Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Klemen Kresnik

Uporaba openEHR na mobilnih napravah

diplomsko delo

univerzitetni študijski program računalništvo in informatika

Mentor: dr. Andrej Brodnik

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

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

Št. naloge: 01962 / 2013
Datum: 15.9.2013

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

Kandidat: **KLEMEN KRESNIK**

Naslov: **UPORABA OPENEHR NA MOBILNIH NAPRAVAH**
USE OF OPENEHR ON MOBILE PLATFORMS

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Osrednji podatek v medicinski informatiki predstavlja osebni zdravstveni karton (Personal Health Record). V njem so shranjeni podatki o zdravstvenem stanju in on zdravniških posegih posameznega pacienta. Da bi lahko postalo zdravljenje v resnici vseprisotno je potrebno izpolniti predvsem dva pogoja: karton je potrebno digitalizirati in digitaliziranega je potrebno shraniti v standardnem zapisu, kar bo omogočalo interoperabilnost. V diplomski nalogi preučite možnost zapisa zdravstvenih podatkov v zapisu openEHR še posebej s poudarkom na možnosti rabe na mobilnih napravah. Slednje zahtevajo po eni strani delovanje z ne velikim pomnilnikom in po drugi strani tudi brez povezave z osrednjim strežnikom v oblaku. Pilotno preverite možnost rabe openEHR zapisa na mobilni napravi na primeru uporabe vzdrževanja stanja pacienta v aplikaciji eŠport, ki uporablja ogrodje eOskrba.

Mentor:

doc. dr. Andrej Brodnik

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Klemen Kresnik, z vpisno številko **63080012**, sem avtor diplomskega dela z naslovom:

Uporaba openEHR na mobilnih napravah

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 11. januarja 2014 Podpis avtorja:

Zahvaljujem se staršem, ki so me podpirali in vzpodbujali skozi celoten študij, mentorju dr. Andreju Brodniku za svetovanje in zelo korekten odnos ter Matetu Bešteku za vso pomoč pri razumevanju spletne aplikacije in nasplošno informatike v zdravstvu.

Kazalo

Povzetek

Abstract

Seznam prevodov in uporabljenih kratic

1 Uvod **1**

1.1 Interoperabilnost 2
1.2 Ontologije 3
1.3 Elektronska zdravstvena kartoteka 3
1.4 Cilji in struktura diplomskega dela 6

2 OpenEHR **9**

2.1 Uvod 9
2.2 Referenčni model 11
2.3 Arhetipi 12
2.3.1 Datoteke ADL 13
2.3.2 Predloge 16
2.4 Vnos podatkov 19
2.5 Dostop do podatkov 21
2.6 Terminologija 21
2.7 Sorodne rešitve 22

3 Android **25**

3.1 Arhitektura platforme 25

3.2 Navidezni stroj 26
3.3 Razvojna okolja 27
3.4 Tehnologije Android 28

4 Ostale tehnologije **33**
4.1 Spletna aplikacija eOskrba 33
4.2 eŠport intervencija 33
4.3 Podatkovne baze 34
4.4 Razpoznava podatkov 35
4.5 Ostalo 36

5 Mobilna aplikacija **37**
5.1 Struktura aplikacije 38
5.2 Prikaz seznama nalog 39
5.3 Validacija in oddajanje meritev 42
5.3.1 Preimenovanje paketov 44
5.3.2 Beleženje dogodkov 45
5.3.3 Zmanjšanje velikosti aplikacije 45
5.3.4 Manjkajoča implementacija vtičnika GSS-API 47
5.4 Shranjevanje podatkov 48

6 Rezultati **49**
6.1 Meritev velikosti kopice 49
6.1.1 Postopek 50
6.1.2 Rezultati 51
6.2 Zajem stanja kopice 51
6.2.1 Postopek zajema 52
6.2.2 Rezultati 53
6.3 Obrazložitev in komentarji rezultatov 53

7 Zaključek **55**
7.1 Primeri izboljšav 55

Dodatek A Definicija arhetipa **57**

Dodatek B Arhetip **61**

Dodatek C Konfiguracijska datoteka ProGuard **67**

Dodatek D Datoteka Android.mk **73**

Povzetek

Interoperabilnost ali zmožnost izmenjave informacij predstavlja enega izmed največjih problemov tovrstne komunikacije med informacijskimi sistemi pri informatiki v zdravstvu. Problem interoperabilnosti rešujemo z vpeljavo standardov, ki poenotijo izmenjavo informacij. Trenutno je na voljo več standardov med njimi tudi openEHR, ki standardizira uporabo elektronskih zdravstvenih kartotek. V današnjem času so močno v porastu pametne mobilne naprave in poleg strežniških sistemov predstavljajo nov segment informatike v zdravstvu. V diplomskem delu predstavimo postopek razvoja mobilne aplikacije, ki predstavlja samostojnega odjemalca za spletno aplikacijo eOskrba. Odjemalec omogoča vnos, validacijo in hranjenje kliničnih podatkov za eŠport intervencijo. Mobilna aplikacija enako kot spletna aplikacija uporablja standard openEHR, ki je zasnovan za strežniške sisteme. Skozi diplomsko delo je zato predstavljen tudi teoretični del standarda openEHR in postopek njegove postavitve na mobilni napravi. Zaradi zasnove za strežniške sisteme smo opravili tudi meritve porabe pomnilnika in s tem ocenili primernost uporabe openEHR na mobilnih napravah. Za mobilno platformo smo zaradi njene razširjenosti izbrali Android.

Ključne besede:

interoperabilnost, elektronska zdravstvena kartoteka, pametna mobilna naprava

Abstract

Interoperability or ability to exchange information represents one of the key issues in exchange of information between healthcare information systems. Problem of interoperability is solved through definition of standards which unify the exchange of information. There are multiple standards currently available, one of them is also openEHR. openEHR defines standard for usage of electronic health records. Besides server systems smart mobile phones are becoming more and more popular and represent new segment of healthcare informatics. In this thesis we describe the development of mobile application as a standalone client for web application eOskrba. Client provides ways of input, validation and persistence of clinical data for eŠport intervention. Similarly as web application, mobile application uses openEHR standard, which is by default intended for server systems. For that reason we describe through thesis theoretical part and also setup of openEHR on mobile devices. We also observed and measured memory usage of application and evaluated the suitability for integrating openEHR on mobile devices. For mobile platform we used Android because of it's widespread use.

Key words:

interoperability, electronic health record, smart mobile device

Seznam prevodov in uporabljenih kratic

ADL (*angl. Archetype Definition Language*) jezik za definiranje arhetipov. Datoteke s končnico ADL v openEHR predstavljajo arhetipe.

API (*angl. Application Programming Interface*) aplikacijski programski vmesnik predstavlja nabor funkcionalnosti, ki jih lahko razvijalci uporabijo med razvojem programske opreme.

BPMN (*angl. Bussiness Process Modelling and Notation*) standard za modeliranje poslovnih procesov.

CCD (*angl. Concept Constraint Definition*) omejitev modela domene v standardu MLHIM.

CLASS datoteka, ki vsebuje ukaze izvajanja programa za navidezni stroj Java.

DEX (*angl. Dalvik Executable*) datoteka, ki vsebuje ukaze izvajanja programa za navidezni stroj Dalvik .

DM (*angl. Domain Model*) model domene v standardu MLHIM.

DSL (*angl. Domain Specific Language*) jezik, specifičen za določeno domeno. Največkrat so to programski jeziki ustvarjeni znotraj točno določene domene.

EHR (*angl. Electroic Health Record*) elektronska zdravstvena kartoteka.

JAR (*angl. Java Archive*) datoteka, ki predstavlja programsko knjižnico v programskem okolju Java.

JSON (*angl. JavaScript Object Notation*) notacija, ki omogoča strukturiran zapis podatkov.

NDK (*angl. Native Development Kit*) nativno razvojno okolje.

MLHIM (*angl. Multi Level Healthcare Information Modelling*) večnivojsko modeliranje informacij v zdravstvu.

OET končnica datotek, ki predstavljajo predlogo v openEHR.

ORM (*angl. Object Relational Mapping*) tehnika programiranja za pretvorbo podatkov iz različnih tipov v predmetno usmerjene strukture.

RM (*angl. Reference Model*) referenčni model v standardu openEHR.

SO (*angl. Shared Object*) deljeni objekt predstavlja deljeno programsko knjižnico, ki si jo lahko deli več programov.

SQL (*angl. Structured Query Language*) strukturirani poizvedovalni jezik.

SDK (*angl. Software Development Kit*) programsko razvojno okolje.

XML (*angl. Extensible Markup Language*) označevalni jezik, ki omogoča zapis podatkov v strukturirani obliki.

XSD (*angl. XML Schema*) shema, ki opisuje strukturo dokumenta XML.

Flatten, flattening postopek izravnave.

Parse, parsing postopek razpoznave, pridobitev podatkov iz notacije

Programming library programske knjižnice predstavljajo skupek programske kode, ki opravlja neko izbrano nalogo. S tem ni treba v vsakem programu naloge ponovno programirati ampak namesto tega uporabimo programsko knjižnico.

Device driver gonilniki, so programi, ki znajo upravljati in komunicirati s strojno opremo.

Poglavje 1

Uvod

V sodobnem času so pametne mobilne naprave del vsakdanjika. Spremljajo nas na vsakem koraku. Nekaj izmed glavnih prednosti so neprekinjena povezljivost s spletom, številni vgrajeni senzorji in zmogljivost, ki se lahko primerja že s šibkejšimi osebnimi računalniki.

Pametni mobilni telefoni prinašajo korenite spremembe v načinu poslovanja, plačevanja, komunikacije, logistike, zdravstvene oskrbe in še bi lahko naštevali. Razširjenost pametnih mobilnih naprave iz dneva v dan raste.

Trenutno je najpriljubljenejša mobilna platforma Android, s tržnim deležem, ki znaša več kot 70% ([12]). Menimo da operacijski sistem Android predstavlja potencialno področje kamor bi lahko razširili zdravstveno oskrbo. Na trgu se že pojavljajo mobilne aplikacije, ki nudijo svetovanje s kliničnim osebjem, bazo znanja o boleznih in simptomih.

Skozi to poglavje se najprej seznanimo s pojmom interoperabilnosti. Kaj interoperabilnost je, na katere kategorije jo razdelimo, zakaj je pomembna v informatiki v zdravstvu. Nadaljujemo z ontologijami in kakšno vlogo imajo v informatiki v zdravstvu. Na koncu predstavimo še elektronsko zdravstveno kartoteko, obstoječe rešitve, pogoje uspešne interoperabilnosti le-teh ter na kratko izpostavimo probleme elektronskih zdravstvenih kartotek.

1.1 Interoperabilnost

Interoperabilnost je možnost izmenjave podatkov dveh ali več informacijskih sis temov v različnih tehnologijah in z različno programsko opremo. Izmenjava podatkov mora biti zanesljiva in konsistentna. Informacijski sistemi morajo prejete informacije znati ustrezno interpretirati in uporabiti ([7]).

Eden izmed ključnih problemov pri informatiki v zdravstvu je pomanjkanje interoperabilnosti med informacijskimi sistemi. Razdelimo jo lahko na dve glavni kategoriji: semantična interoperabilnost in sintaktična interoperabilnost ([7]).

Sintaktična interoperabilnost je definirana kot komunikacijska plast, ki omogoča izmenjavo informacij med dvema ali več sistemi. Sestavljena je iz sledečih plasti ([7]):

- omrežje in transportna plast,
- aplikacijska plast,
- protokol sporočanja in format sporočil.

Sintaktična interoperabilnost zagotavlja, da bo sporočilo, ki je bilo poslano, tudi uspešno dostavljeno, vendar pa ne zagotavlja, da bo uspešno in pravilno interpretirano. Pravilna interpretacija sporočil je definirana s semantično interoperabilnostjo.

Semantična interoperabilnost je zagotovljena, ko so sporočila, ki si jih sistemi med seboj izmenjujejo, razumljena na stopnji formalno definiranih konceptov znotraj domene. Semantika so metapodatki, ki s pomočjo ontologij opisujejo podatke ([7]).

Problem interoperabilnosti lahko razdelimo v dve kategoriji:

1. Interoperabilnost izmenjave sporočil med sistemi: Komunikacija poteka prek vmesnikov, pri čemer ima skoraj vsak sistem svoj vmesnik, prek katerega lahko drugi sistemi komunicirajo. Posledično mora imeti

vsak sistem vmesnik za vse sisteme, s katerimi komunicira. To pomeni, da je lahko celotno število vmesnikov, ki jih mora sistem podpirati, O($\frac{n*(n+1)}{2}$) za n sistemov oziroma aplikacij. Za reševanje tega problema je bila uvedena standardizacija sporočil. Vsi sistemi sprejemajo in pošiljajo sporočila po vnaprej dogovorjenem standardu oziroma vsi uporabljajo enak vmesnik. V domeni zdravstva je trenutno najbolj uveljavljen vmesnik za sporočanje HL7 v2 (Health Level 7 Version 2 Messaging Interface) ([7]).

2. Interoperabilnost elektronskih zdravstvenih kartotek. Poglavje 1.3.

1.2 Ontologije

Semantični splet je tehnologija, pri čemer je vsaki informaciji dodeljen točno določen pomen,kar računalniškim strojem omogoča lažje procesiranje in integracijo informacij. Ena izmed temeljnih tehnologij znotraj semantičnega spleta so ontologije ([13]). Ontologija je formalen nedvoumen podroben opis pojmov, ki jih priznava širša množica strokovnjakov na področju neke izbrane domene. Z drugimi besedami predstavlja skupen slovar znotraj domene ([7]).

Formalnost zagotavlja, da je pomen opisa podan v jeziku, ki ga stroji razumejo, t.i. jezik ontologije. Nedvoumen opis zagotavlja, da so vsi koncepti in razmerja v abstraktnem modelu definirani s konkretnimi imeni in definicijami. Ontologije dajejo pomen informacijskim strukturam, ki si jih izmenjujejo informacijski sistemi in so ena izmed ključnih tehnologij za zagotavljanje semantične interoperabilnosti informacijskih sistemov v zdravstvu ([7]).

1.3 Elektronska zdravstvena kartoteka

Elektronska zdravstvena kartoteka ima zelo pomembno vlogo pri razvoju zdravstva. S prihodom interneta vedno več pacientov išče informacije o

zdravljenju tudi na spletu, hkrati pa želijo biti čimbolj udeleženi med svojim zdravljenjem. Elektronske zdravstvene kartoteke so na voljo prek spleta in omogočajo pacientu dostop do vseh zdravstvenih ter kliničnih informacij kjerkoli in kadarkoli.

Elektronsko zdravstveno kartoteko lahko definiramo kot digitalno shrambo informacij o pacientovem zdravstvenem stanju skozi celotno življenjsko obdobje. Te informacije so lahko na različnih neodvisnih institucijah. Da je lahko zagotovljena primerna zdravstvena oskrba, mora imeti klinično osebje dostop do celotne zgodovine pacientovega zdravstvenega stanja. Za doseganje tega je treba to hranjenje standardizirati in s tem omogočiti interoperabilnost. Trenutni standardi so ISO EN 13606 ([9]), openEHR ([17]) in HL7CDA ([11])([7]).

Implementacija kartotek mora zagotavljati dolgoročno rešitev, ki je odporna proti izgubi podatkov med prenosi in pretvorbo podatkov. Rešitev mora zagotavljati, da so vnosi podatkov natančni in pravilni. Kartoteke vsebujejo zaupne osebne podatke in informacije, zato mora biti poskrbljeno, da so pacientovi zdravstveni podatki ustrezno zaščiteni ter da njegova zasebnost ostane neogrožena. Podprti morata biti globalna in lokalna interoperabilnost. Številni pacienti se v življenju preselijo, še več jih redno potuje, zato je pomembno, da so kartoteke dostopne s katerekoli lokacije ([19]).

Za interoperabilnost elektronske zdravstvene kartoteke morajo biti rešeni sledeči tehnični problemi ([7]):

1. Povezovanje pacientovih identifikatorjev: Glavni problem pri dostopu do kartoteke je pacientov identifikator, pri čemer lahko različne institucije uporabljajo različne identifikatorje za istega pacienta. Možne rešitve so:

    - centralna podatkovna baza z vsemi identifikatorji pacienta;
    - pametna kartica, ki vsebuje pacientove identifikatorje;
    - glavni indeks, ki medsebojno povezuje identifikatorje v različnih sistemih.

2. Preverjanje pristnosti (avtentikacija) uporabnikov: Pristnost uporabnikov mora biti zagotovljena ne samo v lastni domeni, ampak v celotni strukturi sistemov.

3. Treba je zagotoviti časovno sinhronost: Za pravilno delovanje porazdeljenih aplikacij je ključnega pomena, da so vse sistemske ure računalnikov v sistemu sinhronizirane.

4. Zagotovljena hierarhija dostopa in beleženje vseh aktivnosti v sistemu: Preverjanje pristnosti uporabnika ni dovolj. Omejiti je treba dostop med posameznimi deli sistema in posameznimi nastavitvami oskrbe.

Problemi elektronskih zdravstvenih kartotek

Etični problemi Uvedba elektronskih zdravstvenih kartotek lahko znatno izboljša interoperabilnost institucij in kvaliteto zdravstvene oskrbe. Vendar pa prinaša tudi nekaj problemov. Celotna zdravstvena kartoteka bi bila na voljo na spletu in s tem omogoča dostop tretjim osebam do zdravstvenih podatkov pacientov, kjer je lahko ogrožena pacientova zasebnost. Dostop do teh podatkov lahko vodi do diskriminacije med zaposlenimi, sramu in pristranskosti zavarovalnic. Še posebno problematično je deljenje kartotek med sistemi, saj obstaja možnost uhajanja podatkov ([6]).

Družbeni problemi Možnost deljenja kartotek predstavlja tudi družbene probleme, namreč pacienti lahko delijo informacije z družinskimi člani in ostali institucijami, ki zagotavljajo zdravstveno oskrbo. Veliko elektronskih zdravstvenih kartotek ima slabo podporo nadziranja delov kartotek, ki so oz. niso na voljo za deljenje. Na ta način je omogočen dostop entitetam na različnih nivojih,kar predstavlja še dodaten problem zasebnosti. Pacient lahko določene podatke skrije zdravstvenemu osebju in s tem vpliva na kvaliteto oskrbe ter zanesljivost zdravstvene kartoteke. Še posebno če je institucija, ki nudi oskrbo, pravno zavezana, da se mora zanašati na elek-

tronsko zdravstveno kartoteko. Opisan problem se pojavlja že sedaj, vendar odsotnost fizičnega stika med pacientom in oskrbnikom ta problem še poslabša. Težko je tudi definirati pravilno mero izpostavljanja podatkov. Namreč mlajše populacije pacientov so veliko manj občutljive in lažje delijo več informacij kot starejše ([6]). Naslednji problem se pojavi v povezavi z družinskimi člani oziroma sorodniki. Namreč ko delimo podatke o svojem zdravstvenem stanju, lahko posledično delimo tudi zdravstvene podatke o svojih sorodnikih. Problem se bo še poslabšal, ko bodo imele genetske informacije v elektronskih zdravstvenih kartotekah še večji poudarek.

Pravni problemi Elektronske zdravstvene kartoteke še vedno nimajo točno definiranega prava in so vprašanja glede pričakovanj, pravic in dolžnosti pacienta ter institucij, ki nudijo zdravstveno oskrbo, še vedno neodgovorjena. V projektih, ki hranijo in obdelujejo občutljive osebne podatke, je varnost teh podatkov ključnega pomena. Zagotavljanje varnosti elektronskih zdravstvenih kartotek predstavlja velik izziv, namreč težko je narediti enostavno uporaben sistem, hkrati pa zagotoviti ustrezno zaščito. Razpršenost podatkov in zagotovitev dostopa različnim institucijam še dodatno oteži problem učinkovitega dostopa do elektronskih zdravstvenih datotek ([6]).

1.4 Cilji in struktura diplomskega dela

V tem delu na kratko obrazložimo cilje in strukturo diplomskega dela.

Cilji Cilj diplomskega dela je bila mobilna podpora spletne aplikacije eOskrba na mobilnih napravah Android. V ta namen bi razvili mobilno aplikacijo. Z nameščeno mobilno aplikacijo bi naprava predstavljala samostojnega odjemalca in z njo bi lahko uporabnik kjerkoli vnašal klinične podatke. Validacija podatkov bi potekala na napravi. Podatki bi se shranjevali lokalno in bi se ob razpoložljivi povezavi poslali na strežnik. Za dosego teh ciljev je bilo dovolj, da smo podprli vsaj eno izmed intervencij eOskrbe. Izbrali smo intervencijo eŠport.

Struktura Struktura diplomskega dela je zasnovana tako, da bralca postopoma uvede v tematiko. Najprej so predstavljeni pojmi informatike v zdravstvu. Sledi natančnejša obrazložitev tehnologije openEHR, nato predstavitev platforme Android in preostale uporabljene tehnologije. Zatem je predstavljena mobilna aplikacija in obrazložen postopek razvoja aplikacije. Sledi postopek zajema meritev porabe pomnilnika ter predstavitev in komentarji dobljenih rezultatov. Na koncu diplomskega dela navedemo še nekaj možnosti za izboljšave mobilne aplikacije in tehnologije openEHR v povezavi z namembnostjo za mobilne naprave.

Poglavje 2

OpenEHR

V poglavju se seznanimo z glavnimi pojmi in delovanjem tehnologije openEHR. Predstavimo, kako rešuje problem interoperabilnosti elektronskih zdravstvenih kartotek, obrazložimo pojme referenčnega modela, arhetipa, predloge, terminologije ter omejitev in kakšna je povezava med njimi. Na koncu naštejemo še sorodne rešitve v primerjavi z openEHR.

2.1 Uvod

Informacijski sistemi, ki jih razvijamo, so namenjeni neki specifični domeni (v našem primeru zdravstveni domeni). Za pravilno izvedbo in delovanje sistema je potrebnega veliko posvetovanja s strokovnjaki na področju znotraj domene (v našem primeru klinično osebje). Pristop, ki uspešno rešuje problem semantične interoperabilnosti, je uporaba več- ali dvonivojskega modeliranja. Večnivojsko modeliranje odpravi problem posvetovanja in strokovnjake znotraj domene direktno vključi v razvoj. Vsebino, specifično za domeno, ustvarijo strokovnjaki znotraj domene in jo nato predajo razvijalcem programske opreme. Tako vsak opravi delo, ki ga najbolje pozna. S tem se zmanjša možnost za napake ter pospeši razvoj.

Tehnologija oziroma standard openEHR je odgovor na problem semantične interoperabilnosti elektronskih zdravstvenih datotek, ki so modelirane v dveh

nivojih. Prvi nivo predstavlja referenčni model, drugi nivo pa arhetipi in predloge.

OpenEHR je zasnovan povsem generično in bi ga lahko uporabili za kakršnokoli domeno oskrbe, naj si bo v veterini ali pa oskrba javne infrastrukture. Razlog za to so referenčni modeli, ki predstavljajo koncepte storitev in administracije v zvezi z oskrbo. Skozi arhetipe in predloge definiramo pojme, specifične za neko izbrano področje nege in oskrbe ([3]).

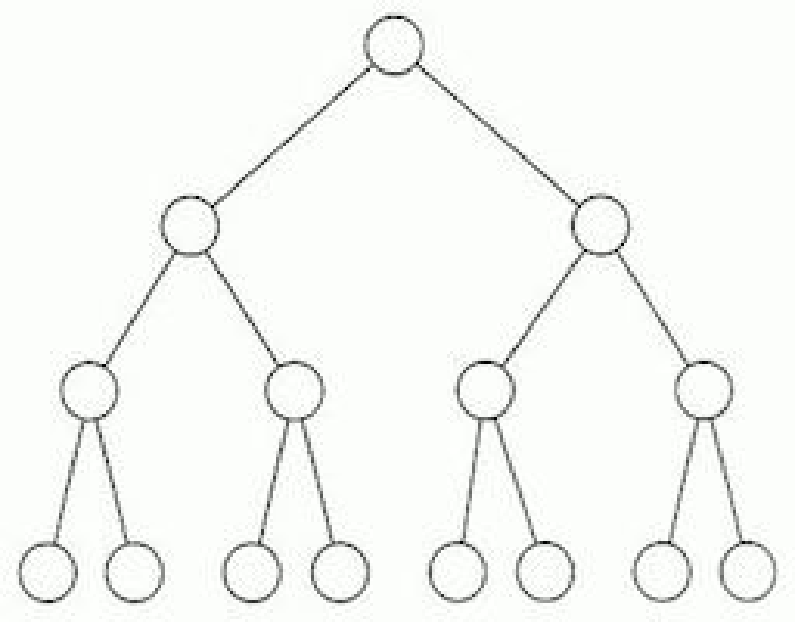
Pri snovanju informacijskega sistema oziroma elektronskih zdravstvenih kartotek je treba konkretneje definirati vse klinične koncepte, ki jih bodo kartoteke uporabljale in hranile, ter obrazce, prek katerih se bodo vnašali klinični podatki v kartoteke. Najverjetneje bo potekala tudi izmenjava kartotek, in če želimo, da bodo klinični podatki pravilno razumljeni in interpretirani v drugih institucijah, je treba podatke povezati z neko skupno terminologijo.

V openEHR klinične koncepte predstavljajo arhetipi, obrazce pa predloge. Definicije strukture arhetipov in predlog ustvari klinično osebje s pomočjo orodja ADL Workbench ([16]). Te so nato predane razvijalcem programske opreme, ki uporabijo definicije pri nadaljnji implementaciji, pri čemer razumevanje specifičnosti domene ni potrebno, saj je to namesto njih opravilo klinično osebje.

Povezava pojmov Preden pričnemo s podrobno obrazložitvijo posameznih pojmov, je bralcu smiselno predstaviti, kako orodje openEHR deluje kot celota. Iz referenčnih modelov s postavljanjem omejitev ustvarimo arhetipe. Ti predstavljajo en sam koncept, na primer telesno težo. Predloge združujejo arhetipe v nove arhetipe in omogočajo beleženje kliničnih dogodkov, na primer rezultate opravljene vadbe (telesna teža, počutje po vadbi, intenzivnosti vadbe). Pri izmenjavi arhetipov med institucijami je pomembno, da so klinični koncepti pravilno interpretirani (sistolični krvni pritisk pomeni sistolični krvni pritisk pri obeh institucijah). Namen terminologij je, da so koncepti med institucijami razumljeni nedvoumno. Vneseni podatki se shranijo v arhetip. Nato poteče validacija arhetipa, ki preveri, ali se podatki

skladajo z omejitvami. Če je validacija uspešna, se lahko arhetip shrani. Kasnejši dostop do podatkov v arhetipu omogoča posebna sintaksa poti in predikatov.

Drevesna podatkovna struktura drevo Drevo je abstraktna podatkovna struktura in je eden izmed načinov hierarhičnega strukturiranja podatkov v pomnilniku. Sestavljena je iz korenskega elementa in poddreves (razmerje starš otrok) . Vozlišča so med seboj ustrezno povezana. Primer drevesa je prikazan na spodnji sliki 2.1.



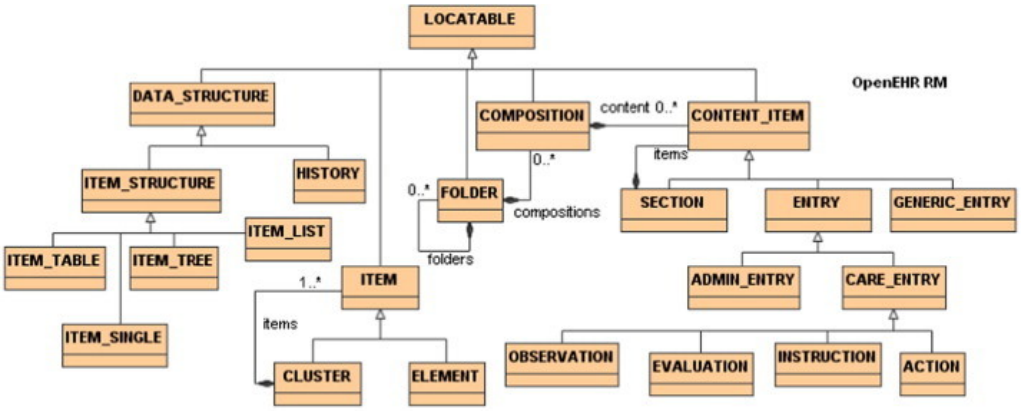
Slika 2.1: Primer drevesa ([10])

2.2 Referenčni model

Prvi nivo openEHR sestavlja referenčni model, ki je specifičen za domeno zdravstva, vendar še vedno zelo generičen. Sestavljen je iz množice razredov, ki se skozi čas ne smejo več spreminjati, in predstavljajo glavne gradnike elektronske zdravstvene kartoteke.

Zajema globalne značilnosti komponent zdravstvene kartoteke. Referenčni model nam s svojo generično strukturo omogoča nešteto različnih kombinacij modela. Vsebuje podporo za podatkovne tipe in strukture ter omogoča verzioniranje arhetipov (naprimer obstoječi arhetip nadgradimo in mu povečamo verzijo).

Shema 2.2 predstavlja drevo vseh razredov, vsebovanih v referenčnem modelu. Na primer: razred *COMPOSITION* lahko vsebuje nič ali več razredov *CONTENT_ITEM*. Razred *COMPOSITION* je lahko znotraj razreda *FOLDER* , pri čemer so lahko *FOLDER* znotraj samega sebe. Vsi razredi so podrazredi razreda *LOCATABLE*.



Slika 2.2: Struktura referenčnega modela ([2])

2.3 Arhetipi

Arhetipi predstavljajo klinične koncepte v obliki strukturiranih in s pravili omejenih kombinacij entitet iz referenčnega modela ([13]). Semantično je arhetip definicija kliničnega koncepta. Tehnično pa je referenčni model na katerem so postavljene omejitve in s tem izmed nešteto možnosti konfiguracije nastane le nekaj veljavnih konfiguracij. V pomnilniku so predstavljeni kot drevesa. Namenjeni so večkratni uporabi.

Zaradi interoperabilnost kartotek so klinični koncepti znotraj arhetipov

največkrat povezani z mednarodno priznano terminologijo. S tem pri izmenjavi arhetipov med institucijami ne pride do dvoumnosti razumevanja in interpretacije kliničnih konceptov.

Formalne definicije strukture (omejitve referenčnega modela) arhetipa so predstavljene kot *.adl* datoteke. Ustvarjene so z jezikom ADL. ADL je formalni jezik, sestavljen iz abstraktne sintakse namenjen kreiranju definicij novih arhetipov s postavljanjem omejitev nad referenčnim modelom ([13]).

2.3.1 Datoteke ADL

Definicija arhetipa sestoji iz štirih delov: glava *(header)*, opis *(description)*, definicija *(definition)* in ontologija *(ontology)*.

Opisana definicija arhetipa predstavlja klinični pojem opisne intenzivnosti gibanja. Celotna definicija arhetipa je na voljo v prilogi A.

- Glava:

```
archetype (adl_version=1.4)
  openEHR-EHR-CLUSTER.intenzivnost_opisno_5st_eo.v1

concept
  [at0000]  -- Intenzivnost opisno 5st eo
language
  original_language = <[ISO_639-1::sl]>
```

  Vsebuje ime arhetipa ter jezik, v katerem je arhetip napisan. V našem primeru je ime arhetipa *openEHR-EHR-CLUSTER.intenzivnost_opisno_5st_eo.v1*, jezik pa *ISO_639-1::sl*.

  Pravilo po katerem so vsi arhetipi poimenovani:

  - *openEHR* sistem, ki mu pripada referenčni model, iz katerega je arhetip narejen
  - *EHR* ime referenčnega modela iz katerega izhajamo
  - *OBSERVATION* ime referenčnega razreda znotraj referenčnega modela: iz tega razreda je bil arhetip ustvarjen.

  - *intenzivnost_opisno_5st_eo* kratek opis kliničnega koncepta
  - *v1* verzija arhetipa.

- Opis:

```
description
  original_author = <
    ["name"] = <"">
  >
  details = <
    ["sl"] = <
      language = <[ISO_639-1::sl]>
      purpose = <"Cluster za opis intenzivnosti pri
         portni/gibalni/telesni dejavnosti.">
      use = <"">
      misuse = <"">
      copyright = <"">
    >
  >
  lifecycle_state = <"0">
  other_contributors = <>
  other_details = <
    ["MD5-CAM-1.0.1"] = <"
       C7747FEC7B2B7B7AFF9046177BBEF58B">
  >
```

Opis vsebuje avtorja arhetipa, kakšen je namen arhetipa, ponovno je naveden jezik ter ostali posamezniki, ki so pomagali soustvarjati arhetip.

- Definicija:

```
definition
  CLUSTER[at0000] matches { -- Intenzivnost opisno 5st eo
    items cardinality matches {1..*; unordered} matches {
      ELEMENT[at0001] occurrences matches {0..1} matches
         {  -- Intenzivnost vadbe (5 stopenj)
        value matches {
          1|[local::at0002],  -- Zelo nizka intenzivnost
```

```
          2|[local::at0003],  -- Nizka intenzivnost
          3|[local::at0004],  -- Zmerna intenzivnost
          4|[local::at0005],  -- Visoka intenzivnost
          5|[local::at0006]   -- Zelo visoka intenzivnost
        }
      }
    }
  }
```

Definicija predstavlja najpomembnejši del, saj vsebuje omejitve nad referenčnim razredom. Posamezne izraze znotraj kliničnega koncepta označimo z *[at(število)]*. Kode nam nato omogočajo dostop do podatkov, ob definiciji terminologij v delu ontologij pa se na vrednosti sklicujemo preko teh kod. Zgornja definicija ima sledeč pomen: Struktura *CLUSTER* lahko vsebuje enega ali več struktur *ELEMENT*. Struktura *ELEMENT* pa lahko vsebuje eno izmed vrednosti *at0002*, *at0003*, *at0004*, *at0005* ali *at0006*.

- Ontologija:

```
ontology
  term_definitions = <
    ["sl"] = <
      items = <
        ["at0000"] = <
          text = <"Intenzivnost opisno 5st eo">
          description = <"unknown">
        >
        ["at0001"] = <
          text = <"Intenzivnost vadbe (5 stopenj)">
          description = <"*">
        >
        ["at0002"] = <
          text = <"Zelo nizka intenzivnost">
          description = <"Regeneracija (do 60 % FSmax).
             Med aktivnostjo je mogo sproen pogovor.">
        >
```

```
            ["at0003"] = <
              text = <"Nizka intenzivnost">
              description = <"Aerobno I (60 do 70 % FSmax).
                  Med aktivnostjo je mogo pogovor s krajimi
                  prekinitvami.">
            >
            ["at0004"] = <
              text = <"Zmerna intenzivnost">
              description = <"Aerobno II (70 do 80 % FSmax).
                  Med vadbo je pogovor oteen, pogovarja se
                  lahko le v zelo kratkih stavkih.">
            >
            ["at0005"] = <
              text = <"Visoka intenzivnost">
              description = <"Aerobno in anaerobno (80 do 90
                  % FSmax). Med vadbo je pogovor oteen,
                  pogovarja se lahko le v zelo kratkih stavkih
                  .">
            >
            ["at0006"] = <
              text = <"Zelo visoka intenzivnost">
              description = <"Preteno anaerobno (90 do 100 %
                  FSmax). Med vadbo pogovor praktino ni mogo
                  .">
            >
          >
        >
      >
```

Vsebuje definicije za vsak koncept in povezave z drugimi terminologijami. Na primer: Koncept *at0006* predstavlja zelo visoko intenzivnost gibanja.

2.3.2 Predloge

Predloga je koncept, ki v openEHR definira drevo enega ali več arhetipov. S pomočjo predlog lahko kombiniramo obstoječe arhetipe in s tem ustvarjamo

nove. Omogoča izpuščanje in preimenovanje določenih vozlišč znotraj arhetipov, ki predlogo sestavljajo, tako da končni nov arhetip ustreza določenemu kliničnemu dogodku. V praksi največkrat predstavljajo spletne obrazce ali natisnjena poročila ([4]).

Tehnično so predloge arhetipi z dvema posebnostma ([3]):

- lahko vključujejo druge arhetipe in predloge,
- lahko zbrišejo dele arhetipa

Podobno kot arhetipe, predloge definiramo z jezikom ADL v *.adl* datotekah. Navsezadnje so tudi predloge arhetipi. Zgradba definicije je enaka kot pri navadnem arhetipu (2.3.1). Programsko predloga uporabi vse arhetipe, ki jo sestavljajo, in iz njih sestavi nov arhetip, v pomnilniku predstavljen kot drevo. Slednje pomeni, da drevo arhetipa sestavljajo posamezni manjši arhetipi. Zato je potrebna še ena datoteka, ki definira poti do manjših arhetipov oziroma poti do podatkov, ki jih vsebujejo manjši arhetipi znotraj drevesa večjega arhetipa. Gre za datoteko XML s končnico *.oet*.

Primer definicije iz *.oet* datoteke (sledeča datoteka je bila uporabljena v mobilni aplikaciji pri obrazcu za oddajo rezultatov):

- Identifikacijska številka:

  ```
  <id>0de2847f-1750-4c08-91d3-7fc5ccb70ebf</id>
  ```

  Predstavlja identifikacijsko številko predloge.

- Ime predloge:

  ```
  <name>OpomnikEoSp</name>
  ```

- Opis:

  ```
  <description>
    <lifecycle_state>Initial</lifecycle_state>
    <details>
      <purpose />
      <use />
  ```

```
      <misuse />
    </details>
    <other_details>
      <item>
        <key>MetaDataSet:Sample Set </key>
        <value>Template metadata sample set </value>
      </item>
      <item>
        <key>User roles</key>
        <value />
      </item>
    </other_details>
  </description>
```

Vsebuje različne dodatne parametre, ki opisujejo predlogo. Zaradi lažje berljivosti smo opis skrajšali.

- Definicija:

```
<definition xsi:type="SECTION" archetype_id="openEHR-
   EHR-SECTION.opomnik_eo_sp.v1" concept_name="Opomnik␣
   eo␣sp">
  <Item xsi:type="OBSERVATION" archetype_id="openEHR-
     EHR-OBSERVATION.body_weight.v1" concept_name="
     Telesna␣teza" path="/items[at0001]" />
  <Item xsi:type="OBSERVATION" archetype_id="openEHR-
     EHR-OBSERVATION.telesna_dejavnost_enkratna_eo.v1"
     concept_name="Telesna␣dejavnost␣enkratna␣eo" path=
     "/items[at0006]">
  <Items xsi:type="CLUSTER" archetype_id="openEHR-EHR-
     CLUSTER.intenzivnost_opisno_5st_eo.v1"
     concept_name="Intenzivnost␣opisno␣5st␣eo" path="/
     data[at0001]/events[at0002]/data[at0003]/items[
     at0051]" />
  <Items xsi:type="CLUSTER" archetype_id="openEHR-EHR-
     CLUSTER.opis_pocutja.v1" concept_name="Opis␣
     pocutja" path="/data[at0001]/events[at0002]/data[
     at0003]/items[at0052]" />
```

```
    <Items xsi:type="CLUSTER" archetype_id="openEHR-EHR-
       CLUSTER.telesna_dejavnost_neopravljena_razlogi.v1"
        concept_name="Telesna␣dejavnost␣neopravljena␣
       razlogi" path="/data[at0001]/events[at0002]/data[
       at0003]/items[at0050]" /></Item>
  </definition>
```

Predstavlja najpomembnejši del predloge. V definiciji so našteti vsi arhetipi, ki sestavljajo predlogo, in za vsakega je navedena pot v drevesu novega arhetipa . ([4]). Na primer: Arhetip *openEHR-EHR-OBSERVATION.body_weight.v1* naj bo v novem arhetipu na poti */items[at0001].*

2.4 Vnos podatkov

Vneseni podatki se shranijo v vozlišča drevesa, ki predstavlja arhetip. Ob vnosu podatkov poteče sledeč postopek: ([19])

1. S postopkom razpoznave se v pomnilnik naložijo predloga in potrebni arhetipi. Razpoznava s pomočjo podatkov (iz *.adl* in *.oet* datotek) ustvari programske predmete predlog in arhetipov

2. S postopkom izravnave se predloga in arhetipi združijo v nov arhetip oziroma se ustvari drevesna struktura. Drevo za vsako vnosno polje v obrazcu vsebuje namensko vozlišče, v katero se shrani vnesen podatek

3. Ob kreiranju drevesa oziroma arhetipa se pobrišejo tudi odvečna vozlišča

4. Drevo arhetipa z vnesenimi podatki gre skozi postopek validacije, ki vrne seznam napak. Če je seznam napak prazen pomeni da so vsi podatki veljavni in je bila validacija uspešna

5. Drevo (arhetip) se pretvori v obliko permanentne notacije (na primer XML) in shrani

Primer shranjenega arhetipa v obliki XML, ki hrani intenzivnost opravljene vadbe:

```
<items archetype_node_id="openEHR-EHR-CLUSTER.
   intenzivnost_opisno_5st_eo.v1" xsi:type="v1:CLUSTER">
    <name>
        <value>Intenzivnost opisno 5st eo</value>
    </name>
    <items archetype_node_id="at0001" xsi:type="v1:ELEMENT">
        <name>
            <value>Intenzivnost vadbe (5 stopenj)</value>
        </name>
        <value xsi:type="v1:DV_ORDINAL">
            <value>2</value>
            <symbol>
                <value>Nizka intenzivnost</value>
                <defining_code>
                    <terminology_id>
                        <value>local</value>
                    </terminology_id>
                    <code_string>at0003</code_string>
                </defining_code>
            </symbol>
        </value>
    </items>
</items>
```

Pri vnosu kliničnih podatkov je trebe vnesene podatke preveriti, preden so shranjeni. Validacija prepreči shranjevanje napačnih kliničnih podatkov. Brez nje bi kartoteke lahko vsebovale nepravilne in netočne informacije. Posledično bi lahko to povzročilo nepravilno oskrbo. Validacija preveri, ali vnešeni podatki ustrezajo omejitvam, ki so definirane v arhetipu ([19]):

- ali so bila vsa obvezna polja izpolnjena,
- ali je tip podatka, ki je bil vnesen enak tipu podatka, ki je naveden v arhetipu,
- vnesen podatek je v dovoljenem območju, ki je definirano v arhetipu

2.5 Dostop do podatkov

Ker so predloge in arhetipi v pomnilniku predstavljeni v obliki dreves, openEHR vključuje mehanizem poti, ki omogoča vnašanje in dostop do podatkov znotraj vozlišč. Pot do vsakega vozlišča je mogoče navesti s potjo od vrha drevesa arhetipa do želenega vozlišča. Sintaksa poti je sledeča:

```
ime_atributa / ime_atributa [predikat]
```

Spodaj je primer poti s katero smo pri vnosu meritev v drevo vnesli telesno težo:

```
/items[openEHR-EHR-OBSERVATION.telesna_dejavnost_enkratna_eo.
    v1]/data[at0001]/events[at0002]/data[at0003]/items[openEHR
    -EHR-CLUSTER.opis_pocutja.v1]/items[at0001]/value
```

S tem primerom je lepo ponazorjena vloga predikata v poti. Na primer v poti */events* je lahko več konceptov, s predikatom *[at0002]* izberemo točno določeni koncept. Znotraj tega koncepta je lahko zopet naštetih več elementov.

2.6 Terminologija

Arhetipi podpirajo učinkovit način definicije pomenov kliničnih in sorodnih podatkov. Omogočajo povezovanje s številnimi terminologijami in slovarji v zdravstvu. Terminologije so v openEHR uporabljenje na sledeče načine ([2]):

- Vrednosti atributov v referenčnem modelu so definirane s terminologijo openEHR.
- Vsak arhetip vsebuje svojo terminologijo, ki definira pomen posameznega elementa.
- V arhetipu so lahko povezave z zunanjimi terminologijami.
- Arhetipi omogočajo izvajanje poizvedb na podlagi zunanjih terminologij.

Interna terminologija arhetipa Interna terminologija je vsebovana znotraj ontologije arhetipa. Uporaba samo interne terminologije prinaša sledeče prednosti ([2]):

- Prevod pojmov je znotraj nedvoumnega konteksta (vsak arhetip definira specifičen pojem).
- Določeni pojmi, ki so navedeni v arhetipih in niso vsebovani v velikih terminologijah.

Interna terminologija je v obliki {*koda, tekst, opis*} in predstavlja semantično definicijo posameznega vozlišča v arhetipu ([4]). Primer interne terminologije iz arhetipa, ki smo ga uporabili pri predstavitvi zgradbe:

```
["at0006"] = <
  text = <"Zelo visoka intenzivnost">
  description = <"Preteno anaerobno (90 do 100 % FSmax). Med
     vadbo pogovor praktino ni mogo.">
>
```

Povezava z zunanjimi terminologijami Interne kode arhetipa lahko povežemo s kodami zunanjih terminologij. Povezave so porazdeljene v skupine na osnovi zunanje terminologije, kar omogoča več povezav zunanje terminologije za eno interno kodo ([2]).

Arhetipi morajo biti povezani z zunanjo terminologijo zaradi deljenja in sklepanja. Ob izmenjavi informacij bodo institucije, ki uporabljajo enako terminologijo, nedvoumno interpretirale klinične koncepte ([3]).

2.7 Sorodne rešitve

MLHIM MLHIM temelji na paradigmi večnivojskega modeliranja. Vsebuje podporo za mobilne naprave in smo jo sprva poskušali uporabiti za razvoj naše mobilne aplikacije. Izkazalo se je da tehnologija še ni implementirana v nobenem programskem jeziku, temveč so definirane samo specifikacije.

Posledično bi bila za delovanje potrebna celotna implementacija standarda, kar pa je bila preobsežna naloga. Čeprav tehnologije nismo uporabili, ji je vredno nameniti nekaj besed. V splošnem MLHIM predstavlja izpopolnjen pristop tehnogije openEHR.

Podobnosti MLHIM z openEHR:

- vsebuje referenčne modele,
- s CCD omejitvami referenčnih modelov ustvarimo modele domene, ki predstavljajo analogijo z arhetipi

Prednosti MLHIM v primerjavi z openEHR ([5]):

- openEHR uporablja specifičen DSL (ADL) za modeliranje kliničnih konceptov. MLHIM namesto tega uporablja kar notacijo XML, kar poenostavi učenje in uporabo tehnologije;
- MLHIM ima manj kompleksno specifikacijo, saj minimizira semantiko v referenčnem modelu

Slabost standarda MLHIM je, da trenutno obstaja samo specifikacija, ne pa tudi implementacija, kar onemogoči uporabo tehnologije. To je bil glavni razlog zakaj standarda MLHIM nismo uporabili.

ISO EN 13606 Standard je podmnožica standarda openEHR. Definira samo sporočila za izmenjavo elektronskih zdravstvenih kartotek in ne celotnega sistema EHR. Celoten sistem EHR potrebuje podporo za verzioniranje in vmesnike za druge sisteme. Zaradi navedenih razlogov ga ni mogoče uporabiti za komunikacijo med ostalimi sistemi openEHR ali katerimikoli drugimi sistemi, ki želijo ohraniti informacije o verzijah podatkov ([21]). Aplikacija eOskrba temelji na standardu openEHR in zato zaradi zgornjih razlogov standarda ISO EN 13606 na mobilni aplikaciji nismo mogli uporabiti.

HL7 Clinical Document Architecture (CDA) HL7CDA je pristop standarda HL7 k reševanju interoperabilnosti EHR. Trenutno standard ne podpira arhetipov in ne vsebuje specifikacije za arhitekturo EHR. Podobno kot ISO EN 13606 ima poudarek samo na izmenjavi delov datotek ([21]). Klinično vsebino hrani dokument imenovan CDA Body pri čemer je struktura oziroma notacija vsebine znotraj dokumenta poljubna ([7]). Na ta način bi lahko uporabili openEHR in shranili vsebino XML v dokument CDA Body. S tem nebi nič pridobili, uporabljati pa bi morali dva sistema, enega znotraj drugega. Zaradi sledeča razloga uporaba standarda HL7CDA ni bila smiselna.

Poglavje 3

Android

V tem poglavju predstavimo mobilno platformo Android. Najprej se seznanimo z arhitekturo platforme in navideznim strojem, na katerem operacijski sistem teče. Nadaljujemo opis razvojnih okolij, sledijo pa tehnologije Android in orodja, ki smo jih uporabili v našem diplomskem delu.

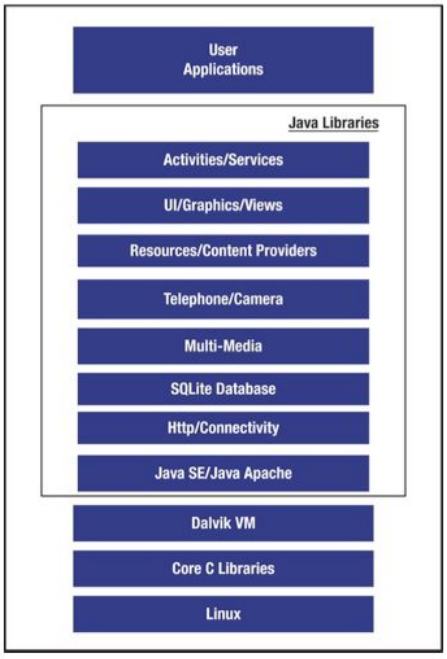
Android je operacijski sistem ustvarjen za mobilne naprave oziroma za naprave z zaslonom na dotik. Operacijski sistem temelji na jedru Linux, uporabniški vmesnik pa na neposredni interakciji (dotikanje) z zaslonom ([15]). Aplikacije so predstavljene kot datoteke APK ([1]). Razvoj aplikacij poteka v programskem okolju Java.

3.1 Arhitektura platforme

Arhitekturo platforme Android sestavlja ([27]):

- jedro Linux, ki vsebuje gonilnike za komunikacijo programske opreme z različnimi komponentami strojne opreme (*Linux* in *Core C Libraries* na sliki 3.1),
- aplikacijske programske knjižnice (API), ki predstavljajo glavne funkcionalnosti platforme Android in jih lahko razvijalci programske opreme uporabijo pri razvoju. (*Java Libraries* na sliki 3.1),

- izvajalno okolje Android (*runtime*) predstavlja množico knjižnic, ki omogočajo razvoj aplikacij v programskem jeziku Java. Del izvajalnega okolja je tudi navidezni stroj Dalvik (*Dalvik VM* na sliki 3.1)



Slika 3.1: Predogled arhitekture ([20])

3.2 Navidezni stroj

Navidezni stroj Dalvik, del operacijskega sistema Android, je posebna različica navideznega stroja, imenovana procesni navidezni stroj. To pomeni, da teče kot aplikacija znotraj operacijskega sistema in podpira največ en proces. Stroj se ustvari, ko se ustvari nov proces, in zaključi, ko proces preneha teči. Njegov namen je, da ustvari od platforme neodvisno programsko okolje, tako da skrije podrobnosti strojne opreme in operacijskega sistema ter omogoča da se program izvede enako na vsaki platformi.

V operacijskem sistemu Android so vse aplikacije predstavljene kot procesi. Ko odpremo aplikacijo, se naredi nova instanca navideznega stroja

Dalvik in ustvari se nov proces. Dalvik izvaja ukaze iz datoteke *DEX*, ki je ob prevajanju programa ustvarjena iz datotek *CLASS*.

3.3 Razvojna okolja

Platforma Android podpira razvoj programske opreme v programskem jeziku Java in C/C++. Glavni razvoj aplikacije (grafični vmesnik in poslovna logika) poteka v programskem jeziku Java. Tudi večina knjižnic in vmesnikov API je na voljo samo v programskem jeziku Java (slika 3.1). Razvoj programske kode v programskem jeziku C/C++ je namenjen v izjemnih primerih (delovanje programa je v določenih delih prepočasno, potrebna je posebna komunikacija s strojno opremo). Izvajanje programske kode, napisane v C/C++ je hitrejše, saj se izvaja na nižjem nivoju (bližje strojni opremi) (slika 3.1).

Programsko razvojno okolje Android SDK SDK predstavlja celoten *Java Libraries* (slika 3.1). Pri razvoju mobilne aplikacije smo uporabili sledeče gradnike ([27]):

- *Activity* predstavlja glavno komponento vsake mobilne aplikacije. Vsebuje komponente grafičnega uporabniškega vmesnika, omogoča upravljanje grafičnega uporabniškega vmesnika ter kontrolno logiko, ki povezuje grafični uporabniški vmesnik s poslovno logiko. V veliko primerih pa vsebuje tudi poslovno logiko aplikacije. Naprimer: v mobilni aplikaciji je prijava *Activity*, seznam nalog drug *Activity* ter obrazec za oddajo meritev tretji *Activity*

- *View* predstavlja osnovno grafično komponento v grafičnem vmesniku. Celoten grafični uporabniški vmesnik izhaja iz te komponente

- *AsyncTask* omogoča asinhrono izvajanje dela programske kode

- *Adapter* povezuje poslovne podatke z grafičnim uporabniškim vmesnikom. V mobilni aplikaciji je uporabljen za prikaz seznama nalog

- *Service* je komponenta brez grafičnega uporabniškega vmesnika. Uporablja se za izvajanje storitev v ozadju (četudi aplikacija ne teče), kjer interakcija z uporabnikom ni potrebna. V mobilni aplikaciji je uporabljena za sinhronizacijo in oddajanje meritev

Nativno razvojno okolje Android NDK Razvoj programske opreme poteka v programskem jeziku C/C++. Izvorno kodo je treba shraniti v mapo *jni*, ki je v korenski mapi projekta. Programska koda v C/C++ se prevede v strojno kodo in rezultat je deljena (SO) knjižnica. Iz programskega okolja Java knjižnico pred uporabo naložimo s klicem *System.loadLibrary*. Kreiranje knjižnice SO iz izvorne kode poteka s pomočjo konfiguracijske datoteke *Android.mk*.

Datoteka *Android.mk* vsebuje vse potrebne informacije, ki jih potrebujejo orodja, ki bodo iz izvorne kode znotraj direktorija *jni* ustvarile deljene knjižnice.

3.4 Tehnologije Android

Opis tehnologij in knjižnic razvitih posebej za platformo Android, ki smo jih uporabili med razvojem mobilne aplikacije.

Dx *Dx* je orodje za pretvorbo datotek *CLASS* v datoteke *DEX*. Uporabljeno je ob prevajanju programske kode na platformi Android in pretvori vso izvorno kodo, ki je v datotekah *CLASS* v eno datoteko *DEX* . Mogoča je tudi ročna uporaba:

```
dx --dex --output=datoteka.dex  datoteka.class datoteka.class
    ...
```

ProGuard ProGuard je orodje, ki zmanjša, optimizira in ustvari neberljivo izvorno kodo, napisano v programskem jeziku Java. Poišče in izbriše neuporabljene razrede, spremenljivke, metode in atribute. Preostalim razredom, spremenljivkam, metodam in atributom dodeli kratka in brezpomenska

imena. Deluje tako, da gradi graf povezav in nato izbriše elemente, ki niso v grafu. Glavni element izvajanja je konfiguracija, ki jo definiramo v tekstovni obliki. V njej navedemo, katerih referenc ne potrebujemo, katere razrede in pakete želimo ohraniti, kje so knjižnice, ki so del projekta. Navedemo tudi ali želimo kateri postopek (optimizacija, ustvarjanje neberljive izvorne kode) izpustiti. Konfiguracija, ki smo jo ustvarili za namen razvoja mobilne aplikacije je na voljo v prilogi C.

APK APK (*Android Application Package*) je posebna arhivska datoteka, namenjena distribuciji aplikacij na platformi Android. Ob prevajanju in nameščanju aplikacije se naprej vsa izvorna koda prevede v datoteke *CLASS*. Te so nato pretvorjene v eno datoteko *DEX*, ki se shrani v APK.Slednja vsebuje tudi grafične elemente, ki so potrebni za prikaz grafičnega uporabniškega vmesnika. Opcijsko lahko vsebuje tudi dodatne datoteke, ki jih aplikacija potrebuje za delovanje (v našem primeru so bili to arhetipi in predloge).

Vtičnik MIT Kerberos GSS-API Java GSS-API (Generic Security Services Application Program Interface) je standarden vmesnik, ki omogoča dostop do varnostnih storitev posamezne varnostne rešitve. Implementacija protokola GSS-API je v Javi v paketu *org.ietf.jgss*.

MIT Kerberos GSS-API je implementacija vmesnika GSS-API za komunikacijo z varnostnim sistemom Kerberos. Predstavlja vtičnik, prek katerega, lahko s protokolom GSS-API, aplikacije komunicirajo s varnostnim sistemom Kerberos ([14]). Kerberos je protokol za avtentikacijo v računalniških omrežjih, razvit s strani MIT (Massachusetts Instutite of Technology). Kot so zapisali avtorji sami je bil projekt narejen z namenom podpore za Kerberos in GSS-API na platformi Android.

Struktura projekta:

- Vtičnik Java GSS-API oziroma paketi *org.ietf.jgss* ter *edu.mit.jgss* (*LAYER 2* slika 3.2)

- Vtičnik generiran z orodjem SWIG , ki predstavlja komunikacijo C-Java (*LAYER 1* slika 3.2)
- Deljene knjižnice MIT Kerberos/GSS-API (*Native MIT libraries* slika 3.2)

Slika 3.2: Struktura MIT Kerberos GSS-API Java vtičnika, ki ponazarja kako poteka celotna komunikacija ([14])

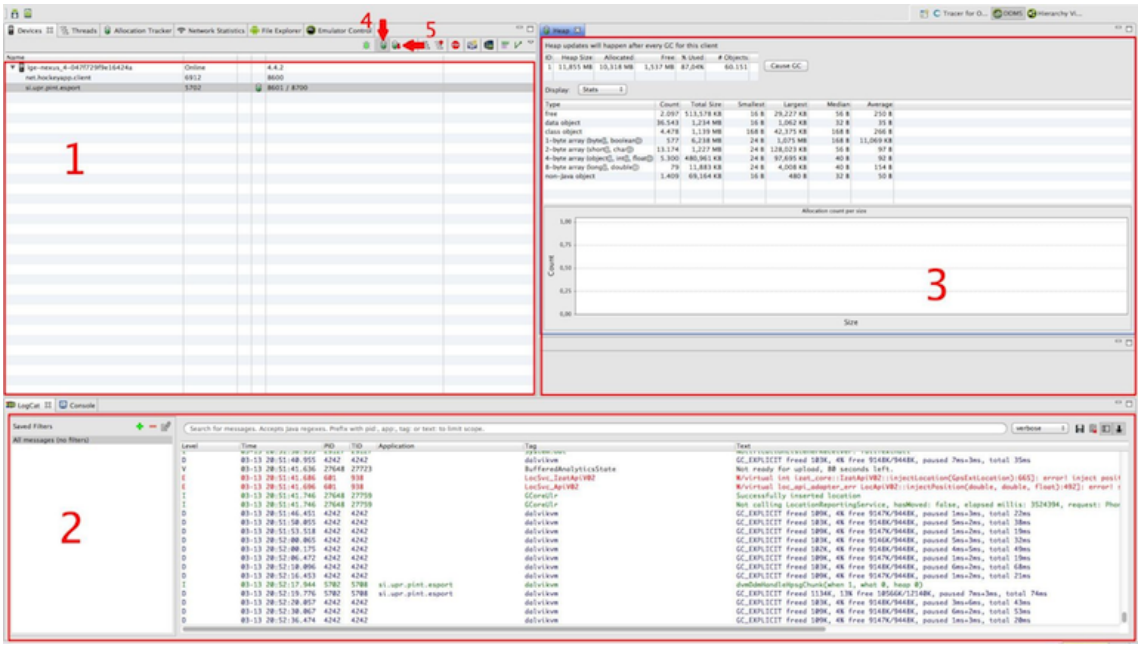
Tape Tape je programska knjižnica, ki vsebuje implementacije različnih tipov podatkovne strukture vrste (*queue*) za platformo Android ([24]). Ena izmed implementacij vrste je tudi stalna podatkovna struktura vrste. V *Tape* je predstavljena kot razred *QueueFile*, vrsta pa je predstavljena kot datoteka na disku. Ob dodajanju elementa v vrsto, se element zapiše v datoteko na disk. V vrsto *QueueFile* lahko dodamo kakršnekoli elemente (edini pogoj je, da so predstavljeni kot podatkovna struktura *byte[]*). V mobilni aplikaciji smo uporabili vrsto *TaskQueue*, ki vsebuje elemente *Task*, za permanentno shranjevanje pa uporablja *QueueFile*. *Task* predstavlja nalogo, ki ima metodo izvedi (*execute*). Če se naloga uspešno izvede (metoda izvedi se je uspešno izvedla), jo odstranimo iz vrste. V kakšni obliki so elementi shranjeni v vrsti na diskum je izbira razvijalca. Za to obstaja razred *Converter*, ki ga je treba implementirati.

Android-logging-log4j Android-logging-log4j je knjižnica, ki omogoča delovanje beleženja log4j na platformi Android .

Android Monitor Android Monitor je orodje namenjeno opravljanju različnih meritev in analizi delovanja aplikacij Android. Izmed številnih funkcionalno-

sti orodja bomo obrazložili le uporabljene pri opravljanju meritev v poglavju 6. Posamezna številka predstavlja številko opisanega dela na sliki 3.3.

1. Pregled nad procesi, ki trenutno tečejo na napravi, pri čemer so prikazani samo tisti, ki dovoljujejo razhroščevanje. Če je razvijalec onemogočil razhroščevanje svoje aplikacije proces tukaj ne bo viden
2. Konzola, kjer se izpisujejo informacije od delovanju aplikacij in operacijskega sistema
3. Pogled, kjer je možno meriti koliko kopice izbran proces porablja
4. Preden lahko začnemo z merjenjem velikosti kopice, je treba omogočiti pridobivanje podatkov o velikosti kopice,
5. Možnost zajema trenutnega stanja kopice.



Slika 3.3: Grafični vmesnik orodja

Poglavje 4

Ostale tehnologije

V tem poglavju predstavimo obstoječo spletno aplikacijo eOskrba in intervencijo eŠport, na kateri temelji mobilna aplikacija. Sledi obrazložitev ostalih tehnologij, ki smo jih uporabili med razvojem.

4.1 Spletna aplikacija eOskrba

eOskrba je spletna aplikacija, ki za izmenjavo in hranjenje elektronskih zdravstvenih kartotek uporablja orodje openEHR. Sestavljena je iz sledečih intervencij: eAstma, eHujšanja, eŠport in eDiabetes. Kartoteke so shranjene v datotekah XML s pomočjo tehnologije eXist. Vse akcije, ki jih izvedemo v spletni aplikaciji, so predstavljene kot procesi. Dodajanje novega pacienta, dodajanje nove naloge, ročni vnos vrednosti, opominjanje pacienta - vse te akcije sprožijo nov proces, ga zaključijo ali pa predstavljajo proces, ki teče ves čas. Procesi so modelirani s tehnologijo Activiti .

4.2 eŠport intervencija

Spletna aplikacija deluje tako, da skrbnik vnese klinične podatke pacienta ter nastavi zahtevnost vadb. Na podlagi vnesenih podatkov se pacientu avtomatsko sestavi ustrezen načrt gibanja. Pacientu je dnevno, tedensko oziroma

glede na načrt gibanja dodeljena naloga. Vsaka naloga predstavlja eno vadbo oziroma rekreacijo, pri čemer je naveden tip, dolžina, trajanje in zahtevnost vadbe. Pacient po opravljeni vadbi prek spletnega obrazca vnese rezultate gibanja. Spletna aplikacija omogoča pacientu tudi ročen vnos vadbe. Na primer: Za določen dan ni bila dodeljena nobena vadba vendar je pacient sam opravil vadbo in želi vnesti rezultate. Podatki o načrtu gibanja, osebni podatki vključitve in rezultati gibanja, ki jih vnaša pacient, so predstavljeni kot arhetipi.

Vključitev novega pacienta Pri postopku vključitve mora skrbnik v spletni obrazec vnesti vsaj sledeče osebne podatke pacienta: ime, priimek, EMŠO, spol, rojstni datum, telesna višina, telesna teža, elektronska pošta, mobilni telefon in poklic. Skrbnik nato izbere program in datum pričetka vadbe. Z oddajo spletnega obrazca se sestavi načrt gibanja za pacienta.

Oddajanje meritev gibanja Po opravljeni vadbi pacient v spletni obrazec vnese rezultate gibanja. Vnesti mora podatke telesne teže: teža, stanje oblačil in drugi dejavniki, ki so vplivali na telesno težo. Vnesti mora tudi podatke o telesni dejavnosti: datum vadbe, športna panoga, trajanje vadbe, razdalja, porabljena energija, intenzivnost vadbe, počutje med vadbo, ustrezna izbira ali je bila vadba opravljena, razlog za neopravljeno vadbo in opombe. Pacient nato odda vnesene meritve, ki predstavljajo klinične podatke in se shranijo v arhetipe.

4.3 Podatkovne baze

eXist Predmetno usmerjena podatkovna baza hrani informacije v obliki programskih predmetov za razliko od relacijske podatkovne baze, kjer so informacije predstavljene v obliki tabel. Predmetno usmerjene podatkovne baze združujejo uporabo podatkovne baze prek paradigme predmetno usmerjenega programiranja.

eXist je odprtokodna rešitev predmetno usmerjene podatkovne baze. Predmeti so predstavljeni kot dokumenti XML. Poizvedbe nad podatki izvajamo s pomočjo tehnologije XQuery ([26]). eXist ima programski vmesnik REST ([8]), kar omogoča brskanje in prenos podatkov prek spleta. Na primeru eOskrbe so predmeti arhetipi.

ORMLite ORMLite je programska knjižnica, ki uporablja tehniko ORM ([22]) za delo s podatkovno bazo SQLite ([23]) v programskem okolju Java.

4.4 Razpoznava podatkov

Vtd-xml Vtd-xml je knjižnica za razpoznavo podatkov iz datotek XML. Razpoznava poteka s pomočjotehnologije XPath ([25]). Knjižnica vtd-xml je ena izmed najhitrejših za razpoznavo datoteke XML. Namesto da razbije podatke na manjše enote in jih nato hrani (kot večina sorodnih tehnologij), si zapomni samo začetni položaj, dolžino vsake enote ter njen odmik od začetnega položaja. Na primer: pri razpoznavni sledečih podatkov

```
<description>
  <lifecycle_state>Initial</lifecycle_state>
</description>
```

,

večina tehnologij za razpoznavanje hrani podatke na sledeč način: *<description >, <lifecycle_state >, Initial, </lifecycle_state >, </description >*,

knjižnica vtd-xml pa si za posamezen podatek, na primer: *<Initial >*, zapomni samo začetni položaj (0), dolžino enote (13) ter odmik od začetnega položaja (31).

XMLBeans XMLBeans je orodje za pretvorbo podatkov XML v Java predmete. Z uporabo orodja XMLBeans ni potrebno ročno razpoznavanje saj so podatki avtomatsko razpoznani in pretvorjeni v predmete Java.

Jackson Jackson je podobno orodje kot XMLBeans vendar namenjeno za delo s podatki JSON . Jackson omogoča avtomatsko razpoznavo podatkov JSON in pretvorbo v predmete Java. Razpoznava uporabi informacije iz razredov Java, katerih predmeti bodo hranili razpoznane podatke.

4.5 Ostalo

Activiti Activiti je odprtokodna rešitev, napisana v programskem jeziku Java, za delo s poslovnimi procesi po standardu BPMN 2.0. Naloga orodja Activiti je, da nadzira procese, ustvarja nove in jih po potrebi ugaša. Tako kot eXist ima programski vmesnik REST ([8]), prek katerega lahko poteka komunikacija.

Jarjar Jarjar je orodje, ki omogoča preimenovanje paketov znotraj datotek JAR.

Swig Swig je programsko orodje, ki omogoča komunikacijo višjenivojskih programskih jezikov (Java, Ruby, Python, ...) s programsko kodo napisano v programskem jeziku C/C++.

Log4j Log4j je knjižnica Java programskem okolju namenjena beleženju dogodkov med izvajanjem programske kode.

Poglavje 5

Mobilna aplikacija

V poglavju je naprej predstavimo funkcionalnosti in videz mobilne aplikacije. Nadaljujemo podroben opis razvoja aplikacije, pri čemer navedemo težave, na katere smo naleteli med razvojem in ustrezne rešitve, ki smo jih uporabili. Posamezni koraki razvoja so razdeljeni v smiselna podpoglavja.

Največji izziv razvoja je predstavljalo orodje openEHR. Cilj naloge je bila implementacija samostojnega odjemalca na mobilni napravi, ki bo med drugim podpiral validacijo kliničnih podatkov. Slednje pomeni uporabo orodja openEHR za potrebe validacije kliničnih podatkov. Ker je openEHR zasnovan za strežniške sisteme in ker na spletu nismo našli informacij o uspešnem delovanju openEHR na mobilnih napravah, sprva nismo vedeli, ali nam bo zadan cilj uspelo doseči. Glavno vprašanje je bilo, ali bo telefon dovolj zmogljiv za opravljanje validacije. Preden pacient odda klinične podatke, validacija preveri, ali ustrezajo omejitvam, definiranim znotraj arhetipov. Brez validacije se podatki takoj pošljejo na strežnik, ki opravi validacijo in vrne odgovor o uspešnosti oddaje. Posledično oddaja brez razpoložljive povezave ni možna, aplikacija pa ni več samostojni odjemalec, saj je močno odvisen od strežnika. Navsezadnje to pomeni, da uporaba openEHR na mobilnih napravah ni mogoča.

5.1 Struktura aplikacije

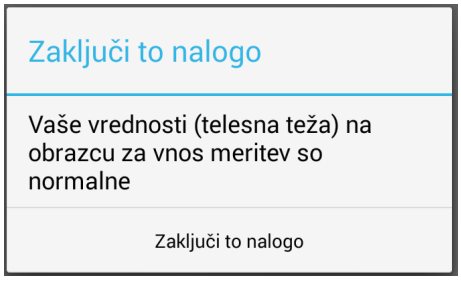
Strukturo aplikacije sestavljajo: prijava, seznama nalog in obrazec za oddajanje nalog oziroma opravljenih vadb. Aplikacija podpira dva različna načina vnosa podatkov opravljene vadbe: vnos rezultatov vadbe, ki mu jo je dodelil sistem, in ročen vnos rezultatov vadbe, ki jo je pacient opravil prostovoljno.

Prijava je sestavljena iz dveh vnosnih polj: uporabniško ime in geslo. Pogled s seznamom nalog je sestavljen iz (slika 5.1):

1. ročnega vnosa rezultatov opravljene vadbe (slika: 5.3),
2. ročne sprožitve sinhronizacije s spletno aplikacijo eOskrba,
3. seznama dodeljenih nalog.

Slika 5.1: Seznam nalog

Na seznamu sta dve vrsti nalog. Prva naloga (od zgoraj navzdol) se ustvari, ko pacient ročno vnese rezultate gibanja. Vnos meritev ni potreben (predhodno jih je vnesel pacient), zato se ob izbiri naloge odpre pojavno okno za zaključek naloge (slika: 5.2). Druga naloga predstavlja vadbo, ki jo je dodelil pacientu sistem. Ob izbiri naloge se odpre obrazec za vnos rezultatov (slika: 5.3). Obrazec ima enaka vnosna polja kot spletni obrazec (poglavje 4.2).



Slika 5.2: Zaključi nalogo

Slika 5.3: Obrazec za vnos rezultatov

5.2 Prikaz seznama nalog

Preko vmesnika Activiti REST pridobimo seznam nalog v notaciji JSON:

```
{
  data: [
    {
    id: "500391",
    name: "Vae vrednosti (telesna tea) na obrazcu za vnos
       meritev so normalne",
    description: "ProceOpomnikVnosNotifierPacient",
    priority: 50,
    assignee: "kk1049@student.uni-lj.si",
    executionId: 500388,
    processInstanceId: "500374",
    formResourceKey: "notify-patient.form"
    }
  ],
  total: 1,
```

```
  start: 0,
  sort: "id",
  order: "asc",
  size: 1
}

,
```

ki jih s pomočjo orodja Jackson pretvorimo v predmet Java. Pred tem je bilo potrebno ustvariti razred Java, ki bo predstavljal seznam nalog. Razred je bil videti tako, pri čemer smo zaradi njegove velikosti vključili le del razreda:

```
@JsonInclude(JsonInclude.Include.NON_NULL)
@JsonPropertyOrder({
        "id",
        "name",
        "description",
        "priority",
        "assignee",
})
public class Task
{

    @JsonProperty("id")
    private String id;

    @JsonProperty("name")
    private String name;

    @JsonProperty("description")
    private String description;

    @JsonProperty("priority")
    private Integer priority;

    @JsonProperty("assignee")
    private String assignee;
}
```

Bralec lahko opazi da parametri razreda vsebujejo anotacije, ki vsebujejo imena spremenljivk v notaciji JSON. To omogoča orodju Jackson da podatke JSON pretvori v predmet Java.

Na tem mestu smo imeli samo podatke o dodeljenih nalogah za pacienta, sedaj pa smo potrebovali podrobne podatke o posamezni vadbi. Podatke smo potrebovali samo za vadbe, ki vsebujejo datum v imenu (slika: 5.1). Podatki o vadbi so v podatkovni bazi eXist v arhetipu (notacija XML), ki vsebuje program vadbe. Celoten arhetip programa vadbe predstavlja velik dokument XML, v našem primeru je vseboval 91 vadb. Primer ene posamezne vadbe je na voljo kot priloga B.

Iz arhetipa programa vadb smo za posamezno vadbo potrebovali: trajanje vadbe, intenzivnost vadbe in tip vadbe. Za razpoznavo podatkov iz arhetipa smo uporabili orodje vtd-xml, ki deluje tako da se s pomočjo jezika XPath ([25]) gibljemo po datoteki XML.

Primer psevdokode razpoznave podatka trajanja vadbe (bralec si za razumevanje vrstnega reda lahko pomaga s prilogo B):

```
selectXPath("//data/events/data/items/items/value[@xsi:type='
    v1:DV_DATE_TIME']/value[.='" + date + "']");
toElement(VTDNav.PARENT);
toElement(VTDNav.PARENT);
toElement(VTDNav.NEXT_SIBLING);
toElement(VTDNav.NEXT_SIBLING);
toElement(VTDNav.FIRST_CHILD);
toElement(VTDNav.NEXT_SIBLING);
toElement(VTDNav.FIRST_CHILD);
getContentFragment();
```

Najprej se pomaknemo na začetno lokacijo v dokumentu, ki jo navedemo prek poti, ki vsebuje datum vadbe (na voljo v imenu vadbe). Zatem se relativno glede na lokacijo gibljemo po datoteki. Ko smo na pravilni lokaciji razpoznamo element. Postopek relativnega gibanja je bilo treba navesti v pravilnem vrstnem redu glede na strukturo dokumenta XML.

Sedaj smo imeli na voljo vse potrebne podatke za prikaz naloge oziroma vadb.

5.3 Validacija in oddajanje meritev

Obrazec za oddajo predstavlja predloga definirana v datotekah *OpomnikEoSp.oet* in *openEHR-EHR-SECTION.opomnik_eo_sp.v1.adl*. Pri oddaji rezultatov so potrebni sledeči arhetipi (slika 5.3). Zraven so dodani opisi kliničnega koncepta, ki ga arhetip predstavlja:

1. *openEHR-EHR-CLUSTER.intenzivnost_opisno_5st_eo.v1.adl* intenzivnost gibanja, predstavljena s petimi stopnjami. Možnosti: zelo nizka intenzivnost, nizka intenzivnost, zmerna intenzivnost, visoka intenzivnost in zelo visoka intenzivnost;

2. *openEHR-EHR-CLUSTER.telesna_dejavnost_neopravljena_razlogi.v1.adl* razlogi za neopravljeno dejavnost. Možnosti: vadba je bila opravljena, lenoba, bolezen, poškodba, odsotnost in drugo;

3. *openEHR-EHR-CLUSTER.opis_pocutja.v1.adl* - počutje po opravljeni vadbi. Možnosti : 1 zelo slabo, 2, 3, 4, 5 odlično;

4. *openEHR-EHR-OBSERVATION.body_weight.v1.adl* telesna teža, komentar, stanje oblačil (lahka oblačila/spodnja oblačila, brez oblačil, povsem oblečen, vključno z obutvijo in plenice) in drugi dejavniki;

5. *openEHR-EHR-OBSERVATION.telesna_dejavnost_enkratna_eo.v1.adl* datum vadbe, športna panoga (aerobika, cooperjev test 2400 m, cooperjev test 12 min, fitnes (delo z utežmi), hoja, igre z loparjem (tenis, badminton, squash), igre z žogo (nogomet, košarka, odbojka, rokomet), kolesarjenje (cestno, gorsko), kolesarjenje (sobno), plavanje, pohodništvo, rolanje, smučanje, sproščanje, tek, tek na smučeh, test hoje 2 km, vaje za gibljivost, vaje za koordinacijo, vaje za moč (doma, ne v fitnesu) in drugo), trajanje vadbe, razdalja, porabljena energija, vadba opravljena in opombe. Arhetip vključuje še 1., 2. in 3. arhetip.

Za uporabo naštetih arhetipov in delovanje validacije smo potrebovali okolje openEHR. Uporabili smo implementacijo Java okolja openEHR . Poslovno logiko validacije smo uporabili kar iz obstoječe spletne aplikacije in

je v sledečih Java razredih: *ArchetypeManager.java*, *Data2RmUtils.java* in *Rm2DataUtils.java*.

OpenEHR in Java Našteti razredi so del openEHR implementacije Java in predstavljajo posamezne funkcionalnosti delovanja orodja openEHR (poglavje 2) oziroma strukture v okolju Java:

- *Archetype* predstavlja programski predmet arhetipa.
- *TEMPLATE* predstavlja programski predmet predloge.
- *ADLParser* razpozna in ustvari programski predmet arhetipa (*Archetype*) iz datoteke *.adl*.
- *OETParser* razpozna definicijo iz datoteke *.oet* in vrne programski predmet *TEMPLATE*.
- *Flattener* s postopkom izravnave zgradi velik arhetip (*Archetype*) iz programskih predmetov *Archetype* in *TEMPLATE* .
- *SkeletonGenerator* predmet v namenska vozlišča drevesa vnese oddane podatke in izbriše odvečna vozlišča.
- *DataValidator* predmet opravi validacijo vnesenih podatkov v drevesu

Psevdokoda validacije Dodajanje in razpoznava arhetipov,

```
ArchetypeManager.addArchetype(openEHR-EHR-CLUSTER.
    intenzivnost_opisno_5st_eo.v1.adl)
ArchetypeManager.addArchetype(openEHR-EHR-CLUSTER.
    opis_pocutja.v1.adl)
ArchetypeManager.addArchetype(openEHR-EHR-OBSERVATION.
    body_weight.v1.adl)
ArchetypeManager.addArchetype(openEHR-EHR-OBSERVATION.
    telesna_dejavnost_enkratna_eo.v1.adl)
ArchetypeManager.addArchetype(openEHR-EHR-SECTION.
    opomnik_eo_sp.v1.adl)
```

```
ArchetypeManager.addArchetype(openEHR-EHR-CLUSTER.
    telesna_dejavnost_neopravljena_razlogi.v1.adl)
```

dodajanje in razpoznava predloge,

```
ArchetypeManager.setTemplate(OpomnikEoSp.oet)
```

sledi postopek izravnave,

```
ArchetypeManager.flattenArchetypeObject();
```

kreiranje novega arhetipa z vnesenimi podatki, pri čemer so tu shranjeni kot ključ-vrednost. Ključ predstavlja pot (primer v podpoglavju 2.5) v drevesu do vozlišča, v katero naj se posamezen podatek (vrednost) zapiše,

```
arhetipSpodatki = ArchetypeManager.populateSkeleton(kljuc-
    vrednost podatki);
```

in na koncu validacija

```
napake = ArchetypeManager.validateData(arhetipSpodatki);
```

Predmet *ArchetypeManager* skrije delo s posameznimi predmeti okolja openEHR in vse funkcionalnosti, potrebne za validacijo, omogoči prek zgoraj naštetih metod. Predmeta *Data2RmUtils* in *Rm2DataUtils* uporablja *ArchetypeManager*, saj vsebujeta poslovno logiko za vstavljanje podatkov v drevo in brisanje odvečnih vozlišč. Cilj je bil, da bi se metoda *ArchetypeManager.validateData* uspešno izvedla in vrnila seznam napak (lahko tudi prazen). Postopek, s katerim smo to dosegli, je opisan skozi sledeče razdelke: 5.3.1, 5.3.3, 5.3.2, 5.3.4 .

5.3.1 Preimenovanje paketov

Prevajanje programske kode, ki je vsebovala knjižnice Java openEHR ni uspelo. OpenEHR je vseboval v Javi pakete, katerih dodajanje s programskimi knjižnicami na platformi Android ni dovoljeno. To so paketi *java.* in javax.**. V teh paketih so ključni razredi platforme Android (*JavaSE/Java Apache* slika 3.1). Dodajanje teh paketov skozi knjižnice lahko povztroči konflikte. Ključni razredi se lahko z razvojem platforme spreminjajo, in če

dodajamo te pakete skozi knjižnice, potem aplikacija uporablja stare verzije, kar lahko povzroči nedelovanje.

S pomočjo orodja jarjar je bilo treba vsem paketom dodati predpono *si.upr.pint.esport*. Treba je bilo preimenovati tudi knjižnice, ki so uporabljale te pakete. Problem je bil v paketu *javax.xml.stream*, ki je bil znotraj knjižnice XMLBeans. Knjižnico uporablja orodje openEHR za delo z datotekami XML.

Za preimenovanje je treba kreirati datoteko s pravili za preimenovanje. Pravilo:

```
rule javax.xml.stream.** si.upr.pint.esport.javax.xml.stream.
    @1
```

Sledi ukaz:

```
java -jar jarjar.jar process <datotekaSpravili> <knjiznicaJar
    > <preimenovanaKnjiznicaJar>
```

5.3.2 Beleženje dogodkov

OpenEHR za beleženje vseh dogodkov uporablja knjižnico log4j. Knjižnica na platformi Android ni delovala zaradi manjkajočega paketa java.beans.*. Rešitev je bila uporaba orodja *android-logging-log4j*. Vsi dogodki so se sedaj med delovanjem openEHR uspešno beležili.

5.3.3 Zmanjšanje velikosti aplikacije

Programsko kodo s preimenovanimi paketi smo poskušali znova prevesti. Prišlo je do nove težave. Posamezna datoteka *DEX* lahko vsebuje največ 65536 metod. Z integracijo okolja openEHR smo število dovoljenih metod presegli. Ostajata dva pristopa reševanja tega problema.

Metoda uporabe več datotek *DEX* Celotna aplikacija je predstavljena kot datoteka *DEX* v datoteki APK. Ob izvajanju aplikacije se v resnici izvaja datoteka *DEX*. Če celotne programske kode ni možno shraniti v eno datoteko

DEX, lahko preostalo programsko logiko zapišemo v drugo datoteko *DEX*. Če je potrebno, najprej pretvorimo izvorno kodo v datoteke *CLASS*, nato ročno z orodjem *dx* ustvarimo datoteko *DEX* in jo dodamo, podobno kot arhetipe in predloge, kot prilogo aplikacije. Na ta način bo druga datoteka *DEX* shranjena v APK. Ob izvajanju aplikacije nato dinamično naložimo programsko kodo iz druge datoteke *DEX* in jo začnemo uporabljati.

Metoda z uporabo orodja ProGuard (poglavje 3.4) Treba je spisati ustrezno konfiguracijsko datoteko, ki je uporabljena ob izvajanju orodja ProGuard. Orodje se sproži ob prevajanju izvorne kode v datoteke CLASS, preden so shranjene v datoteko *DEX*.

Najprej smo uporabili prvo metodo, tako da smo celotno okolje openEHR shranili v drugo datoteko *DEX*. Z uporabljeno metodo postopek izravnave ni bil uspešen. Poleg tega je bilo treba uporabiti posebno tehniko programiranja. Vsak predmet Java, ki ga potrebujemo je treba dinamično naložiti, pri čemer je treba poznati njegovo polno ime, vključno s paketom, v katerem je. Tudi metode predmeta, ki jih želimo uporabljati, moramo najprej dinamično najti in jih nato poklicati. Posledično je programiranje zelo oteženo in prinese veliko dodatne programske kode, ki je drugače nebi potrebovali.

Ker nam vzroka problema pri prvi metodi ni uspelo najti, smo se odločili vzporedno uporabiti drugo metodo. Ob upoštevanju dejstva da znotraj orodja openEHR potrebujemo samo funkcionalnost validacije, bi lahko ostale neuporabljene dele orodja odstranili.

Cilj druge metode je bil zmanjšanje programske kode oziroma zmanjšanje števila metod v projektu. Konfiguracija, ki smo jo ob tem uporabili, je na voljo kot priloga C. Metoda je delovala, število metod se je zmanjšalo. Zatem smo znova poskušali oddati rezultate vadbe skozi obrazec v mobilni aplikaciji. Tokrat je postopek izravnave deloval. Postopek je deloval do klica metode *validateData* (psevdokoda 5.3), potem pa je prišlo do napake.

5.3.4 Manjkajoča implementacija vtičnika GSS-API

Do napake je prišlo ker je del programske kode znotraj okolja openEHR potreboval predmet *Oid*, ki pa je v paketu *org.ietf.jgss*. Z nekaj raziskovanja smo ugotovili da tega paketa vključno z razredi v tem paketu, v delu *Java Libraries* (slika 3.1) na platformi Android ni. Paket *org.ietf.jgss* predstavlja implementacijo vtičnika GSS-API (podpoglavje 3.4) v programskem okolju Java.

Ob iskanju rešitve smo naleteli na projekt univerze MIT, imenovan MIT Kerberos GSS-API Java. Projekt je vseboval paket oziroma implementacijo vtičnika GSS-API v programskem okolju Java in C . Vtičnik za delovanje potrebuje implementacijo v programskem okolju Java in C oziroma treba je bilo uporabiti Android SDK in Android NDK (podpoglavje 3.3).

Android SDK V programsko kodo mobilne aplikacije je bilo treba dodati paket *org.ietf.jgfss*. MIT Kerberos GSS-API Java je implementacija namenjena varnostnemu sistemu Kerberos (podpoglavje 3.4) zato je bilo treba za delovanje dodati še *org.ietf.jgfss*.

Android NDK V mapo *jni* smo dodali izvorno kodo vtičnika v programskem jeziku C (*gsswrapper.i* in *gsswrapper_wrap.h*). Nato smo z ukazom

```
swig -java -package edu.mit.jgss.swig -outdir ./src/edu/mit/
    jgss/swig -o ./jni/gsswrapper\_wrap.c ./jni/gsswrapper.i
```

s pomočjo orodja Swig ustvarili vmesnik za komunikacijo programske kode Java s programsko kodo C. Sledila je konfiguracija datoteke Android.mk (na voljo v prilogi D). Sedaj smo imeli vse pripravljeno za generiranje knjižnice *gsswrapper.so*, ki je predstavljala implementacijo vtičnika v programskem jeziku C. Knjižnico smo generirali z ukazom *ndk-build*.

Zatem smo znova poskusili z validacijo. Tokrat so se vse metode (psevdokoda 5.3) vključno z *validateData* uspešno izvedle. Validacija je delovala. Glavni cilj ni bil protokol GSS-API ali varnostni sistem Kerberos zato smo implementacijo preučili le do mere, ki je bila potrebna za delovanje validacije.

5.4 Shranjevanje podatkov

Celotna aplikacija je zasnovana tako da deluje tudi brez povezave. Pri kliničnih podatkih je ključnega pomena da se podatki, če je le mogoče ne izgubljajo. Aplikacija shranjuje podatke o nalogah, ki jih pridobi s spleta, v podatkovno bazo, oddane rezultate pa v datoteko, ki predstavlja permanentno vrsto. Tako lahko pacient pregleda seznam nalog in odda rezultate gibanja tudi ko ni razpoložljive povezave. Ob nerazpoložljivi povezavi se oddani rezultati shranijo na disk in pošljejo, ko je mogoče.

Znotraj aplikacije je implementirana storitev, ki ob vsaki uporabi mobilne aplikacije, naredi sinhronizacijo podatkovne baze, če so na voljo kakšne naloge oziroma zbriše naloge, če so bile zaključene na spletu. Obenem pa pošlje vse vnesene podatke na strežnik. Za shranjevanje v podatkovno bazo je uporabljena tehnologija ORMLite.

Permanentno shranjevanje oddanih rezultatov je implementirano z orodjem Tape (poglavje 3.4). Posamezna oddaja je predstavljena kot *Task*. Ko pacient vnese podatke in je bila validacija uspešna se ustvari predmet *Task*, ki vsebuje oddane rezultate. Predmet se nato doda v permanentno vrsto. Ob dodajanju v vrsto se sproži storitev, ki poskuša rezultate prenesti v spletno aplikacijo. Če oddaja ni uspešna, se storitev prekine, ob naslednji uporabi aplikacije ali oddaji rezultatov, pa ponovno poskuša oddati rezultate.

Poglavje 6

Rezultati

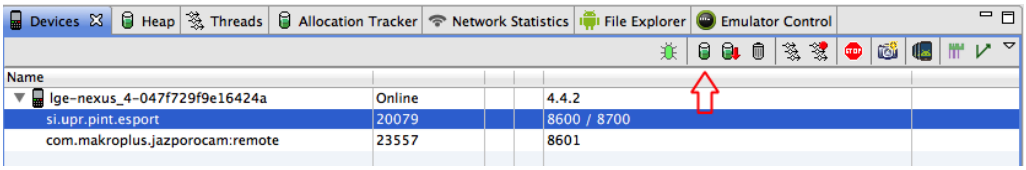
Glavni cilj diplomske naloge je bil dosežen. OpenEHR validacija podatkov v arhetipu je delovala. Ker je aplikacija delovala smo lahko sklepali, da so mobilni telefoni dovolj zmogljivi za delovanje orodja openEHR na platformi Android. Vendar točnih podatkov o porabi pomnilnika nismo imeli. Platforma Android ima omejitev koliko pomnilnika lahko aplikacija naenkrat porablja oziroma največjo velikost kopice. Omejitev je odvisna od zmogljivosti mobilne naprave vendar večina naprav dovoljuje kopice do velikosti 64MB. Predvsem nas je zanimalo kakšna je poraba pomnilnika oziroma velikost kopice glede na največjo dovoljeno velikost kopice.

6.1 Meritev velikosti kopice

S pomočjo orodja Android Monitor smo opravili meritve velikosti kopice. Izmerili smo jo pred postopkom validacije in po njem. Za lažjo primerjavo smo izmerili še velikost kopice standardne aplikacije Android ([18]). Za standardno aplikacijo smo jo označili xato ker predstavlja funkcionalnosti, ki so podobne v večini aplikacij (prikaz podatkov preko REST vmesnika, zajem slike in uporaba geolokacijskih storitev).

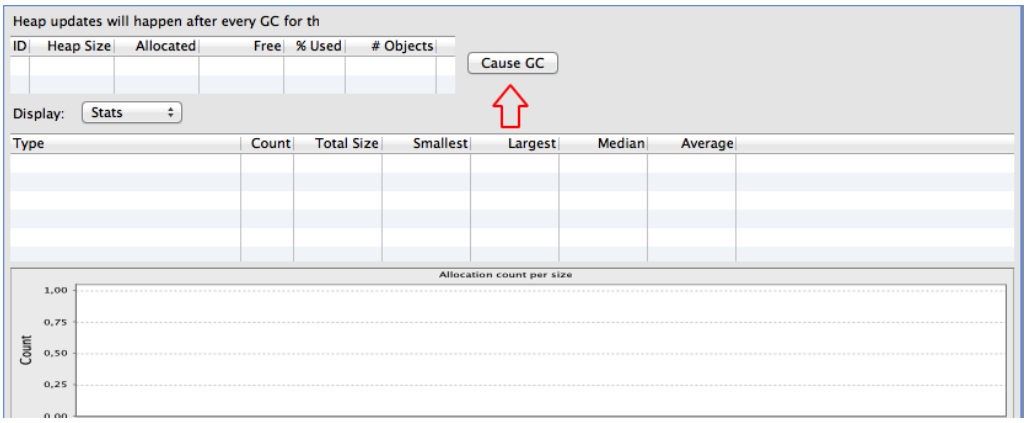
6.1.1 Postopek

V grafičnem vmesniku je naprej treba izbrati proces, ki ga želimo analizirati. S klikom na Update Heap (slika 6.1) omogočimo zajemanje meritev.



Slika 6.1: Pričetek zajema velikosti kopice

Nato izberemo zavihek Heap, kjer s klikom na Cause GC (slika 6.2) zajamemo meritev.



Slika 6.2: Zajema velikosti kopice

Obrazložitev podatkov o kopici:

- *Heap Size* velikost celotne kopice
- *Allocated* velikost kopice v uporabi
- *Free* velikost proste kopice
- *% Used* delež uporabljene kopice
- *# Objects* število predmetov v kopici

Parametra *Heap Size* in *Allocate* bi lahko razložili še natančneje. Android zaradi optimizacije ne alocira celotne dovoljene velikosti kopice. Alokacije potekajo po potrebi. Na začetku alocira določeno velikost in če je kopica premajhna jo ustrezno poveča. Ko to ni več potrebno kopico zmanjša. Podatek *Heap Size* predstavlja celotno velikost kopice v danem trenutku. Parameter *Allocated* pa prikazuje velikost kopice v uporabi. Če *Allocated* preseže *Heap Size*, se velikost kopice oziroma *Heap Size* poveča.

6.1.2 Rezultati

ID	Heap Size	Allocated	Free	% Used	# Objects
1	11,875 MB	10,326 MB	1,549 MB	86,95%	60.206

Slika 6.3: Velikost kopice pred postopkom validacije

ID	Heap Size	Allocated	Free	% Used	# Objects
1	16,543 MB	14,489 MB	2,054 MB	87,58%	124.567

Slika 6.4: Velikost kopice po validaciji

ID	Heap Size	Allocated	Free	% Used	# Objects
1	19,293 MB	13,268 MB	6,025 MB	68,77%	75.139

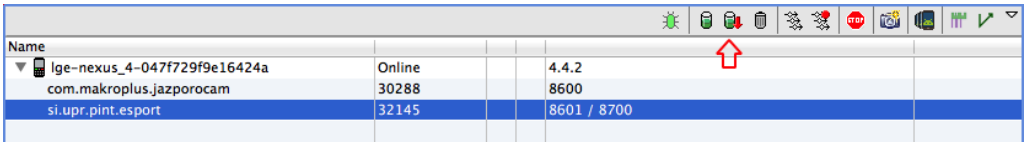
Slika 6.5: Velikost kopice standardne aplikacije

6.2 Zajem stanja kopice

S pomočjo orodij Android Monitor in Eclipse Memory Analyzer smo zajeli in grafično prikazali podrobnejše stanje kopice. Sledeč postopek je največkrat namenjen iskanju napak uhajanja pomnilnika.

6.2.1 Postopek zajema

Najprej je bilo treba v orodju Android Monitor izbrati proces in nato s klikom na Heap Dump (slika 6.6) sprožiti zajem. Rezultat je datoteka s stanjem kopice, zapisana v *hprof*. Sliki 6.7 in 6.8, prikazujeta stanje kopice, pri čemer so našteti predmeti glede na velikost (v MB), ki jo zasedajo v kopici.



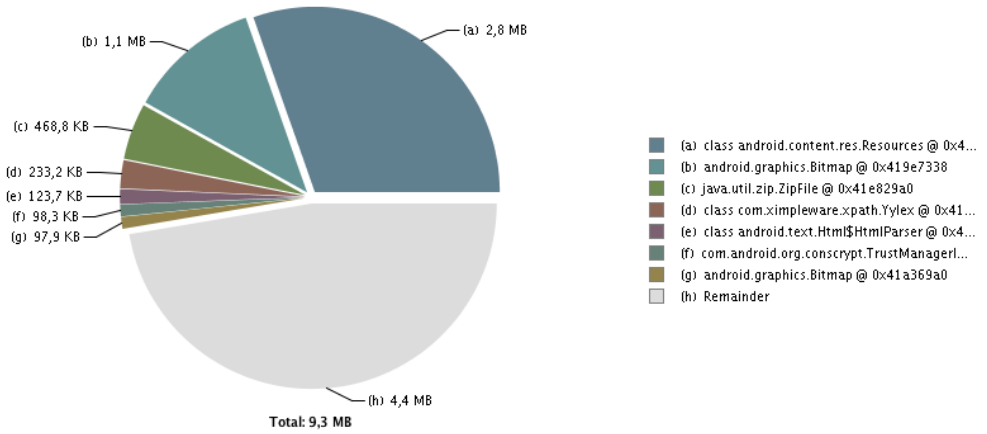
Slika 6.6: Izpis stanja kopice

Da lahko podatke prikažemo v orodju Eclipse Memory Analyzer, je bilo treba datoteko pretvoriti v standarden format *hprof*. Namreč prvoten hprof, generiran iz okolja Dalvik ima malo drugačen format. Da lahko datoteko prikažemo v orodju Eclipse Memory Analyzer, moramo hprof format spremeniti v standarden format Java. To naredimo s sledečim ukazom:

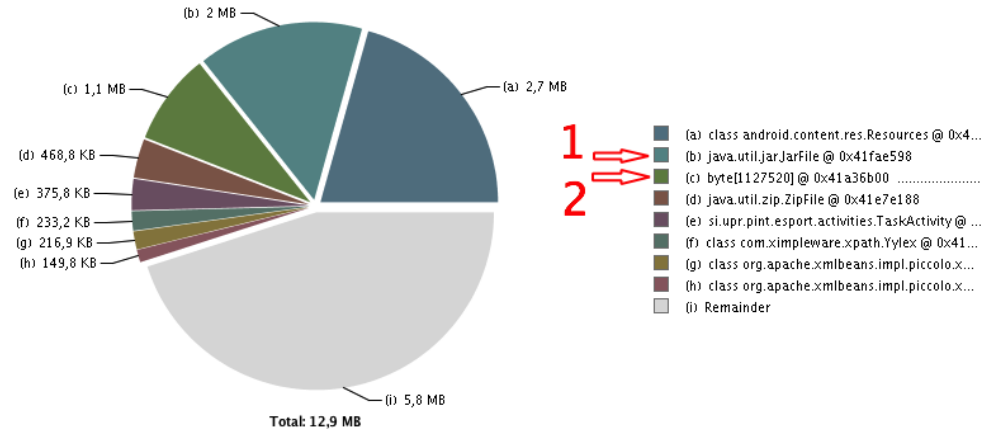
```
hprof-conf <vhodna_datoteka.hprof> <izhodna_datoteka.hprof>
```

Ustvarjeno datoteko nato odpremo v orodju Eclipse Memory Analyzer.

6.2.2 Rezultati



Slika 6.7: Stanje kopice pred validacijo



Slika 6.8: Stanje kopice po validaciji

6.3 Obrazložitev in komentarji rezultatov

Velikost kopice Kopica se je pri izvajanju validacije povečala iz 10MB (slika 6.3) na 14MB (slika 6.4) oziroma iz 9MB (slika 6.7) na 13MB (slika 6.8). Celotna velikost kopice je sprva znašala 11MB (slika 6.3) in je nato zrasla

na 16MB (slika 6.4). Iz rezultatov lahko sklepamo, da velikost kopice med delovanjem ne dosega niti ne presega mejnih vrednosti dovoljene velikosti. V primerjavi s standardno aplikacijo, velikosti 13MB (slika 6.5), bi lahko sklepali, da je njena poraba relativno nizka.

Stanje predmetov v kopici Stanje kopice po validaciji (slika 6.7) vsebuje dva predmeta, ki ju stanje pred validacijo (slika 6.8) ne vsebuje. To sta predmeta *JarFile* ((slika 6.7) število 1) in *byte[]* ((slika 6.7) število 2). Implementacija openEHR je v programskem okolju Java predstavljena kot JAR datoteke. Pri validaciji se ustvarijo predmeti openEHR (podpoglavje 5.3), ki so v datotekah JAR. Sklepamo lahko, da *JarFile* predstavlja programsko kodo implementacije openEHR. Struktura *byte[]* je velika 1MB in najverjetneje predstavlja celoten arhetip v obliki drevesa.

Velikost arhetipa v primerjavi s celotno kopico Velikost kopice po validaciji je 13MB, pri čemer je ocena velikosti dela openEHR 2MB in velikost celotnega arhetipa v pomnilniku 1,1MB (slika 6.8).

Poglavje 7

Zaključek

Kljub zapletenemu postopku nam je uspelo razviti prototip, ki uspešno uporablja orodje openEHR za validacijo kliničnih podatkov. Izraz prototip smo uporabili namenoma, uporabili smo namreč preproste predloge in arhetipe. Namen je bil zgolj preveriti, ali je delovanje orodja openEHR sploh mogoče. Obstaja veliko možnosti za izboljšave predvsem v implementaciji orodja openEHR. Trenutna implementacija ni razvita z zavednostjo mobilnih naprav. Uporablja tudi veliko knjižnic, ki so za mobilne naprave preobsežne in neprimerne. Takšni primeri so: log4j, Joda-Time, XMLBeans.

7.1 Primeri izboljšav

MLHIM Na mobilni napravi bi lahko namesto orodja openEHR uporabili MLHIM.

Log4j Knjižnico log4j za beleženje bi lahko nadomestili z orodjem LogCat, ki je del platforme Android za beleženje dogodkov.

Joda-Time Joda-Time je obsežen aplikacijski programski vmesnik za delo s časi in datumi v programskem okolju Java. Joda-Time bi lahko nadomestili z razredi za delo s časi in datumi, ki so del platforme Android.

XMLBeans OpenEHR uporablja orodje XMLBeans za delo z dokumenti XML. Menimo da je to največja ovira za zanesljivo delovanje openEHR na mobilnih napravah. V našem diplomskem delu smo uporabljali orodje openEHR le za potrebe validacije in z dokumenti XML nismo imeli opravka. Vseeno pa smo ob podrobnejšem raziskovanju ugotovili da lahko XMLBeans povzroči visoko porabo kopice in je znan kot prezahtevno orodje za mobilne naprave. Sorodne tehnologije, ki bi zanesljivo delovala na platformi Android ni. Najboljša rešitev bi bila uporaba notacije JSON namesto XML, slednja ima tudi veliko boljšo podporo orodij za mobilne naprave.

Lokalno shranjevanje arhetipov Ob upoštevanju zgornjih izboljšav bi lahko izboljšali tudi mobilno aplikacijo, tako da bi hranila arhetipe na mobilni napravi in opravljala sinhronizacijo arhetipov. Trenutno so shranjeni samo oddani rezultati. Arhetip se sicer ustvari vendar samo v pomnilniku za potrebe validacije. Naslednji korak bi bil da bi arhetip pretvorili v JSON, ga shranili na mobilno napravo ter nato poslali na strežnik.

Dodatek A

Definicija arhetipa

```
archetype (adl_version=1.4)
  openEHR-EHR-CLUSTER.intenzivnost_opisno_5st_eo.v1

concept
  [at0000]  -- Intenzivnost opisno 5st eo
language
  original_language = <[ISO_639-1::sl]>
description
  original_author = <
    ["name"] = <"">
  >
  details = <
    ["sl"] = <
      language = <[ISO_639-1::sl]>
      purpose = <"Cluster za opis intenzivnosti pri portni/
         gibalni/telesni dejavnosti.">
      use = <"">
      misuse = <"">
      copyright = <"">
    >
  >
  lifecycle_state = <"0">
  other_contributors = <>
  other_details = <
```

```
    ["MD5-CAM-1.0.1"] = <"C7747FEC7B2B7B7AFF9046177BBEF58B">
  >

definition
  CLUSTER[at0000] matches { -- Intenzivnost opisno 5st eo
    items cardinality matches {1..*; unordered} matches {
      ELEMENT[at0001] occurrences matches {0..1} matches {
         -- Intenzivnost vadbe (5 stopenj)
        value matches {
          1|[local::at0002],  -- Zelo nizka intenzivnost
          2|[local::at0003],  -- Nizka intenzivnost
          3|[local::at0004],  -- Zmerna intenzivnost
          4|[local::at0005],  -- Visoka intenzivnost
          5|[local::at0006]   -- Zelo visoka intenzivnost
        }
      }
    }
  }

ontology
  term_definitions = <
    ["sl"] = <
      items = <
        ["at0000"] = <
          text = <"Intenzivnost opisno 5st eo">
          description = <"unknown">
        >
        ["at0001"] = <
          text = <"Intenzivnost vadbe (5 stopenj)">
          description = <"*">
        >
        ["at0002"] = <
          text = <"Zelo nizka intenzivnost">
          description = <"Regeneracija (do 60 % FSmax). Med
              aktivnostjo je mogo sproen pogovor.">
        >
        ["at0003"] = <
          text = <"Nizka intenzivnost">
```

```
        description = <"Aerobno I (60 do 70 % FSmax). Med
            aktivnostjo je mogo pogovor s krajimi
            prekinitvami.">
      >
      ["at0004"] = <
        text = <"Zmerna intenzivnost">
        description = <"Aerobno II (70 do 80 % FSmax). Med
            vadbo je pogovor oteen, pogovarja se lahko le v
            zelo kratkih stavkih.">
      >
      ["at0005"] = <
        text = <"Visoka intenzivnost">
        description = <"Aerobno in anaerobno (80 do 90 %
            FSmax). Med vadbo je pogovor oteen, pogovarja se
             lahko le v zelo kratkih stavkih.">
      >
      ["at0006"] = <
        text = <"Zelo visoka intenzivnost">
        description = <"Preteno anaerobno (90 do 100 %
            FSmax). Med vadbo pogovor praktino ni mogo.">
      >
    >
  >
>
```

Dodatek B

Arhetip

```
<items xmlns="http://schemas.openehr.org/v1" xmlns:xsi="http:
   //www.w3.org/2001/XMLSchema-instance" archetype_node_id="
   openEHR-EHR-OBSERVATION.sport_koledar.v1" xsi:type="
   OBSERVATION">
    <name>
        <value>Sport calendar</value>
    </name>
    <archetype_details>
        <archetype_id>
            <value>openEHR-EHR-OBSERVATION.sport_koledar.v1</
               value>
        </archetype_id>
        <rm_version>1.0.1</rm_version>
    </archetype_details>
    <language>
        <terminology_id>
            <value>ISO_639-1</value>
        </terminology_id>
        <code_string>en</code_string>
    </language>
    <encoding>
        <terminology_id>
            <value>IANA_character-sets</value>
        </terminology_id>
```

```
        <code_string>UTF-8</code_string>
    </encoding>
    <subject xmlns:v1="http://schemas.openehr.org/v1"
       xsi:type="v1:PARTY_SELF">
        <external_ref>
            <id xsi:type="v1:HIER_OBJECT_ID">
                <value>1.2.4.5.6.12.1</value>
            </id>
            <type>PARTY</type>
        </external_ref>
    </subject>
    <data archetype_node_id="at0001">
        <name>
            <value>Event Series</value>
        </name>
        <origin>
            <value>2013-08-21T21:30:24,095</value>
        </origin>
        <events xmlns:v1="http://schemas.openehr.org/v1"
           archetype_node_id="at0002" xsi:type="
           v1:POINT_EVENT">
            <name>
                <value>Any event</value>
            </name>
            <time>
                <value>2013-08-21T21:30:24,095</value>
            </time>
            <data archetype_node_id="at0003" xsi:type="
               v1:ITEM_TREE">
                <name>
                    <value>Tree</value>
                </name><items archetype_node_id="openEHR-EHR-
                   CLUSTER.sport_koledar.v1" xsi:type="
                   v1:CLUSTER">
                    <name>
                        <value>unknown</value>
                    </name>
```

```
<items archetype_node_id="at0001"
   xsi:type="v1:ELEMENT">
    <name>
        <value>Program</value>
    </name>
    <value xsi:type="v1:DV_CODED_TEXT">
        <value>Visoka intenzivnost</value
           >
        <defining_code>
            <terminology_id>
                <value>local</value>
            </terminology_id>
            <code_string>at0008</
               code_string>
        </defining_code>
    </value>
</items>
<items archetype_node_id="at0002"
   xsi:type="v1:ELEMENT">
    <name>
        <value>Datum vadbe</value>
    </name>
    <value xsi:type="v1:DV_DATE_TIME">
        <value>2014-02-06T00</value>
    </value>
</items>
<items archetype_node_id="at0003"
   xsi:type="v1:ELEMENT">
    <name>
        <value>Podtip vadbe</value>
    </name>
    <value xsi:type="v1:DV_CODED_TEXT">
        <value>Klanec</value>
        <defining_code>
            <terminology_id>
                <value>local</value>
            </terminology_id>
```

```
                <code_string>at0011</
                    code_string>
            </defining_code>
        </value>
    </items>
    <items archetype_node_id="at0004"
       xsi:type="v1:ELEMENT">
        <name>
            <value>Trajanje vadbe</value>
        </name>
        <value xsi:type="v1:DV_TIME">
            <value>01:00:00</value>
        </value>
    </items>
    <items archetype_node_id="at0005"
       xsi:type="v1:ELEMENT">
        <name>
            <value>Disciplina</value>
        </name>
        <value xsi:type="v1:DV_CODED_TEXT">
            <value>Vzdrljivostna vadba</value
               >
            <defining_code>
                <terminology_id>
                    <value>local</value>
                </terminology_id>
                <code_string>at0013</
                    code_string>
            </defining_code>
        </value>
    </items>
    <items archetype_node_id="at0006"
       xsi:type="v1:ELEMENT">
        <name>
            <value>Intenzivnost vadbe</value>
        </name>
        <value xsi:type="v1:DV_CODED_TEXT">
```

```
                        <value>Zelo nizka intenzivnost</
                            value>
                        <defining_code>
                            <terminology_id>
                                <value>local</value>
                            </terminology_id>
                            <code_string>at0015</
                                code_string>
                        </defining_code>
                    </value>
                </items>
            </items>
        </data>
    </events>
</data>
</items>
```

Dodatek C

Konfiguracijska datoteka ProGuard

```
# This is a configuration file for ProGuard.
# http://proguard.sourceforge.net/index.html#manual/usage.
   html

-dontusemixedcaseclassnames
-dontskipnonpubliclibraryclasses
-verbose

# Optimization is turned off by default. Dex does not like
   code run
# through the ProGuard optimize and preverify steps (and
   performs some
# of these optimizations on its own).
-dontoptimize
-dontpreverify
-dontobfuscate
# Note that if you want to enable optimization, you cannot
   just
# include optimization flags in your own project
   configuration file;
# instead you will need to point to the
```

```
# "proguard-android-optimize.txt" file instead of this one
   from your
# project.properties file.

-keepattributes *Annotation*
-keep public class com.google.vending.licensing.
   ILicensingService
-keep public class com.android.vending.licensing.
   ILicensingService

# For native methods, see http://proguard.sourceforge.net/
   manual/examples.html#native
-keepclasseswithmembernames class * {
    native <methods>;
}

# keep setters in Views so that animations can still work.
# see http://proguard.sourceforge.net/manual/examples.html#
   beans
-keepclassmembers public class * extends android.view.View {
   void set*(***);
   *** get*();
}

# We want to keep methods in Activity that could be used in
   the XML attribute onClick
-keepclassmembers class * extends android.app.Activity {
   public void *(android.view.View);
}

# For enumeration classes, see http://proguard.sourceforge.
   net/manual/examples.html#enumerations
-keepclassmembers enum * {
    public static **[] values();
    public static ** valueOf(java.lang.String);
}

-keep class * implements android.os.Parcelable {
```

```
  public static final android.os.Parcelable$Creator *;
}

-keepclassmembers class **.R$* {
    public static <fields>;
}


# The support library contains references to newer platform
   versions.
# Don't warn about those in case this app is linking against
   an older
# platform version.  We know about them, and they are safe.
-dontwarn android.support.**


-dontwarn retrofit.**
-dontwarn org.apache.log4j.**
-dontwarn org.jaxen.**
-dontwarn org.joda.time.**
-dontwarn org.apache.xmlbeans.impl.jam.**
-dontwarn org.jdom.JDOMException
-dontwarn com.fasterxml.jackson.databind.ext.**
-dontwarn org.w3c.dom.**
-dontwarn org.openehr.rm.support.identification.**
-dontwarn org.apache.tools.ant.taskdefs.Rmic
-dontwarn org.apache.xmlbeans.impl.tool.**

#-dontwarn org.jdom.**
#-dontwarn com.fasterxml.jackson.databind.**
#-dontwarn si.upr.pint.esport.javax.xml.stream.**
#-dontwarn org.w3c.dom.**
#-dontwarn openEHR.v1.template.impl.**
#-dontwarn org.openehr.schemas.v1.impl.**
#-dontwarn com.oceaninformatics.characterMapping.impl.**
#-dontwarn org.openehr.rm.support.**

#Libs
```

```
-keep class retrofit.** { *; }

#Jackson
-dontskipnonpubliclibraryclassmembers
-keepattributes *Annotation*,EnclosingMethod
-keepnames class org.codehaus.jackson.** { *; }

#OpenEHR
-keep class javax.measure.** { *; }
-keep class org.apache.** { *; }
-keep class org.jdom.** { *; }
-keep class org.joda.time.** { *; }
-keep class org.openehr.** { *; }
-keep class org.openehr.** { *; }
-keep class se.acode.openehr.** { *; }

-keep class si.upr.pint.esport.javax.xml.stream.** { *; }

-keep class schemaorg_apache_xmlbeans.** { *; }
-keep class openEHR.v1.template.** { *; }
-keep class org.openehr.** { *; }
-keep class com.oceaninformatics.characterMapping.** { *; }

-keep class org.apache.xmlbeans.** { *; }
-keep class org.w3c.dom.** { *; }

-keep class repackage.** { *; }


#Otto
-keepclassmembers class ** {
    @com.squareup.otto.Subscribe public *;
    @com.squareup.otto.Produce public *;
}

#Butter Knife
-dontwarn butterknife.internal.**
-keep class **$$ViewInjector { *; }
```

```
-keepnames class * { @butterknife.InjectView *;}
-keepnames class * { @butterknife.OnItemClick *;}
-keepnames class * { @butterknife.OnClick *;}


# ORMLITE
-keep class com.j256.**
-keepclassmembers class com.j256.** { *; }
-keep enum com.j256.**
-keepclassmembers enum com.j256.** { *; }
-keep interface com.j256.**
-keepclassmembers interface com.j256.** { *; }

-keep class si.upr.pint.esport.database.DatabaseHelper { *; }
-keep class si.upr.pint.esport.database.Task  { *; }

#Beans
-keep public class si.upr.pint.esport.beans.**
-keepclassmembers public class si.upr.pint.esport.beans.** {
   *; }

-assumenosideeffects class org.apache.log4j.Logger { *; }
```

Dodatek D

Datoteka Android.mk

```
LOCAL_PATH := $(call my-dir)

## libkadm5srv_mit
include $(CLEAR_VARS)
LOCAL_MODULE := libkadm5srv_mit
LOCAL_SRC_FILES := lib/libkadm5srv_mit.a
include $(PREBUILT_STATIC_LIBRARY)

## libkdb5
include $(CLEAR_VARS)
LOCAL_MODULE := libkdb5
LOCAL_SRC_FILES := lib/libkdb5.a
include $(PREBUILT_STATIC_LIBRARY)

## libkrb5_db2
include $(CLEAR_VARS)
LOCAL_MODULE := libkrb5_db2
LOCAL_SRC_FILES := lib/libkrb5_db2.a
include $(PREBUILT_STATIC_LIBRARY)

## libgssrpc
include $(CLEAR_VARS)
LOCAL_MODULE := libgssrpc
LOCAL_SRC_FILES := lib/libgssrpc.a
```

```
include $(PREBUILT_STATIC_LIBRARY)

## libgssapi_krb5
include $(CLEAR_VARS)
LOCAL_MODULE := libgssapi_krb5
LOCAL_SRC_FILES := lib/libgssapi_krb5.a
include $(PREBUILT_STATIC_LIBRARY)

## libkrb5
include $(CLEAR_VARS)
LOCAL_MODULE := libkrb5
LOCAL_SRC_FILES := lib/libkrb5.a
include $(PREBUILT_STATIC_LIBRARY)

## libk5crypto
include $(CLEAR_VARS)
LOCAL_MODULE := libk5crypto
LOCAL_SRC_FILES := lib/libk5crypto.a
include $(PREBUILT_STATIC_LIBRARY)

## libcom_err
include $(CLEAR_VARS)
LOCAL_MODULE := libcom_err
LOCAL_SRC_FILES := lib/libcom_err.a
include $(PREBUILT_STATIC_LIBRARY)

## libkrb5support
include $(CLEAR_VARS)
LOCAL_MODULE := libkrb5support
LOCAL_SRC_FILES := lib/libkrb5support.a
include $(PREBUILT_STATIC_LIBRARY)

## libcyassl
include $(CLEAR_VARS)
LOCAL_MODULE := libcyassl
LOCAL_SRC_FILES := lib/libcyassl.a
include $(PREBUILT_STATIC_LIBRARY)
```

```
## Kerberos Test Application JNI Library (KerberosApp)
include $(CLEAR_VARS)

LOCAL_MODULE     := gsswrapper
LOCAL_C_INCLUDES := $(LOCAL_PATH) \
                    $(LOCAL_PATH)/include \

LOCAL_SRC_FILES  := gsswrapper_wrap.c \


LOCAL_CFLAGS     := -DKRB5_DEPRECATED=1 -DKRB5_PRIVATE -
   DANDROID -fno-common -Wall -Wcast-align -Wmissing-
   prototypes -Wno-format-zero-length -Woverflow -Wstrict-
   overflow -Wmissing-prototypes -Wreturn-type -Wmissing-
   braces -Wparentheses -Wswitch -Wunused-function -Wunused-
   label -Wunused-variable -Wunused-value -Wunknown-pragmas -
   Wsign-compare -Werror=uninitialized -Werror=declaration-
   after-statement -Werror=variadic-macros -Werror-implicit-
   function-declaration -Wstrict-aliasing -Wpointer-arith -
   Waddress
LOCAL_LDLIBS     := -llog

##LOCAL_STATIC_LIBRARIES := libkrb5 libcom_err libkadm5srv
   libkdb5 libk5crypto libkrb5support
LOCAL_STATIC_LIBRARIES := libkadm5srv_mit libkdb5 libkrb5_db2
    libgssrpc libgssapi_krb5 libkrb5 libk5crypto libcom_err
   libkrb5support libcyassl

include $(BUILD_SHARED_LIBRARY)
```

Literatura

[1] "Android Application Package," Android, Dostopano: 13.3.2014. Na voljo na spletu: http://developer.android.com/google/play/expansion-files.html

[2] T. Beale, "openEHR Architecture Architecture Overview," april 2007, Dostopano: 20.1.2014. Na voljo na spletu: https://github.com/openEHR/specifications/blob/master/publishing/architecture/overview.pdf

[3] T. Beale, "openEHR ADL 1.5 training 1/8 - Introduction," september 2012, Dostopano: 3.1.2014. Na voljo na spletu: http://www.youtube.com/watch?v=t4cjFkJo4R4

[4] T. Beale, "openEHR ADL 1.5 training 8/8 - Templates," september 2012, Dostopano: 25.1.2014. Na voljo na spletu: http://www.youtube.com/watch?v=WayGRCtc8ys

[5] L. T. Cavalini, "FHIES 2013 Presentation on MLHIM," oktober 2013, Dostopano: 15.1.2014. Na voljo na spletu: http://www.youtube.com/watch?v=S6nd7ZqSGds

[6] R. Cushman, A. M. Froomkin, A. Cava, P. Abril, in K. W. Goodman, "Ethical, legal and social issues for personal health records and applications," *Journal of Biomedical Informatics*, zv. 43, št. 5, str. 51–55, oktober 2010.

[7] A. Dogac, T. Namli, A. Okcan, G. Laleci, Y. Kabak, in M. Eichelberg, "Key Issues of Technical Interoperability Solutions in eHealth," str. 1–6, 2006.

[8] "Representational State Transfer (REST)," Donald Bren School of Information and Compute Sciences, Dostopano: 13.3.2014. Na voljo na spletu: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[9] "ISO EN 13606," EN 13606 Association, Dostopano: 5.3.2014. Na voljo na spletu: http://www.en13606.org/the-ceniso-en13606-standard

[10] "Tree," Gigaflop, Dostopano: 13.3.2014. Na voljo na spletu: http://www.gigaflop.co.uk/comp/chapt3.shtml

[11] "HL7CDA," Health Level Seven International, Dostopano: 8.3.2014. Na voljo na spletu: http://www.hl7.org/index.cfm?ref=nav

[12] Ingrid Lunden, Tech Crunch, Dostopano: 16.1.2014. Na voljo na spletu: http://techcrunch.com/2013/07/01/android-led-by-samsung-continues-to-storm-the-smartphone-market-pushing-a-global-70-market-share/?ncid=tcdaily

[13] C. Martínez-Costa, M. Menárguez-Tortosa, in J. T. Fernández-Breis, "An approach for the semantic interoperability of ISO EN 13606 and openEHR archetypes," *Journal of Biomedical Informatics*, zv. 43, št. 5, str. 736–746, oktober 2010.

[14] "MIT Kerberos GSS-API Java Interface," MIT, Dostopano: 12.1.2014. Na voljo na spletu: https://github.com/cconlon/kerberos-java-gssapi

[15] "Android," Open Handset Alliance, Dostopano: 13.3.2014. Na voljo na spletu: http://www.openhandsetalliance.com/android_overview.html

[16] openEHR, "ADL Workbench," Dostopano: 12.3.2014. Na voljo na spletu: http://www.openehr.org/downloads/ADLworkbench/home

[17] "openEHR," openEHR, Dostopano: 11.3.2014. Na voljo na spletu: http://www.openehr.org/

[18] "Mobilna aplikacija Jaz Poročam," Play Store, Dostopano: 15.3.2014. Na voljo na spletu: https://play.google.com/store/apps/details?id=com.makroplus.jazporocam

[19] I. D. Păun, D. G. Sauciuc, N. O. Iosif, O. Stan, A. Perşe, C. Dehelan, in L. Miclea, "Local EHR Management Based on openEHR and EN 13606," *Journal of Medical Systems*, zv. 35, št. 4, str. 585–590, avgust 2011.

[20] Satya Komatineni in Dave MacLean, *Pro Android 4*. 233 Spring Street, 6th Floor, New York, NY 10013: Appress, 2012.

[21] P. Schloeffel, T. Beale, G. Hayworth, S. Heard, in L. Heather, "The relationship between CEN 13606, HL7, and openEHR," *HIC 2006 and HINZ 2006: Proceedings*, zv. 24, str. 24–28, 2006. Na voljo na spletu: http://my.openehr.org/wiki/download/attachments/2949261/005_schloeffel.pdf

[22] Scott Wambler, "Mapping Objects to Relational Databases: O/R Mapping In Detail," Agile Data, Dostopano: 12.2.2014. Na voljo na spletu: http://www.agiledata.org/essays/mappingObjects.html

[23] "SQLite," SQLite, Dostopano: 13.3.2014. Na voljo na spletu: https://sqlite.org/

[24] "Tape," Square, Inc., Dostopano: 13.3.2014. Na voljo na spletu: http://square.github.io/tape/

[25] "XML Path Language," W3C, Dostopano: 13.3.2014. Na voljo na spletu: http://www.w3.org/TR/xpath/

[26] "XQuery 1.0: An XML Query Language," W3C, Dostopano: 13.3.2014. Na voljo na spletu: http://www.w3.org/TR/xquery/

[27] Wei-Meng Lee, *Beginning Android 4 Application Development.* 10475 Crosspoint Boulevard Indianapolis, IN 46256: John Wiley & Sons, Inc., 2012.