena TF-IDF (ang. *TF-IDF score*) – Ocena TF-IDF se uporablja za iskanje posameznih besed, ki so močno relevantna glede na dokumente v nekaterih nastopajo. TF (*term frequnecy*) označuje frekvenco posamezne besede, ki se pojavi v nekem dokumentu, DFT (*document term frequency*) pa označuje frekvence besede, ki se pojavlja v vseh dokumentih. Za izračun ocene TF-IDF se uporablja inverz omenjene frekvence DFT, saj so besede, ki se le redko pojavijo v vseh dokumentih, se pa zato pogosto pojavljajo znotraj teh dokumentov, veliko bolj zanimive za klasifikatorje kot tiste, ki se redno pojavijo v vseh različno klasificiranih dokumentih [13].
- Normalizacija dolžine besedil (*normalizeDocLength*) – Normalizacija pri izračunu frekvence neke besede upošteva tudi dolžino samega besedila. Beseda, ki se enkrat pojavi v kratkem besedilu ima po normalizaciji večjo frekvenco od besede, ki se je enkrat pojavila v daljšem besedilu.
- Določitev minimalne frekvence besede (*minTermFreq*) – S pomočjo tega filtra določimo kolikokrat se mora beseda pojaviti v dokumentih, da jo lahko še štejemo kot atribut.
- Leksikalna analiza za določitev besed (ang. *tokenization*) – Leksikalna analiza besedil predvideva uporabo različnih načinov za delitev teksta v posamezne dele. V

primeru analize *NGram* izbrani besedi določimo N-sorodnih besed in ocenimo verjetnost njihove pojavitve, če se zraven pojavi tudi sorodna beseda.



Viewer

Relation: chicago_classes5_LAST-weka.filters.unsupervised.attribute.NominalToString-Clast-weka.filters.unsupervised.attribute.StringToWordVector-R2-W1000-prune-rate-1.0-N0-stemmerweka.core.s...

No.	1: Code Nominal	2: $15-$30 Numeric	3: After Numeric	4: Available Numeric	5: Brunch Numeric	6: Cab Numeric	7: Decor Numeric	8: Dining Numeric	9: Excellent Numeric	10: Extraordinary Numeric	11: Food Numeric	12: Hours Numeric	13: Italian Numeric	14: Late Numeric	15: Menu Numeric	16: Night Numeric
1	Italian	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	American	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
3	Indian	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	NA	1.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
5	NA	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
6	NA	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0
7	Italian	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
8	NA	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0
9	Thai	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
10	Chinese	1.0	0.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
11	NA	0.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
12	Medite...	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
13	Italian	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
14	American	1.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
15	Italian	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
16	Thai	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
17	American	1.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
18	American	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
19	Italian	1.0	0.0	0.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
20	Southern	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
21	Italian	1.0	1.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0
22	European	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
23	Thai	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
24	American	1.0	1.0	0.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0	1.0	1.0
25	American	1.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
26	Mexican	1.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
27	American	0.0	0.0	0.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
28	French	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
29	NA	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
30	Greek	1.0	0.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
31	NA	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0
32	American	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
33	Thai	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0
34	Chinese	1.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
35	Spanish	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0
36	Middle ...	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

Undo OK Cancel

Slika 16: Datoteka z vektorsko obliko besedila po izvedbi filtra *StringToWordVector*.

Po pripravi datoteke z vektorsko obliko besedila je bilo potrebno pripraviti še podatke, ki jih bomo klasificirali. Pri tem se obrnili na različne ponudnike javno dostopnih virov podatkov, ki se lahko uporabijo za različne namene klasifikacije. Večina virov, namenjenim tekstovni klasifikaciji, vsebuje zapise iz literarnih del, časopisov in ostalih korpusov. Iskali smo podatke za aplikacijo, ki opisuje neko fizično entiteto, ta pa ima več različnih lastnosti, po katerih jo lahko kategoriziramo. Kot najprimernejša se je izkazala baza podatkov o restavracijah, ki se je uporabljala v spletnem priporočilnem sistemu *Entree* oziroma *FINDME* [17]. Ta za regijo Chicago vsebuje približno 680 restavracij, ki za svoj opis vsebujejo različne oznake, ki so v tej datoteki predstavljene s trimestnimi številkami. Te predstavljajo oznako opisa, ki se nahaja ločeni datoteki s značilnostmi (Programska koda 3). S pomočjo teh oznak smo posamezni restavraciji določili tip hrane, ki jo nudi, ter povprečno ceno obroka (cena obroka je že zabeležena kot oznaka). Ostale oznake smo uporabili za opis restavracije, ki se bo v postopku klasifikacije uporabljala in primerjala z vnesenimi iskalnimi podatki uporabnika.

```
0000001 Village 026 249 174 004 132 249 198 191 192 125 075 205 054 165
0000002 Millrose Brewing Company 137 249 194 215 213 174 249 191 192 008
075 205 054 165
0000000 Moti Mahal 214 035 149 021 117 075 204 051 163
0000003 Dover Straits 137 190 174 249 212 075 205 053 165
0000004 Eat Your Hearts Out 214 249 249 197 111 025 025 112 075 205 053 164
0000005 Pizzeria Uno & Due 026 249 004 132 100 086 149 182 076 204 052 163
0000006 Trattoria Franco 214 136 125 078 208 056 170
0000007 Little Bucharest 214 004 132 249 198 201 074 205 052 164
0000008 Pattaya 026 235 074 205 051 163
0000009 House of Hunan 026 191 192 024 021 039 075 205 052 164
0000010 Morton's of Chicago 137 174 099 249 245 189 191 192 114 024 225 076
206 053 168
0000011 Jezebel 214 174 249 200 196 125 140 075 205 053 166
…
```

Programska koda 3: Izpis izseka podatkovnega seta z restavracijami in njihovimi oznakami značilnosti v regiji Chicago.

V primerih testiranja klasifikacije preko grafičnega vmesnika imamo pri klasifikaciji po določeni metodi na voljo izbiro testnega nabora podatkov, nad katerim bo opravljena ocena uspešnosti posameznega modela. Na izbiro imamo uporabo validacije nad učnim naborom podatkov za izgradnjo modela (ang. *train set*), izbiro ločeno definiranega testnega nabora (ang. *test set*), določitev razmerja s katerim učni nabor podatkov "odrežemo" in tega dela zgradimo testni nabor ali pa uporabimo prečno preverjanja. Zadnja metoda se uporablja za primere, ko imamo na voljo zelo malo vhodnih podatkov za izgradnjo modela in bi vsakršna izločitev primerov, ki bili namenjeni le ocenjevanja uspešnosti tega modela, lahko prikrajšala postopek generiranja modela in vplivala na njegovo uspešnost. Tudi podatke o uporabnikovem iskanju je potrebno umestiti v ARFF datoteko, pri kateri moramo paziti, da ima ta vse možne razrede iste kot v ARFF datoteki z opisi restavracij. V nasprotnem primeru dobimo opozorilo, da struktura datoteke ni ustrezna.

Po uspešno ustvarjenih naborih podatkih je potrebno na teh izbrati ustrezne metode, določiti njihove parametre ter jih preizkusiti. Različni metode se drugače obnašajo glede na različne dejavnike, začenši s strukturo in lastnostmi podatkov, na katerih izvajajo učenje. Pri tem moramo vzeti v obzir tudi njihovo optimalno delovanje na mobilnih napravah. V primeru enostavnih uporabniških aplikacij mora biti razmerje med natančnostjo in hitrostjo odločanja bolj v prid slednjemu, saj je poleg dobre uporabniške izkušnje s tem pogojena tudi poraba sredstev in energije. Nezanemarljiva je tudi poraba pomnilnika, vsaka aplikacija v Androidu ima namreč natančno odmerjen določen del pomnilnika (ang. *heap space*). Tega v preteklosti ni bilo možno spreminjati, od verzije 2.3 pa je to pogojno možno. Spodaj so navedene nekatere metode, ki se uporabljajo v tekstovni klasifikaciji skupaj z našo oceno prenosljivosti na mobilna napravo:

- **Naivni Bayesov klasifikator** (ang. *Naive Bayes classifier*) – Omenjeni klasifikator v Weki obstaja v več različnih izvedenkah. V osnovi deluje tako, da za vsak atribut, ki v našem primeru predstavlja pojavitev ali odsotnost neke besede, izračuna pogojno verjetnost pojavitve glede na pripadnost posameznemu razred ter na koncu za posamezni razred sešteje omenjene verjetnosti. V kolikor so verjetnosti določenega atributa med posameznimi razredi zelo podobne (tako je razmerje med temi vrednostmi zelo podobno), to pomeni, da se zelo enakomerno pojavljajo v vseh primerkih različnih razredov in bi zato morale imeti manj vpliva pri klasifikaciji. Ime "naivni" se sicer nanaša na dejstvo, da se verjetnosti posameznih pojavitev besed znotraj enega razreda pri določanju verjetnosti razreda upoštevajo neodvisno od drugih besed. Zaradi tega razloga je ta metoda računsko zelo učinkovita, glede na pristop pa tudi relativno natančna in zato primerna za nadzorovano učenje, Dobro se obnese prav tudi pri podatkih, ki ima veliko število atributov proti številu instanc, kot je to pogosto pri klasifikaciji dokumentov.
- **Multinomialni Bayesov klasifikator** (ang. *Multinomial Bayes classifier*) –Posebna izvedenka Bayesovega klasifikatorja, ki je namenjena prav klasifikaciji tekstov in predvsem ugotavljanju razlik med različnimi tehnikami gradnje ustreznega učnega nabora, uporabi različnih možnosti sintaktične analize, ter njihovih vplivih na procesiranja teksta. V osnovi je zelo podoben osnovnemu Bayesovemu klasifikatorju, le da ob učenje upošteva tudi distribucijo posameznih atributov v razredih, kjer se za vse razrede uporablja multinomialna distribucija. Zaradi tega lahko v Weki ob analizi tega klasifikatorja vidimo tudi podatke o verjetnostih pojavitve določene besede glede na posamezen razred.
- **K-najbližjih sosedov** (ang. *K-nearest neighbors, k-NN*) – Metoda *k*-najbližjih razredov se uporablja za določitev razreda nekega primerka, kjer glede na lastnosti tega primerka izberemo nekaj najbližjih sosedov v učni množici in mu priredimo razred, ki je med temi sosedi najbolj pogost. S številom *k* tako določimo, koliko najbližjih sosedov obravnavamo. Vrednost *k* se v različnih primerih določa eksperimentalno. V kolikor so v učni množici možne napačno klasificirane instance z izbiro večjega števila *k* zmanjšamo verjetnost, da so bi bili vsi *k* najbližji sosedi iz učne množice napačni. Večje število *k* bo v primeru diskretnih razredov ponujal večjo natančnost napovedi, medtem ko bo z izbiro manjšega števila *k* algoritem porabil manj virov. Za izračun razdalje do *k*-najbližjih sosedov se uporablja več različnih metod, v Weki je privzeta metoda evklidska razdalja.
- **Algoritem J48/C4.5** (ang. *J48/C4.5 decision tree algorithm*) – Algoritem J48 je odprtokodna implementacija algoritma C4.5 v Javi, ki se uporablja za izgradnjo odločitvenega drevesa. Sestavljeno je iz atributov, predstavljenimi z notranjimi vozlišči, podmnožicam atributom kot vejami in razredi, ki predstavljajo liste. Ena pot od korena preko vozlišč do vej in končnega lista ustreza enemu odločitvenemu

pravilu. Število listov je tako enako številu odločitvenih pravil. Znan je kot relativno hiter in (glede na hitrost) tudi natančen algoritem, ki pa ima lahko pri večjih podatkovnih naborih težave s porabo pomnilnika. S pomočjo metode rezanja (ang. *pruning*) lahko postopoma izločimo posamezne dele drevesa, ki ne prispevajo veliko h končni natančnosti algoritma.

Posamezne algoritme smo preizkusili na naši testni podatkovni bazi, kjer smo za analizo konfiguracij uporabili postopke prečnega preverjanja in izmerili čase izvajanja posameznih korakov. Posamezne konfiguracije so se med seboj ločevale predvsem pri izbiri različnih parametrov operacije *StringToWordVector*. Nekatere od teh so namenjene različnim nalogam, nekatere so specifične za klasifikacijo večjega nabora besedil, kjer ustrezna uporaba teh parametrov bolj vpliva na uspešnost klasifikatorja. V primeru našega podatkovnega nabora, kjer za analizo uporabljamo le značke, večina teh parametrov ni imela velikega učinka. V primeru leksikalne analize označevanja z uporabo n-gramov se je, pri upoštevanju ene predhodne besede, močno povečal samo končni čas izvajanja, medtem ko je končna natančnost klasifikatorja ostala približno enaka. Podrobnosti teh meritev in njihova analiza sledi v naslednjem poglavju, spodaj pa se nahaja nekaj osnovnih razredov za definiranje klasifikatorja (Programska koda 4), definiranje modela s pomočjo klasifikatorja in podatkov za učenje (Programska koda 5) ter uporaba klasifikatorja za klasificiranje instanc (Programska koda 6).

```
public void createClassifier() throws Exception{

        trainData.setClassIndex(0);
        filter = new MultiFilter();

        STWFilter = new StringToWordVector();
        STWFilter.setAttributeIndices("last");
        STWFilter.setDoNotOperateOnPerClassBasis(true);
        STWFilter.setLowerCaseTokens(true);
        STWFilter.setOptions(weka.core.Utils.splitOptions(STWoptions));
        STWFilter.setTokenizer(new WordTokenizer());

        InfoGainAttributeEval IGAEval = new InfoGainAttributeEval();
        Ranker ranker = new Ranker();
        ranker.setThreshold(0.0);
        AttFilter.setEvaluator(IGAEval);
        AttFilter.setSearch(ranker);

        filter.setFilters(new Filter[]{STWFilter});


        classifier = new FilteredClassifier();
        classifier.setFilter(filter);

        J48 actualClassifier = new J48();

        classifier.setClassifier(actualClassifier);

}
```

Programska koda 4: Metoda za določitev klasifikatorja in njegovih parametrov.

```
public void learn() {
	try {
		createClassifier();

		classifier.buildClassifier(trainData);

		FileOutputStream fos = openFileOutput(modelFile, Context.MODE_PRIVATE);
		weka.core.SerializationHelper.write(fos, classifier);
		fos.close();

		System.out.println("== Training on filtered (training) dataset done ==");
	}
	catch (Exception e) {
		System.out.println("Problem found when training");
	}
}
```

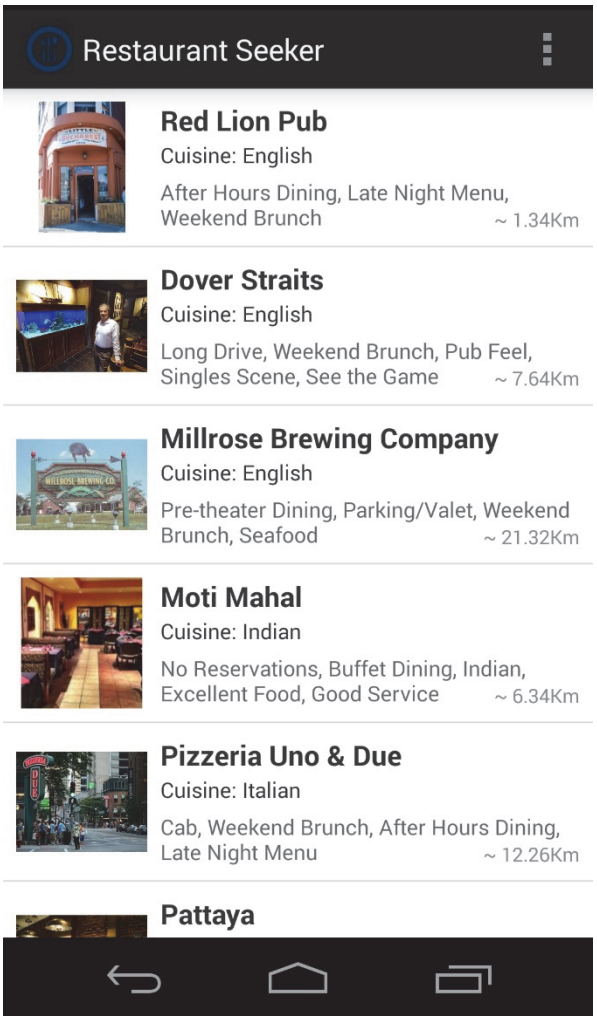
Programska koda 5: Metoda za izdelavo modela in njegovo shranjevanje.

```
public void classify() {
	try {
		FileInputStream fin = openFileInput(modelFile);
		classifier = (FilteredClassifier) weka.core.SerializationHelper
				.read(fin);
		fin.close();

		double pred = classifier.classifyInstance(instances.instance(0));
		double[] eval = classifier.distributionForInstance(instances
				.instance(0));

		System.out.println("===== Classified instance =====");
		System.out.println("Class predicted: "
				+ instances.classAttribute().value((int) pred));
		System.out.println("Class predicted: " + (int) pred);
		System.out.println("Class distributions: " + Arrays.toString(eval));

	} catch (Exception e) {
		System.out.println("Problem found when classifying the text");
	}
}
```

Programska koda 6: Metoda za klasifikacijo izbrane instance in izpis rezultatov.

Končni izgled aplikacije je predstavljen na spodnji sliki (Slika 17). Kot je bilo omenjeno v začetku opisa aplikacije ta za določitev seznama restavracij upošteva najbližje restavracije ter ji tako tudi razvrsti. Ob prejemu podatka o uporabnikovem iskanju, se izvrši posebna metoda, ki na naboru prejetih podatkov zažene tekstovno klasifikacijo. Po opravljeni klasifikaciji metoda preuči vrnjene podatke, v katerih najdemo ustrezno oznako klasificiranega razreda in vrednosti zaupnosti (ang. *confidence*) za posamezne razrede glede na instanco. S pomočjo tega podatka lahko ugotovimo, ali bi lahko tak rezultat sploh uporabili za določanje najljubših tipov restavracij. Po uspešni določitvi tega razreda se vsebina v aplikaciji asinhrono osveži in na vrhu

predstavi tri uporabniku najbližje restavracije, ki ustrezajo predvidenemu tipu. Ostale restavracije se glede na oddaljenost razvrstijo za njimi.



Slika 17: Uporabniški vmesnik naše kontekstno odvisne aplikacije za iskanje bližnjih restavracij.

7. EVALVACIJA PREDLAGANE REŠITVE

Predlagano arhitekturo kontekstnega strežnika na mobilni napravi in zasnovo kontekstno odvisne aplikacije smo prenesli na mobilno platform Android in preizkusili njuno delovanje. Prva faza testiranja delovanja je na strani kontekstnega strežnika obsegala izvajanje različnih aktivnosti aktivacije posameznih virov kontekstnih podatkov, ki so polnili podatkovno bazo in obremenjevali napravo, ter ob tem opazovali delovanje kontekstnega strežnika. Hkrati smo začeli uporabljati tudi našo kontekstno odvisno aplikacijo, ki pa od kontekstnega strežnika še ni prejela nobenih podatkov, saj smo jo za namene testiranja odjavili od prejema kontekstnih podatkov. Ta faza je potekala najdlje, saj je potekala v več različnih korakih implementacije ter razvoja kontekstnega strežnika in aplikacije, ki je poleg raziskave in testiranja posameznih modulov v sistemu Android vključevala odpravljanja napak in hroščev, ki so se pojavili med razvojem celotne strukture.

Druga faza je obsegala testiranje medsebojnega delovanja kontekstnega strežnika in kontekstno odvisnih aplikacij. Opazovali smo predvsem funkcijo samodejnega pridobivanja podatkov s strani kontekstnega strežnika in funkcijo obveščanja aplikacij o spremembah v kontekstu. Pri testiranju pošiljanja obvestil smo uporabili več preprostih kontekstno odvisnih aplikacij, ki so obvestila o podatkih samo prejemale, medtem ko je glavnino testiranja predstavljala omenjena aplikacija za iskanje restavracij. Na strani te aplikacije so v teh primerih potekale operacije samodejnega razčlenjevanja prejetih podatkov, gradnja ustreznih podatkovnih setov za postopke klasifikacije iskalnih nizov, klasificiranje podatkov iz podatkovne baze ter uveljavitvi prikaza rezultatov. Testiranje v tej fazi je obsegalo predvsem performančno testiranje metod za klasifikacijo iskanih pojmov. Da bi lahko dobili širšo sliko o smotrnosti takšnega procesiranja smo omenjeno testiranje izvedli na dveh različnih mobilnih napravah in njune rezultate primerjali z izvajanjem identičnih postopkov na klasičnem računalniškem sistemu.

Končna evalvacija mobilnega kontekstnega strežnika je zelo težko izvedljiva in predstavljiva s pomočjo objektivnih meritev, saj se večina prednosti in slabosti takšnega načina procesiranja konteksta, poleg performaničnih, izraža lahko le kot subjektivna ocena dejanske uporabnosti omenjene arhitekture. Njena uporabnost je lahko ovrednotena le z naborom pripadajočih aplikacij, ki lahko podatke kontekstnega strežnika učinkovito uporabijo in predstavijo. Ker lahko končni uporabniki takšne aplikacije, glede na princip samodejnega opazovanja in zbiranja podatkov, dojemajo različno in jih kot take tudi ovrednotijo, smo se za namen celovite ocene strukture odločili uporabiti analizo SWOT. To je ena najbolj popularnih analiz, ki se uporablja za podajanje ocene različnih entitet, od oseb do produktov, serije produktov ali neke druge rešitve. Analizo razmejimo glede na štiri dele oz. aspekte, ki predstavljajo prednosti in slabosti (ang. *Strengths/Weaknesses*) ter priložnosti in nevarnosti (ang. *Opportunities/Threats*), ki vplivajo na opisovano entiteto. Razlika med njima je predvsem v tem, da se prvi dve večinoma

nanašata na notranje dejavnike, medtem ko se drugi dve osredotočata na zunanje dejavnike in vplive. S pomočjo te analize smo skušali oceniti dejansko vrednost omenjene arhitekture ter jo bolj učinkovito primerjati s standardno arhitekturo kontekstnega strežnika.

7.1. Prva faza – splošno testiranje kontekstnega strežnika

V prvi fazi evalvacije smo predlagano shemo kontekstnega procesiranja po predlaganih korakih začeli implementirati v sistemu Android. Ker so bili ti koraki med seboj zelo močno odvisni in povezani, smo morali že v začetnih korakih veliko pozornosti nameniti testiranju teh korakov.

V fazi zajema podatkov je bilo največ težav z zajemom podatkov v primerih, ko uporabnik naprave ne uporablja, saj se lahko tak zajem med neaktivnostjo preneha. Med testiranjem tega smo morali preizkusiti tudi poslušalce kontekstnih virov in ostale razrede za samodejni zajem konteksta. Primer tega je poslušalec obiskanih spletnih strani brskalnika, ki ni hotel delovati. Razlog je tičal v napačno definiranem naslovu tega ponudnika konteksta, saj se ta s prihodom novega brskalnika zamenjal. Drugače je sam zajem podatkov deloval presenetljivo transparentno, saj večjih zakasnitev pri delovanju ni bilo opaziti. Pri uporabi zajema iz več senzorjev naenkrat je bilo obremenitev osrednjega procesorja moč zaznati le v ustrezni aplikaciji za nadzor obremenitve procesorja. Pri prehodu iz zaznave procesiranja v agregacijo teh podatkov prav tako ni bilo opaznih sprememb v delovanju in napakah, ki ne bi bili posledica naše krivde. Shranjevanje podatkov v podatkovno bazo smo najprej preizkusili implementirati brez razreda *ContentProvider*, kar se je izkazalo za zelo napačno. Pri tem smo namreč v konzoli občasno prejeli različne napake, saj se podatki v bazo niso mogli zapisovati iz istega razreda kontekstnega strežnika. Omenjeni razred je te napake uspešno odpravil, poleg tega pa je bilo potrebno vsakemu delujočemu senzorju ustvariti še lastni proces. Omenjeni proces je bilo potrebno ustvariti tudi zaradi napak pri shranjevanju nastavitev za vsak posamičen senzor. Deljenje podatkov s pomočjo razreda *BroadcastReceiver* smo poznali že od prej, vendar smo tekom implementacije morali odstraniti možnost dodajanja podatkov v osrednji razred teh opozoril. Omenjenih opozoril je bilo namreč pri testiranju sočasnega zajema iz več senzorjev enostavno preveč, kar se je poznalo tudi na odzivnosti same aplikacije in predvsem pri porabi virov. Preizkusili smo tudi delovanje MQTT modula kontekstnega strežnika, ki smo mu s pomočjo omenjene spletne pošiljali osnovna sporočila.

Ob koncu implementacije smo še enkrat preizkusili delovanje posameznih sklopov. Cilj za uspešno delovanje je bilo že prej omenjeno ustvarjanje ločenih procesov oziroma storitev za vsak senzor posebej. To je bilo nujno tudi zaradi omogočitve delovanja naprave v mirovanju. Žal ima takšno delovanje tudi posledice pri porabi energije, saj je bila ta v primerjavi z drugimi aplikacijami, sploh v primerih energijsko zelo potratnih senzorjev, zelo znatna. Gledano z

vidika celotnega sistema je bil ob testiranjih tak kontekstni strežnika relativno stabilen in dobro odziven. Veliko zaslug za to gre šibko sklopljeni arhitekturi posameznih faz, ki delujejo vzajemno, vendar niso popolnoma odvisne ena od druge. Končni nabor podprtih kontekstnih virov je tako naslednji: merilnik pospeška, podatki o nameščeni aplikacijah in novih namestitvah, barometer, podatki o bateriji, podatki o *Bluetooth* komunikacijiah, podatki o klicih, gravitacijski senzor, žiroskop, merilnik svetlobe, podatki o lokaciji, linearni merilnik pospeška, magnetogram, omrežni podatki, merilnik oddaljenosti, podatki o statusu ekrana, temperatura, trenutna časovna cona in podatki o brezžičnih WiFi komukacijah.

7.2. Druga faza – testiranje delovanja kontekstnega strežnika in aplikacije

V drugi fazi smo s pomočjo delujoče instance kontekstnega strežnika izdelali kontekstno odvisno aplikacijo in testirali njihovo medsebojno delovanje. Posebno pozornost smo tukaj namenili kontekstno odvisni aplikaciji, v kateri smo izvajali metode klasifikacije teksta za ugotavljanje sorodnosti uporabnikovega iskanja in podatki te aplikacije. Pri tem smo uporabili že v poglavju 6.4 omenjeno podatkovno bazo o restavracijah.

7.2.1. **Vzajemno delovanje strežnika in aplikacije**

Podobno kot v testiranjih v prvi fazi, smo tudi pri izdelavi aplikacije definirali ustrezen prejemnik obvestil o novih kontekstnih podatkih in tako testirali vzajemno delovanje sicer ločenih aktivnosti aplikacije in kontekstnega strežnika. Na napravi smo nato odprli brskalnik in v spletne strani iskalnikov vnesli nekaj različnih pojmov. Ti pojmi so se ob vnosu shranili v privzeto tabelo ponudnika tega vira in obvestili kontekstni strežnik. Ta je sporočilo posredoval aplikaciji, ki je podatke prebrala in shranila v svojo bazo.

Na strani aplikacije smo nato preverjali število in lastnosti zajetih dogodkov ter kasnejše razčlenjevanje iskanih pojmov glede na obisk določene strani. Ta je bil za namene testiranja relativno enostaven in je podpiral le nekaj najbolj znanih spletnih iskalnikov. Rezultati so bili primerljivi s preteklimi testiranji, saj smo na strani aplikacije uspešno prejeli vse uporabnikove dogodke in iz njih izluščili iskane pojme.

7.2.2. Testiranje delovanja tekstovne klasifikacije

Za namene tekstovne klasifikacije smo morali najprej pripraviti podatke o restavracijah, ki jih bomo uporabili za tekstovno klasifikacijo in prikaz v omenjeni aplikaciji. Omenjene podatke smo tako s pomočjo razčlenjevalnika prebrali iz običajne datoteke in jih shranili v bazo. Za namene tekstovne klasifikacije to še ni bilo dovolj, saj je bilo potrebno te podatke razčleniti ter iz njih ustvariti ustrezno datoteko ARFF, ki jo lahko uporabimo za metode tekstovne klasifikacije.

Po ustvaritvi te datoteke smo na napravi začeli s testiranjem različnih metod, ki omogočajo tekstovno klasifikacijo. Vsako metodo smo v fazi raziskovanja večkrat preizkusili ter iz izpisa prečnih preverjanj preučevali ocene ustvarjenega modela. Pri tem smo ocene modelov primerjali z rezultati testnega projekta v klasičnem javanskem projektu na osebnem računalniku. Pri tem razlik v omenjenih ocenah modelov na obeh platformah ni bilo opaziti. Po uporabi različnih klasifikatorjev in metode predprocesiranja podatkov smo primerjali tudi čas izvajanja posameznega algoritma na obeh platformah, s čimer smo skušali bolj učinkovito primerjati obe platformi. Za primerjavo platform smo uporabili kar čas izvajanja posameznih algoritmov. Ta rezultat mogoče ne odraža vseh kompleksnosti posameznih algoritmov, je pa dobra referenca za njuno primerjavo, ki jo v mogoči zakasnitvi delovanja mogoče opazil tudi končni uporabnik.

Testiranje delovanja omenjenih operacij smo tako izvedli na prenosnem računalniku ter dveh mobilnih napravah:

- Prenosni računalnik Lenovo s štirijedrnim procesorjem Intel Core i7 3630QM s frekvenco 2.4–3.4 Ghz ter 12Gb pomnilnika. Leto izdaje procesorja je 2012.
- Mobilna naprava LG Nexus 5 s sistemom Qualcomm Snapdragon 800 MSM8974, ki vključuje štirijedrni procesor Krait 400 s frekvenco 2.3Ghz ter 2Gb pomnilnika. Leto izdaje nabora je 2014. Nameščeni operacijski sistem je Android 4.4.4.
- Mobilna naprava LG Optimus G z vgrajenim sistemom Qualcomm MDM9615/APQ8064, ki vključuje štirijedrni procesor Krait 200 s frekvenco 1.5Ghz ter 2Gb pomnilnika. Leto izdaje nabora je 2012. Nameščeni operacijski sistem je Android 4.1.2.

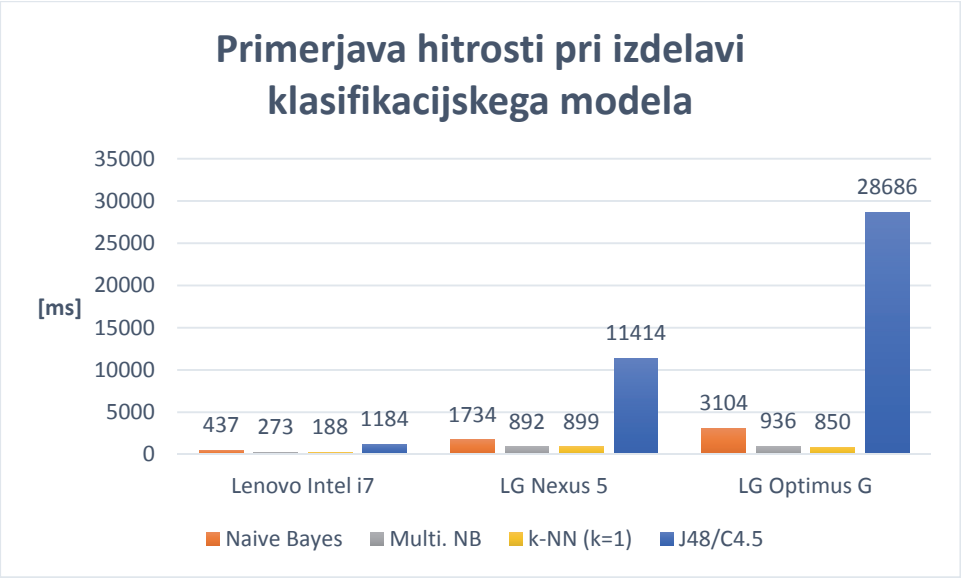
Pri izdelavi testnih scenarijev smo uporabili našo bazo restavracij in njihovih opisov ter izdelali dva ločena projekta za izvedbo testiranja in preučitev rezultatov. Standardni javanska aplikacija je namenjena izvajanju na klasični računalniški platformi, medtem ko se je aplikacija v Androidu izvajala na obeh omenjenih mobilnih napravah. Oba sta uporabljaje iste razrede in izvajale iste funkcije, kot so tiste v kontekstno odvisni aplikaciji.

Upoštevali smo posamezne korake tekstovne klasifikacije in testirali posamezne korake na omenjenih platformah. Ti so obsegali definicijo osnovnega klasifikatorja in filtra za pretvorbo posameznih besed v vektorje. Pred začetkom smo na obeh platformah izvedli prečno preverjanje omenjenih rešitev, s katerimi smo dobili osnovne podatke o zmožnostih tako generiranega modela za klasifikacijo novih instanc. Te podatke smo medsebojno primerjali in uvideli, da je natančnost omenjenih klasifikatorjev identična na obeh platformah. Ena od ocen takšnega modela je tako tudi ocena v obliki odstotka pravilno klasificiranih dogodkov. Omenjeno oceno smo vključili tudi v tabelo rezultatov testiranj (Tabela 2).

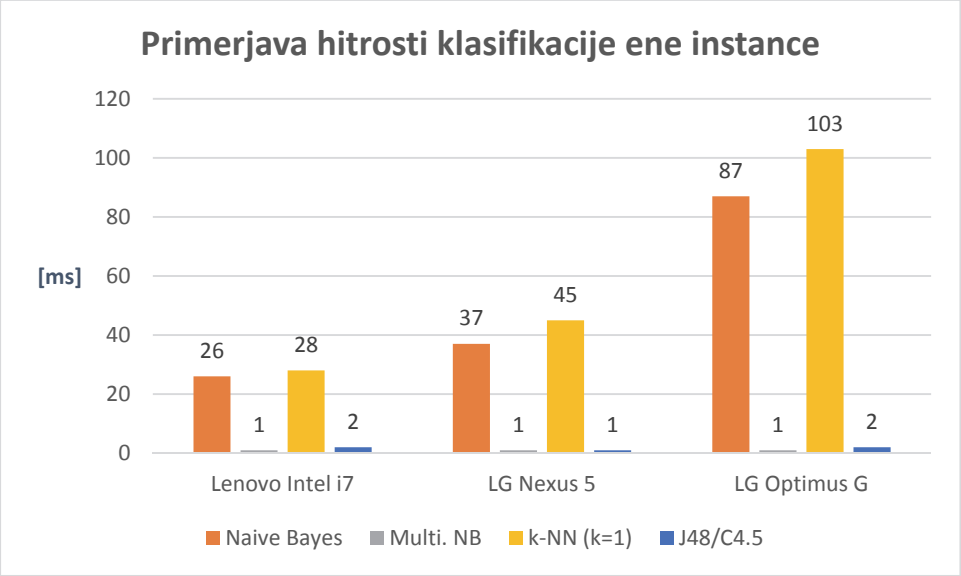
		LENOVO INTEL I7		**LG NEXUS 5**		**LG OPTIMUS G**	
ALGORITEM	Natančnost klasifikatorja	Generir.. modela [ms]	Klasifik. instance [ms]	Generir. modela [ms]	Klasifik. instance [ms]	Generir. modela [ms]	Klasifik. instance [ms]
NAIVE BAYES	79,59%	437	26	1734	37	3104	87
MULTINO. NB	67,90%	273	1	892	1	936	1
K-NN (K=1)	55,92%	188	28	899	45	850	103
J48/C4.5	89,65%	1184	2	11414	1	28686	2

Tabela 2: Podatki o uspešnosti posameznih algoritmov in izvajalni časi posameznih korakov tekstovne klasifikacije na klasičnem računalniku in dveh mobilnih platformah.

Nato smo nadaljevali s testiranjem hitrosti posameznih algoritmov. Posamezni deli testiranja so obsegali hitrost generiranja ustreznega modela za klasifikacijo ter kasnejšo uporabo modela za klasificiranje posameznih instanc. Prvi omenjeni korak je zaradi uporabe posebnega konstruktorja klasifikatorja že vseboval funkcijo pretvarjanja posameznih besed v vektorje in vse vključene postopke priprave in obdelave teh besed. Ker je omenjeno delovanje prisotno tudi v primeru naše kontekstno odvisne aplikacije teh korakov nismo obravnavali ločeno, ampak smo jih razdelili samo na korak generiranja modela in klasifikacije instance. Po uspešnem generiranju modela smo tega s pomočjo ustreznih razredov shranili v datoteko v serializirani obliki. Pri testiranju klasifikacije smo ta model v obratnem postopku prebrali v razred klasifikatorja in ga uporabili pri klasifikaciji instanc. Pri merjenju hitrosti obeh korakov smo uporabili standardne razrede za pridobivanje trenutnega časa sistema v nanosekundah ter vsako meritev večkrat ponovili. Povprečje pridobljenih meritev smo nato vnesli v omenjeno tabelo. Pri omenjenem merjenje nismo upoštevali izvajanja ostalih pomožnih aktivnosti, kot so zapisovanja in branje omenjenega modela ter izpis morebitnih rezultatov v konzoli.



Slika 18: Graf primerjave hitrosti izdelave modela v odvisnosti od uporabljenega algoritma na različnih platformah.



Slika 19: Graf primerjava hitrosti klasifikacije instance v odvisnosti od uporabljenega algoritma za izdelavo modela na različnih platformah.

Omenjene rezultate smo predstavili tudi v obliki dveh grafov, na katerih lahko še enostavneje razločimo razlike med omenjenima platformama in uporabi različnih metod za klasifikacijo (Slika 18, Slika 19). Po pregledu podatkov o potrebnem času za generiranje modela lahko opazimo, da je bila klasična platforma pri tej nalogi še vedno dosti hitrejša od obeh mobilnih. Pri prvih treh metodah je klasična platforma za okoli štirikrat hitrejša od mobilne platforme naprave Nexus 5, medtem ko je faktor hitrosti pri uporabi algoritma za generiranje dreves na

klasični platformi kar desetkrat večji. Po našem mnenju na omenjeni faktor poleg delovanja same procesne enote precej vpliva tudi boljši izkoristek prostega pomnilniškega prostora pri običajnem javanskem navideznem stroju. Kljub temu, da imata obe mobilni napravi dovolj pomnilnika, pa je prav upravljanje s tem pomnilnikom tisto, ki prispeva k hitrosti delovanja. Predvsem pri malo starejši napravi LG Optimus G smo v konzoli ob opravljanju te operacije zaznali močno povečano delovanje vgrajenega orodja za sproščanje pomnilnika (ang. *garbage collection*). Pri enem od bolj stresnih preizkusov ta ni mogel sprostiti dovolj pomnilnika za delovanja algoritma, zato je posledično sledilo zaprtje aplikacije. Omenjeni izpis konzole lahko vidimo v spodnjem izseku kode (Programska koda 7). Zanimiva je predvsem zadnja vrstica, kjer lahko razločimo prejem signala za prisilno ustavitev niti procesa.

```
09-23 05:30:02.635: D/dalvikvm(12232): WAIT_FOR_CONCURRENT_GC blocked 28ms
09-23 05:30:02.955: D/dalvikvm(12232): GC_CONCURRENT freed 2895K, 38% free
12396K/19843K, paused 12ms+4ms, total 60ms
09-23 05:30:16.880: D/dalvikvm(12232): GC_CONCURRENT freed 2865K, 37% free
12621K/19843K, paused 13ms+3ms, total 51ms
09-23 05:30:25.389: D/dalvikvm(12232): GC_CONCURRENT freed 3435K, 36% free
12878K/19843K, paused 2ms+3ms, total 46ms
09-23 05:30:26.641: D/dalvikvm(12232): GC_CONCURRENT freed 4278K, 35% free
12920K/19843K, paused 3ms+12ms, total 58ms
09-23 05:30:27.251: D/dalvikvm(12232): GC_CONCURRENT freed 5086K, 39% free
12262K/19843K, paused 12ms+12ms, total 65ms
09-23 05:30:27.602: D/dalvikvm(12232): GC_CONCURRENT freed 3069K, 39% free
12225K/19843K, paused 3ms+3ms, total 42ms
09-23 05:30:27.852: D/dalvikvm(12232): GC_CONCURRENT freed 2701K, 38% free
12385K/19843K, paused 14ms+3ms, total 44ms
09-23 05:30:30.985: D/dalvikvm(12232): GC_CONCURRENT freed 1251K, 38% free
12488K/19843K, paused 2ms+24ms, total 58ms
09-23 05:30:33.127: A/libc(12232): Fatal signal 6 (SIGABRT) at 0x00000226
(code=0), thread 12232 (le.wekaandroid1)
```

Programska koda 7: Izpis orodja za sproščanja pomnilnika v konzoli, ki je nastal ob stresnem testiranju algoritma za gradnjo dreves.

Pri ogledu drugega grafa s primerjavo hitrosti dejanske klasifikacije instance (Slika 19) pa lahko opazimo, da je v tem primeru razkorak med klasično in mobilno platformo dosti manjši kot v prvem primeru. Faktor hitrejšega delovanja klasične platforme v primeru uporabe algoritmov *Naive Bayes* in *k-NN* znaša okoli 1.5, medtem ko je pri uporabi ostalih dveh algoritmov praktično enak na vseh treh platformah. Rezultat je, vsaj glede na razmerja pri postopkih generiranja modela, dokaj presenetljiv. Zelo opazna je tudi razlika med dvema generacijama naprav na mobilni platformi, saj je naprava z novejšim operacijskim sistemom in predvsem boljšimi strojnimi možnostmi glede na rezultate tudi do dvakrat hitrejša v primeru klasifikacije z algoritmoma *Naive Bayes* in *k-NN.*

Omenjeni rezultati kažejo na dejstvo, da se razkorak med običajnimi ter mobilnimi računalniškimi sistemi v smislu zmogljivosti vse bolj manjša. Razlika je opazna predvsem med razlikami v zmogljivosti različnih generacij omenjenih naprav. Poleg boljših strojnih lastnosti

napredujejo tudi sistemi, ki lahko te zmožnosti vse bolj učinkovito izkoriščajo. V zaključku druge faze testiranja lahko tako sklenemo, da so zmogljivosti mobilne platforme v določenih primerih dovolj učinkovite, da bi lahko na njih lahko izvajali nekatere tipe procesiranja, kot je v tem primeru omenjena končna klasifikacija uporabnikovih iskanih besed. Če se ozremo po lastnostih kontekstno odvisne aplikacije je to zagotovo tudi akcija, ki bi se v njej tudi najpogosteje izvajala. Morebitno generiranje novega modela se namreč izvrši le takrat, ko bi v sistem uvozili nove podatke o restavracijah, ali pa bi želeli v proces generiranja modela vključiti, s strani uporabnikov, potrjene primere uspešnih primerov klasifikacij. Te bi namreč lahko izboljšale natančnost omenjenega modela pri klasifikacijah novih podatkov s strani istega uporabnika.

Pri dejanski uporabi klasifikacije smo opazili, da je uporabljen podatkovni set za določanje tipa zelo predvidljiv in da bi bilo potrebno iz njega izločiti nekatere označbe, kot sta tip restavracije in cena. S pomočjo tako reduciranega podatkovnega seta smo posledično dobili veliko slabše ocene natančnosti, kar nakazuje na to, da se klasifikacija omenjenega seta opira ravno na omenjena atributa. Pri izdelavi resnične aplikacije bi ta dejstva morali vzeti v obzir in glede na naše podatke izdelati ustrezen nabor označb, s katerimi bi lahko bolj učinkovito klasificirali restavracije.

7.3. Tretja faza – Analiza SWOT

Končna fazo evalvacije kontekstnega strežnika predstavlja analiza SWOT, ki se osredotoča ne štiri različne poglede. Za analizo implementacije kontekstnega strežnika bomo tako uporabili različne opise glede na aspekte prednosti, slabosti, priložnosti in nevarnosti predlagane arhitekture v primerjavi s klasično strežniško arhitekturo kontekstnega strežnika. Prvi dve obravnavata predvsem notranje dejavnike, medtem ko se preostali dve ukvarjata z zunanjimi dejavniki in vplivi [18].

7.3.1. **Prednosti**

Kontekstni strežnik na mobilni napravi zaradi prenosljivosti in vseprisotnosti naprav omogoča boljše pogoje za zajem celovitega konteksta nekega uporabnika, kot bi to lahko storili s katerikoli drugo, manj prenosno napravo. Ker prenosne naprave na veliko področjih že prevzemajo primat klasičnim, v preteklosti sicer bolje odzivnim in programsko podprtim računalniškim sistemov, se te uporabljajo bolj pogosto in več različnih situacijah, kar omogoča zajem večjih količin kontekstnih podatkov in s tem doseganje boljših rezultatov kontekstne

obdelave. Ker so te naprave v preteklosti, gledano z vidika tehnoloških zmožnosti, občutno napredovale, lahko kontekst procesiramo direktno na napravi sami. Zaradi te lastnosti kontekstni strežnik za svoje delovanje ne potrebuje vseskozne prisotnosti mobilne ali brezžične povezave, saj deluje neodvisno. Kljub temu, da so se standardi brezžičnih komunikacij do danes že občutno razvili in dosegajo že veliko hitrosti, pa se njihova pokritost iz prehajanja iz kraja v kraj lahko občutno razlikuje in je v določenih bolj ali manj odročnih predelih premalo zanesljiva za vseskozno kontekstno procesiranje. Hkrati je potrebno takšno povezavo tudi plačevati, problematično pa je lahko tudi prehajanje iz države v državo zaradi spremenljivih cen gostovanj. Hkrati je lahko udaru tudi varnost teh omrežij, saj se ti podatki večinoma nezaščiteni prenašajo preko omrežij in so tako lahko tarča različnih napadov, kot sta prisluškovanje, spreminjanje in potvarjanje podatkov.

Ker so podatki pri mobilnem kontekstnem strežniku vseskozi prisotni le na napravi in se v osnovni implementaciji prenašajo samo med aplikacijami, je s tem delno izboljšana zaupnost kontekstnih podatkov in predvsem izpeljanih informacij, ki imajo veliko večjo informacijsko vrednost.

7.3.2. **Slabosti**

Med razvojem in implementacijo arhitekture smo naleteli tudi na nekaj slabosti, ki so lahko posledica lastnosti programske platforme, tehnoloških značilnosti mobilnih naprav, specifik izbranih modulov in tehnologij ter napak v shemi ali zasnovi arhitekture. Omenjenim smo se skušali v veliki meri izogniti ali jih vsaj zaobiti, medtem ko nekatere, vsaj v sedanjosti, še vedno predstavljajo prevelik tehnološki izziv.

Ena od slabosti kontekstnega strežnika je prav gotovo zelo velik nabor kontekstnih virov podatkov, ki se od vira do vira ter od naprave do naprave lahko zelo močno razlikujejo. Večino od teh virov se da na določeni stopnji še generalizirati, medtem ko bi za specifične aplikacije neobstoj določenega senzorja območje lahko močno omejil ali celo preprečil kontekstno delovanje takšne aplikacije.

Po pregledu rezultatov naprednega kontekstnega procesiranja v naši testni aplikacijami smo lahko tudi jasno videli, da še vedno obstaja razkorak med zmogljivostmi klasičnih in mobilnih sistemov. V primerjavi s strežniškimi bi bil ta še večji. Bolj kot zmogljivost pa je problematična prav poraba energije. Sodobne mobilne naprave so zelo napredovale v tehnološkem smislu, medtem ko na področju razvoja baterij v smislu boljše kapacitete še ni bilo nekih izboljšanih konceptov. Za veliko uporabnikov je avtonomija baterije ena od glavnih lastnosti, po kateri ocenjujejo uporabnost neke mobilne naprave.

7.3.3. **Priložnosti in nevarnosti**

Pri oceni priložnosti naše implementacije moramo upoštevati predvsem to, da strežnik sam nima velike vrednosti, če ga aplikacije ne podpirajo. Naša strežnik bi lahko bil konkurenčen v primeru, da bi na trgu obstajal dovolj velik naborom uporabnih in zmogljivih aplikacij, ki bi tak strežnik uporabljale in na tak način izboljšale svoje delovanje ter se bolje prilagodile okolju in uporabniku. Na trgu je namreč že veliko aplikacij, ki za svoje delovanje uporabljajo kontekstno odvisne vire.

Takšen strežnik ima priložnost predvsem na raziskovalnem področju. Raziskovalcem omogoča, da raziskave določenega tipa konteksta raje usmerijo na podatke in ne na implementacijo. S pomočjo aplikacij, ki bi bile v sklop takšnih raziskav razvite, bi lahko bolje ocenili potencialne priložnosti takšnega koncepta. Zelo verjetno bi bilo potrebno v tem primeru kodo izdati pod eno izmed odprtokodnih licenc, saj bi uporabniki tako lahko v sistem dodajali nove vire konteksta. V nasprotnem primeru bi lahko obstajala nevarnost, da bi bila implementacija takšnega kontekstnega strežnika preveč omejujoča.

Kot morebitne nevarnosti bi lahko izpostavili problem osrednjega hranjenja podatkov o uporabniku. Ta je sicer pogojena s samo varnostjo osrednjih komponent sistema Android, ki jih strežnik uporablja, vseeno pa bi se lahko potencialni napadalec zaradi znanih lastnosti strežnika osredotočil ravno nanj. Problematične bi lahko bili novi morebitni zakoni o zbiranju podatkov o uporabnikih, ki so se začeli predlagati po nekaterih aferah zaradi prikritega zbiranja, hranjenja in trgovanja s podatki, v katerih so bile udeležene tudi nekatere večje korporacije.

Že v pregledu obstoječih kontekstnih strežnikov je bilo omenjeno, da na tem področju ni veliko konkretnih implementacij, zato je tudi takšen kontekstni stroj v omenjenih aspektih težko primerjati s konkurenco. Prav gotovo pa je na trgu zelo veliko aplikacij, ki uporabljajo kontekstno odvisne vire, a je njihov način pridobivanja in obdelave teh virov dosežen na drugačen način.

8. ZAKLJUČEK

Kontekstno delovanje aplikacij obravnava uporabo različnih podatkov v namene izboljšanja njihovega delovanja in nudenju večjega nabora storitev, glede na posameznega uporabnika in njegovo okolico. V kolikor se pri osredotočimo na le določen tip podatkov, ki se nam v nekem trenutku zdi relevanten, lahko pri tem izgubimo kontekst, ki ga prej niti nismo imeli namena zaznavati. Kontekstno odvisne aplikacije niso namenjene temu, da bi jih uporabnik nadzoroval ter spreminjal, ampak morajo težiti k samodejnemu delovanju ter ponujanju informacij za katere še sam uporabnik v določeni situaciji ne ve, da jih res potrebuje. Takšna implementacija ima veliko lastnosti inteligentih agentov, ki sprejemajo informacije in vplivajo na okolje. Za implementacijo takšne aplikacije bi bilo v njen razvoj, poleg že omenjene, potrebno vključiti še veliko različnih strok in ostalih področij.

V začetnem delu naloge smo skušali razjasniti pomen konteksta ter omeniti nekaj njegovih osnovnih lastnosti. Omenili smo nekaj načinov zaznave in zajema konteksta ter predstavili nekaj primerov srečevanja s primerki konteksta. Ta se namreč uporablja vsakodnevno pri naši interakciji z informacijskimi sistemi, prebiranju elektronske pošte, nakupih in drugje. Postopoma z razvojem družbe smo bili priča tudi razvoju novih tipov informacij, ki jim človek vse težje sledil in jih sčasoma ni mogel več zaznavati v celoti. Zato smo v nadaljevanju predstavili arhitekturo, ki je zmožna zaznavanja in procesiranja velikih količin konteksta. Te arhitekture smo povezovali z imenom kontekstni stroj, ki pa so večinoma del veliko večje celote za porazdeljeno procesiranje konteksta in serviranje informacij, ki jo imenujemo kontekstni strežniki. V tem delu tudi opišemo njihove glavne lastnosti in hkrati izpostavimo nekatere pomanjkljivosti.

Nato smo se osredotočili na pregled lastnosti mobilnih naprav, ki praviloma vseskozi spremljajo uporabnika in so v zadnjih letih na različnih področjih občutno napredovale. Posebno vlogo v tem poglavju imajo predvsem kontekstno odvisne aplikacije, ki zmorejo različne kontekstne podatke zajemati same in z njihovo pomočjo obogatiti svoje delovanje. Z uporabo nekaterih osnovnih lastnosti teh aplikacij smo v naslednjem poglavju predstavili idejo in ustvarili shemo lastnega modela kontekstnega strežnika, ki bi deloval na mobilni napravi in bi omogočal delovanje različnih kontekstno odvisnih aplikacij.

V naslednjem poglavju smo omenjeno shemo kontekstnega strežnika implementirali v okviru mobilne platforme Android in opisali posamezne korake implementacije strežnika Ta je potekala v treh fazah, nato pa je sledil še opis razvoja posebne kontekstno odvisne aplikacije. Ta poleg zajema konteksta iz osnovnih virov, kot so senzorji, podatki o nameščenih aplikacijah, podatki o omrežju, in drugih, obravnava še uporabo podatkov o uporabnikovi interakciji s

sistemom. S pomočjo te aplikacije smo tako na mobilni platformi preizkusili metode tekstovne klasifikacije, ki smo jo implementirali z uporabo odprtokodnih razredov orodja Weka.

Sledila je evalvacija posameznih delov kontekstnega strežnika in pripadajoče kontekstno odvisne aplikacije. Evalvacija je potekala v treh korakih, ki so obsegali standardno testiranje delovanja kontekstnega strežnika in aplikacije, pri tem pa smo postopke tekstovne klasifikacije tudi ovrednotili z vidika zmogljivosti mobilne arhitekture v primerjavi s klasično računalniško arhitekturo. Končni korak evalvacije je obsegal postopke analize SWOT, s katerimi smo opravili oceno uporabnosti, njenih izrazitih pomanjkljivosti in morebitnih priložnostih v prihodnje.

Po rezultatih zmogljivostnih testiranj je bilo videti, da se takšna arhitektura še ne more v celoti primerjati z veliko kompleksnejšo in zmogljivejšo arhitekturo odjemalec-strežnik. Ta zmore procesirati veliko več podatkov naenkrat in pri tem uporabiti zelo obsežne zbirke preteklih podatkov. Omenjena arhitektura je imela nekaj težav tudi s samo porabo energije, saj je celodnevno zaznavanje in obdelovanje kontekstnih podatkov močno vplivalo na avtonomijo naprave v primerjavi z običajno uporabo. Vseeno pa je zelo pozitivno presenetilo učinkovito deljenje podatkov med aplikacijo in kontekstnim strežnikom, ki je delovalo zelo zanesljivo in avtonomno.

Glede na omenjena dejstva lahko podamo zaključek, da je predlagana arhitektura kontekstnega strežnika namenjena procesiranju manjše količine podatkov, ki niso kritičnega pomena. Hkrati bi lahko takšno implementacijo uporabili kot nekakšen rezervni sistem, ki bi lahko prevzel vlogo pomožne arhitekture takrat, ko naprava ne bi mogla dostopati do storitev glavnega kontekstnega strežnika. Kljub mnogim člankom in raziskavam je to področje še vedno premalo raziskano in preveč usmerjeno na točno določen tip delovanja. Večina omenjenih prototipov aplikacij tudi ni na voljo za testiranje. Zasnova naše arhitekture tako predstavlja primer preprostega kontekstnega strežnika na mobilni napravi, ki za svoje delovanje ne potrebuje posebnih namestitev strojne in programske opreme ter omogoča gradnjo različnih kontekstno odvisnih aplikacij.

V prihodnosti bi lahko takšen kontekstni strežnik podpiral še več virov kontekstnih podatkov. Naprave imajo sicer vgrajenih veliko senzorjev, vendar večina od teh nima lastnih koprocesorjev, ki razbremenijo osnovno procesno enoto in omogočajo manjšo porabo energije v mirovanju. V bodoče bi se lahko posamezni kontekstni strežniki na različnih napravah v okolici tudi povezovali med seboj in tako omogočali še boljši pregled nad kontekstom okolice. Posamezni moduli za kontekstno procesiranje bi se lahko javno objavljali in bili na uporabnikovo zahtevo pripravljeni na namestitev. Namestitev bi lahko zahtevala tudi kontekstno odvisna aplikacija. Na tak način razvijalcem te aplikacije ne bi bilo potrebno definirati ločenega procesiranja v aplikaciji.

VIRI IN LITERATURA

[1] S. Komatineni, D. MacLean, *Pro Android 4*, New York: Apress, 2012, pogl. 1-5.

[2] B. N. Schilit, "A System Architecture for Context-Aware Mobile Computing", Doktorska disertacija, Columbia University, 1995, pogl. 2-5.

[3] A. K. Dey, " Understanding and Using Context", Georgia Institute of Technology, 2001.

[4] T. Hofer, W. Schwinger, M. Pichler, G. Leonhartsberger, J. Altmann, "Context-Awareness on Mobile Devices - the Hydrogen Approach", *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, 2003, pogl. 1-4.

[5] L. Oliveira, A. N. Ribeiro, J. C. Campos, "The Mobile Context Framework: providing context to mobile applications", *Distributed, Ambient, and Pervasive Interactions*, 2013.

[6] R. Rupnik, C. Laskovsky, M. Suga, "Specifikacija dostave relevantnih podatkov", UL FRI/CHS, 2012, pogl 1-5.

[7] G. Kapitsaki, G. Prezerakos, Tselikas c, I. S. Venieris, "Context-aware service engineering, A survey", *The Journal of Systems and Software 82*, 2009.

[8] (2011) Rules Engine on mobile platform. Dostopno na : Getting Rule engine to work on Mobile Platform. Dostopno na: http://tech-voyage.blogspot.com/2011/06/getting-rule-engine-to-work-on-mobile.html.

[9] T. Ruotsalo, K. Haav, A. Stoyanov, S. Roche, E. Fani, R. Deliai, E. Mäkelä, T. Kauppinen, E. Hyvönen, "SMARTMUSEUM: A mobile recommender system for the Web of Data", *Web Semantics: Science, Services and Agents on the World Wide Web*, 2013.

[10] S. Abbate, M. Avvenutia, F. Bonatesta, G. Cola, P. Corsinia, A. Vecchioa, "A smartphone-based fall detection system", Dip. di Ingegneria del Informazione, University of Pisa, avg. 2012, pogl. 1-7.

[11] J. Y.-K. Yau, M. Joy, "An adaptive context-aware mobile learning framework based on a usability perspective", *International Journal of Mobile Learning and Organisation*, zv. 10, št. 10, 2010, pogl. 1-4.

[12] I. Kononenko, M. R. Šikonja, *Inteligentni sistemi*, 1. izd., Ljubljana: Založba FE in FRI, 2010, pogl. 1-12.

[13] D. Jurafsky, J. H. Martin, *Speech and Language processing*, 2. izd., New Jersey: Prentice Hall, 2009, pogl 2-6, 22.

[14] (2014) Weka 3: Data Mining Software in Java. Dostopno na: http://www.cs.waikato.ac.nz/ml/weka/.

[15] (2010) Rjmarsan.: Weka-for-Android. Dostopno na: https://github.com/rjmarsan/Weka-for-Android.

[16] (2014) Text categorization with WEKA. Dostopno na: http://weka.wikispaces.com/Text+categorization+with+WEKA.

[17] R. Burke, K. J . Hammond, B. C. Young, "The FindMe Approach to Assisted Browsing", *IEEE Expert, Volume 12, Issue 4*, 1997.

[18] (2014) SWOT Analysis: Strengths, Weaknesses, Opportunities, and Threats. Dostopno na: http://ctb.ku.edu/en/table-of-contents/assessment/assessing-community-needs-and-resources/swot-analysis/main.

KAZALO SLIK