Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Nejc Korasa

Računanje učinkovin ter aplikabilnost pri elektronskem predpisovanju zdravil

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in matematika

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Področju informatizacije zdravstva (eZdravje) v Sloveniji že kar nekaj časa namenjamo veliko pozornosti. V okviru eZdravja se uvajajo sodobne in večstransko uporabne informacijske rešitve in se izvaja integracija lokalnih informacijskih sistemov. Pri tem igra pomembno vlogo standard openEHR, ki se uporablja za shranjevanje zdravstvenih podatkov. Eno izmed pomembnih problemskih področij je tudi širša podpora predpisovanju zdravil, kjer lahko pride do številnih napak in bi jih lahko z informatizacijo omilili oz. odpravili. V diplomskem delu naj študent predstavi primer rešitve takšne informacijske podpore z računanjem učinkovin, kjer naj bo podprt šifrant enot, šifrant učinkovin, medsebojna pretvorba, administracija zdravil, shranjevanje podatkov na standardiziran način (openEHR). Ustreznost rešitve naj študent objektivno ovrednoti in preveri na realnih podatkih.

Zahvaljujem se mentorju za podporo, znanje in predloge za razvoj sistema. Zahvalil bi se tudi družini, prijateljem, Sanji Zagorc in sodelavcem podjetja Marand d.o.o.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ADE	Adverse drug events	Neželeni dogodki z zdravili
ATC	Anatomical Therapeutic Chemical	Anatomska terapevtska kemikalija
AMP	Actual Medicinal Product	Dejansko zdravilo
AMPP	Actual Medicinal Product Pack	Dejansko pakiranje zdravila
API	Application Programming Interface	Vmesnik uporabniškega programa
BPM	Business process management	Upravljanje poslovnih procesov
BPMN	Business Process Model and Notation	Model poslovnih procesov in zapisov
CBZ	Central database of medicines	Centralna baza zdravil
CDS	Clinical decision support	Sistem za podporo pri kliničnem odločanju
CRUD	Create, Read, Update, Delete	Ustvari, beri, posodobi, izbriši
DMD	Dictionary of Medicines and Devices	Slovar zdravil in pripomočkov
EHR	Electronic health record	Elektronski zdravstveni zapis
FDB	First Data Bank	
IHE	Integrating the Healthcare Enterprise	
JSON	JavaScript Object Notation	JavaScript objektna notacija

OTC	Over-the-counter	Zdravila brez recepta
POJO	Plain Old Java Object	Osnovni Java objekt
RDBMS	Relational Database Management System	Sistem za upravljanje z relacijskimi podatkovnimi bazami
REST	Representational state transfer	Reprezentativni protokol prenosa
SNOMED CT	Systemised Nomenclature of Medicine	Sistematizirana nomenklatura medicine
SQL	Structured Query Language	Strukturiran poizvedovalni jezik
VMP	Virtual Medicinal Product	Virtualno zdravilo
VMPP	Virtual Medicinal Product Pack	Virtualno pakiranje zdravila
VTM	Virtual Therapeutic Moiety	Virtualna terapevtska oblika
XML	Extensible Markup Language	Razširljiv označevalni jezik

Povzetek

Naslov: Računanje učinkovin ter aplikabilnost pri elektronskem predpisovanju zdravil

Avtor: Nejc Korasa

Diplomska naloga opisuje rešitev preračuna aplicirane učinkovine zdravila v aplikacijah za elektronsko predpisovanje zdravil. Izpostavi pogoste probleme takšnih aplikacij in predstavi njihove rešitve. Ključna področja za uspešnost rešitve so pravilno strukturiran, večnivojski šifrant zdravil, šifrant enot in možnost medsebojne konverzije ter sistem za kreiranje in apliciranje administracij zdravil. Sistemi za elektronsko predpisovanje zdravil zelo dobro sovpadajo z uporabo openEHR standarda, ki predstavlja standardizirano strukturo shranjevanja kliničnih podatkov na mednarodnem nivoju. Diplomsko delo predstavi uporabo takšnega sistema kot rešitev za preračun prekomerne doze aplicirane učinkovine v aplikacijah za elektronsko predpisovanje zdravil. Predstavljeni so primeri uporabe v produktu Think!Meds. Opravljena je bila tudi statistična analiza takšnega sistema pri uporabi opozoril o možnem predoziranju učinkovine paracetamol na realnih podatkih bolnišnice.

Ključne besede: openEHR, elektronsko predpisovanje zdravil, podpora za klinične odločitve, šifrant zdravil, paracetamol.

Abstract

Title: Computation of medication ingredients and applicability in electronic prescribing of medicinal products

Author: Nejc Korasa

The thesis describes the solution of calculating the applied medication ingredients in the electronic prescribing applications. It presents the common problems of such applications and their solutions. The key areas for the success are a properly structured, multi-level drug dictionary, unit dictionary with mutual conversion option and a system for creating and administering medications. Electronic prescribing systems coincide very well with the use of an openEHR standard, which represents a standardized structure for storing clinical data at the international level. The thesis presents the applicability of such a system as a solution for computing an overdose of an applied medication ingredient in applications for electronic prescribing. Solution is presented as used in the Think!Meds product. A statistical analysis of such a system was also performed on the real data of the hospital. System is implemented as clinical decision support for warnings about possible overdose of paracetamol.

Keywords: openEHR, electronic prescribing, clinical decision support, drug dictionary, paracetamol.

Poglavje 1

Uvod

1.1 Motivacija

Za opis tematike v mojem diplomskem delu sem se odločil zaradi sodelovanja s podjetjem Marand d.o.o. Pri omenjenem podjetju sem del ekipe, ki razvija aplikacijo za predpisovanje zdravil poznano pod imenom Think!Meds, na angleškem trgu tudi kot OPENeP. V zadnjih letih sem dobil veliko vsebinskega znanja s tega področja in se srečal z mnogimi izzivi, katere smo morali rešiti pri implementaciji.

Poleg tega, se je v zadnjih letih na trgu pojavilo veliko zanimanja po aplikacijah za elektronsko predpisovanje zdravil. Tak pristop bolnišničnim ustanovam obljublja veliko koristi. S polno implementacijo elektronskega predpisovanja zdravniško in pisarniško osebje znatno prihrani s časom. Iz rezultatov raziskave v Angliji pa je tudi razvidno, da tudi že različice implementacij elektronskega predpisovanja, ki zgolj nadomestijo rokopisne predpise z računalniško natisnjenimi, dosežejo te izboljšave [17].

Med najpogostejšimi napakami pri predpisovanju zdravil so opažene naslednje [16]:

- nepravilen odmerek,
- napačno zdravilo,

- prepozno aplicirano zdravilo,
- zdravila niso na razpolago,
- zdravilo dobi napačen pacient.

Najpogostejši vzroki za napake pa so:

- neberljiva ali pomanjkljivo napisana navodila,
- nepopolne informacije o zgodovini zdravljenja z zdravili ob sprejemu bolnika,
- ne uskladitev sprememb v terapiji ob odpustu bolnika,
- neupoštevanje bolnikove občutljivosti, alergij na določena zdravila
- neupoštevanje neželenih učinkov zaradi interakcij med zdravili, ki jih bolnik jemlje.

Uporaba elektronskega predpisovanja zelo dobro sovpada z uporabo openEHR sistema za shranjevanje podatkov. Podatki so shranjeni v mednarodno priznani strukturi in tako dostopni tudi preko ostalih aplikacij znotraj EHR ekosistema. Shranjevanje kliničnih podatkov po openEHR standardu je podprto tudi v Think!Meds aplikaciji. Zdravniki v Angliji so koristi enotno potrdili. EHR izboljša učinkovitost - tako prakso kot tudi boljšo uporabo časa [21].

1.2 Cilj diplomske naloge

Ključni del aplikacije za predpisovanje zdravil je tudi podpora za reševanje problemov/odločanja (angl. clinical decision support) v času procesa predpisovanja, takoj ko je to možno. Na ta način uporabniku priskrbimo informacije takoj, ko jih ta potrebuje.

Z uporabo sistema za klinične odločitve je z vidika zdravstvenega varstva možno odkriti naslednje koristi [22]:

- zmanjšanje napak pri zdravilih in neželenih medicinskih dogodkih,
- izboljšano upravljanje specifičnih akutnih in kroničnih bolezni,
- izboljšana personalizacija oskrbe posameznih pacientov,
- stroškovno učinkovita in primerna uporaba zdravil na recept,
- boljše poročanje in spremljanje neželenih dogodkov.

Informacije sistema za klinične odločitve so uporabniku na voljo preko enostavnega dostopa do vseh kliničnih podatkov o pacientu, ki so shranjeni po openEHR standardu. V kolikor je zdravilo, ki ga zdravnik želi predpisati v interakciji z alergijo, ki jo ima pacient, se ta podatek jasno pokaže na zaslonu. Pri predpisovanju novega zdravila sistem v ozadju preišče vsa predpisana zdravila in v primeru kontraindikacije ali nevarne interakcije s katerim izmed predpisanih zdravil podatek pokaže na zaslonu. Vse to je v Think!Meds aplikaciji omogočeno z uporabo Medi Span produkta za klinične odločitve.

V diplomskem delu želim opisati implementacijo sistema za klinične odločitve, ki obravnava možnost prekoračitve priporočene ali dovoljene doze poljubne učinkovine. Opisal bom ključne elemente, ki jih mora takšna rešitev vsebovati:

- Struktura šifranta zdravil. Uvoz iz nacionalnih virov.
- Implementacija šifranta enot, medsebojne konverzije in šifrant učinkovin.
- Shranjevanje in implementacija administracij zdravil.
- Shranjevanje kliničnih podatkov na standardiziran način (preko standarda openEHR). To vključuje shranjevanje terapijskih predpisov, administracij zdravil in podatkov o pacientu.
- Implementacija sistema za preračun aplicirane količine poljubne učinkovine. Razširitve takega sistema - izvajanje preračunov o možnosti prekoračitve dovoljene doze.

Vsi postopki bodo tudi opisani s konkretno implementacijo v produtku Think!Meds. Tam se sistem uporablja, da uporabnikom nudi obvestila o možni prekoračitvi priporočene dnevne doze paracetamola. Preračun je odvisen tudi od pacientovih podatkov:

- starost,
- teža.

1.3 Struktura

V poglavju 2 so predstavljene uporabljene tehnologije pri razvoju sistema. 3. poglavje opisuje ključne elemente aplikacije za elektronsko predpisovanje zdravil. Vsi elementi so opisani tudi s konkretno implementacijo v produktu Think!Meds. V poglavju 4 je opisana rešitev sistema. Predstavljena je kot nadgradnja osnovnih elementov elektronskega predpisovanja (opisanih v poglavju 3). Poglavje opisuje uvedbo šifranta enot, medsebojne pretvorbe enot, dodatnih struktur (doza za predpisovanje zdravil), ki omogočajo implementacijo preračuna. Opisana je arhitekturna implementacija preračuna, ki se uporablja v Think!Meds produktu. Predstavljena je tudi možnost nadgradnje sistema za preračun. V poglavju 5 so opisani načini uporabe sistema za preračun učinkovin kot pomoč pri kliničnih odločitvah v elektronskem predpisovanju zdravil, prikazovanje opozoril ob predpisovanju in administraciji zdravila. Predstavljena je uporaba sistema za preračun v produktu Think!Meds in analiza uporabe sistema za preračun na realnih podatkih bolnišnice. Preračun obravnava morebitno preseženo aplicirano dozo paracetamola v obdobju zadnjih 24 ur.

Poglavje 2

Uporabljene tehnologije

2.1 OpenEHR

Fundacija openEHR je bila ustanovljena leta 2001. Glavna ustanovitelja sta Ocean Informatics Pty Ltd, Avstralija in University College London, Anglija [15]. Predstavlja neprofitno mednarodno organizacijo, ki se ukvarja s preoblikovanjem zdravstvenih podatkov iz fizične oblike v elektronsko obliko. Zagotavlja univerzalno interoperabilnost med vsemi oblikami elektronskih podatkov. Primarni poudarek je na elektronski zdravstveni evidenci (EHR) in povezanih sistemih [14].

2.2 Think!Ehr Platform

Think!Ehr Platform [18] je produkt podjetja Marand d.o.o. Predstavlja podatkovno rešitev namenjeno shranjevanju, upravljanju, poizvedovanju, pridobivanju in izmenjavi strukturiranih elektronskih zdravstvenih zapisov, ki temeljijo na najnovejši različici specifikacij openEHR. Vse klinične informacije so shranjene v neodvisnih arhetipih in predlogah, kar omogoča standardni vnos in nalaganje podatkov s terminološko validacijo.

2.3 Think!Meds

Sistem za upravljanje z zdravili Think!Meds je celovita rešitev za elektronsko predpisovanje in administracijo zdravil. Namenjena je izboljšanju varnosti uporabe zdravil, zmanjšanju neželenih učinkov na zdravila (ADE) in izboljšanju komunikacije med zdravniškim osebjem. Poleg tega zagotavlja poln dostop do vseh strukturiranih kliničnih podatkov.

Think!Meds [19] temelji na platformi Think!EHR Platform. Podpira izmenjavo podatkov na podlagi standardnih odprtih podatkovnih standardov, vključno z IHE (angl. Integrating the Healthcare Enterprise) in openEHR. Think!Meds je bil razvit na podjetju Marand d.o.o. v sodelovanju z Univerzitetno pediatrično bolnišnico Slovenije. Uporablja se tudi na angleškem trgu v sodelovanju z NHS England.

2.4 Centralna baza zdravil

Centralna baza zdravil [2] je zbirka podatkov o zdravilih vključno z zdravili brez recepta. Centralno bazo zdravil urejata Javna agencija Republike Slovenije za zdravila in medicinske pripomočke ter Zakon o zdravstvenem zavarovanju in zdravstvenem varstvu. Centralna baza zdravil vsebuje podatke o gotovih zdravilih, vključno z OTC (angl. Over-the-counter) zdravili, podatke o razvrščenih galenskih pripravkih ter podatke o razvrščenih živilih za posebne zdravstvene namene. Na voljo so administrativni podatki, podatki o sestavi zdravil, o režimih izdaje, o odločbah, cenah, razvrstitvah na liste, podatki o ATC (angl. Anatomical Therapeutic Chemical) klasifikaciji, o medsebojno zamenljivih zdravilih ter podatki o najvišjih priznanih vrednostih. Podatki Centralne baze zdravil so javno dostopni. Omogočen je izvoz podatkov v obliki Microsoft Excel.

2.5 DMD šifrant zdravil

NHS slovar zdravil in pripomočkov je bil razvit preko partnerstva med NHS Connecting for Health in NHS Business Services Authority. Zagotavlja identifikacijo izdelkov za zdravila in pripomočke, ki se uporabljajo za primarno in sekundarno oskrbo. Uporaba tega posebnega identifikatorja olajšuje obdelavo in povračilo receptov, elektronsko predpisovanje in elektronski zapis bolnikov. DMD je podmnožica SNOMED CT [1] (angl. Systemised Nomenclature of Medicine) in je zato priznana kot NHS standard. DMD opisuje izdelke kot koncepte (imenovane virtualna zdravila ali VMP) in prave izdelke (imenovane dejanska zdravila ali AMP) [4].

2.6 Spring okolja

2.6.1 Spring

Spring Framework [13] je platforma za programski jezik Java, ki nudi celovito infrastrukturno podporo za razvoj aplikacij. Ukvarja se z infrastrukturo, omogoča izdelovanje aplikacij iz POJO (angl. Plain Old Java Object) objektov. To velja za programski model Java SE in Java EE.

2.6.2 Spring Boot

Spring Boot [10] olajša razvoj samostojnih aplikacij v Spring okolju na produkcijskem nivoju. Omogoča razvoj aplikacij z minimalno konfiguracijo. Omogoča neposredno vključitev spletnih strežnikov Tomcat, Jetty ali Undertow. Spring boot ne zahteva XML konfiguracije.

[1] SNOMED CT je celovita terminologija kliničnega zdravstvenega varstva, vir z znanstveno potrjeno klinično vsebino [9]

2.6.3 Spring Cloud

Spring Cloud [11] ponuja orodja za razvijalce, da hitro zgradijo nekaj skupnih vzorcev v porazdeljenih sistemih. To so naprimer upravljanje konfiguracije (angl. configuration management), odkrivanje storitev (angl. service discovery)... Uporaba Spring Cloud omogoča hitro vzpostavitev storitev in aplikacij, ki implementirajo te vzorce.

2.7 Hibernate Validator

Hibernate Validator [7] je odprtokodna knjižnica, ki implementira specifikacijo za validiranje razredov Bean Validation 2.0 (JSR 380) v programskem jeziku Java. Privzeta metoda za definiranje validacij so anotacije, ki se lahko razširijo z uporabo XML. Knjižnica ni vezana na določeno aplikacijsko stopnjo ali programski model. Na voljo je za programiranje aplikacij strežnikov in odjemalcev.

Poglavje 3

Osnovni elementi elektronskega predpisovanja zdravil

3.1 Šifrant zdravil

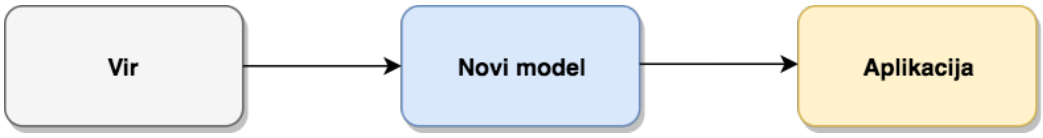
3.1.1 Mapiranje iz nacionalne baze zdravil

Ključni del aplikacije za elektronsko predpisovanje zdravil so podatki zdravil. Vir šifranta mora biti ažuren, struktura podatkovne baze v kateri so shranjeni podatki o zdravilih pa mora biti ustrezna. V nekaterih primerih je struktura teh podatkov enotna za eno bolnico, lahko pa si isti bazni model deli tudi večja mreža bolnic. Ponavadi je model definiran na območju države ali pa celo na mednarodnem nivoju.

- Za primer Slovenije šifrant zdravil izdaja **CBZ** (centralna baza zdravil). Centralna baza zdravil vsebuje podatke o gotovih zdravilih, vključno z OTC zdravili, podatke o razvrščenih galenskih pripravkih ter podatke o razvrščenih živilih za posebne zdravstvene namene [2].
- Za primer Anglije šifrant zdravil izdaja **DMD** (angl. Dictionary of Medicines and Devices).

Strokovna baza zdravil skrbi tudi za posodobitve šifranta ob pojavu sprememb. Pri zasnovi aplikacije za elektronsko predpisovanje je struktura ba-

znega modela zdravil zelo pomembna. Dobro zasnovan model ponudi možnost implementacije dodatnih funkcionalnosti, ki izboljšajo korist in uporabo aplikacije. V kolikor šifrant zdravil ne zadošča našim pogojem (je ploščat ali podatki niso dovolj razdrobljeni) je potrebno strukturo nadgraditi. Med šifrantom zdravil in aplikacijo ustvarimo dodaten nivo (lahko ga vidimo na sliki 3.1), ki predstavlja novi lokalni model in je izboljšana verzija prvotnega šifranta.



Slika 3.1: Uvedba dodatnega nivoja, ki predstavlja izboljšano bazno strukturo šifranta zdravil

Prvotni šifrant v tem primeru postane vir, aplikacija pa se zaveda samo novega modela. Med virom in novim modelom je potrebno implementirani ustrezno mapiranje in postaviti arhitekturo, ki omogoča komunikacijo med tema nivojema.

Idealno je, da se v primeru posodobitev šifranta v viru, le te avtomatsko prenesejo v novi model in tako postanejo dostopne aplikaciji. V praksi se implementacija takega sistema za avtomatsko posodabljanje baznega modela izkaže za zelo težavno. Glavna razloga sta dva:

- izvorni bazni model vsebuje premalo podatkov o zdravilu, ali pa so ti premalo razčlenjeni,
- podatkovna baza s katero je povezana aplikacija potrebuje poleg kliničnih podatkov tudi dodatne poslovne podatke.

Avtomatske posodobitve v tem primeru izpustimo in se izvajajo ročno ali pa te postanejo pol avtomatske. Na koncu jih oseba fizično pregleda, dopolni morebitne manjkajoče klinične ter poslovne podatke in potrdi spremembe.

3.1.2 Struktura podatkov

Dobra struktura je večnivojska. Tak primer modela je mogoče najti v DMD šifrantu in ga sestavlja 5 nivojev [3]:

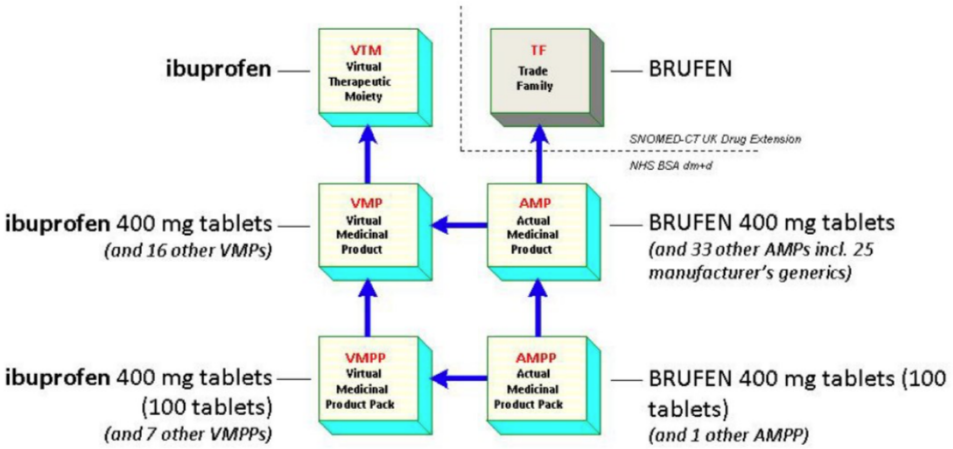
AMP - Actual Medicinal Product

AMPP - Actual Medicinal Product Pack

VMP - Virtual Medicinal Product

VMPP - Virtual Medicinal Product Pack

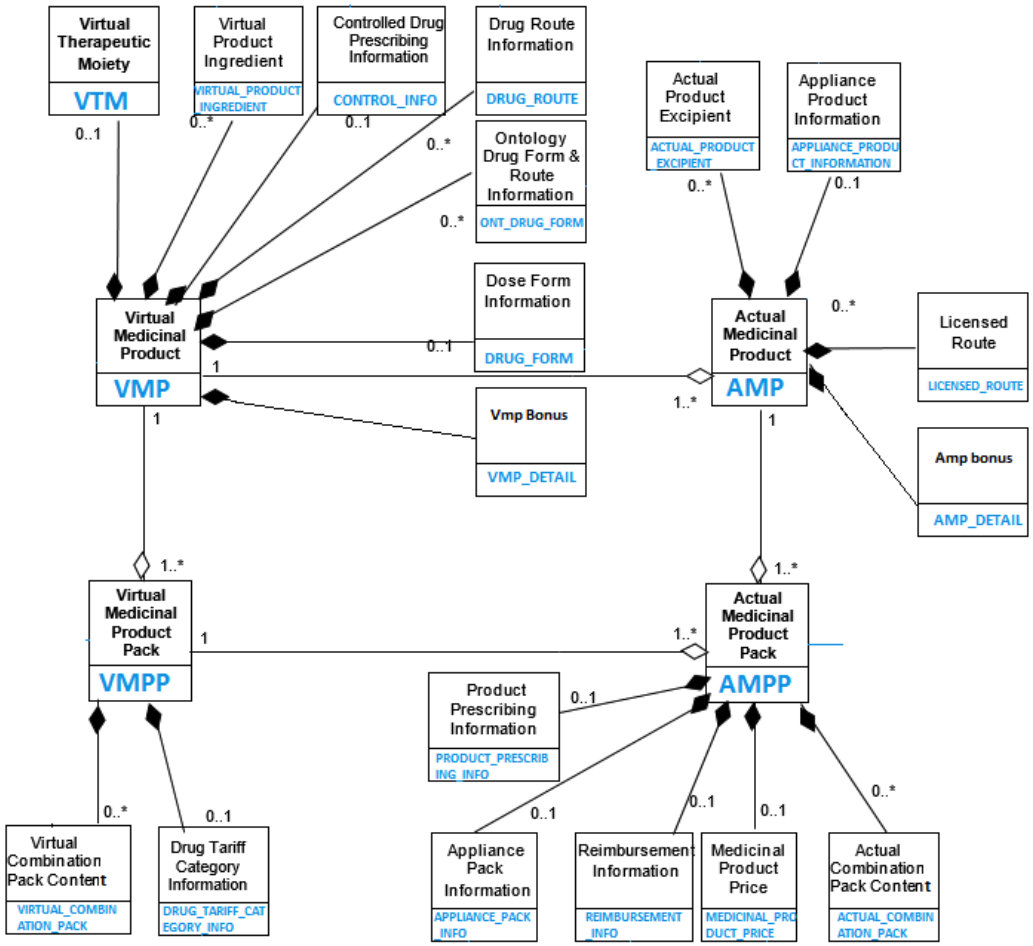
VTM - Virtual Therapeutic Moiety



Slika 3.2: Prikaz večnivojske strukture šifranta na produktu Ibuprufen [16]

Celotna relacijska struktura DMD šifranta je predstavljena na sliki 3.3. Tako predstavljen model vsebuje tudi dedovanje podatkov od vseh višjih nivojev. Tako zdravilo na AMP nivoju deduje vse podatke iz njegovega VMP in VTM nivoja.

Pri implementaciji Think!Meds smo odstranili nivoja produktov pakiranj AMPP in VMPP. Zaradi večnivojskih podatkov v šifrantu je najbolj ustrezno uporabiti SQL (angl. Structured Query Language) bazno okolje ali katero



Slika 3.3: Prikaz celotne strukture DMD sifranta

izmed relacijskih baznih struktur, ki temeljijo na RDBMS (angl. Relational database management system). Think!Meds aplikacija je kompatibilna z baznimi okolji:

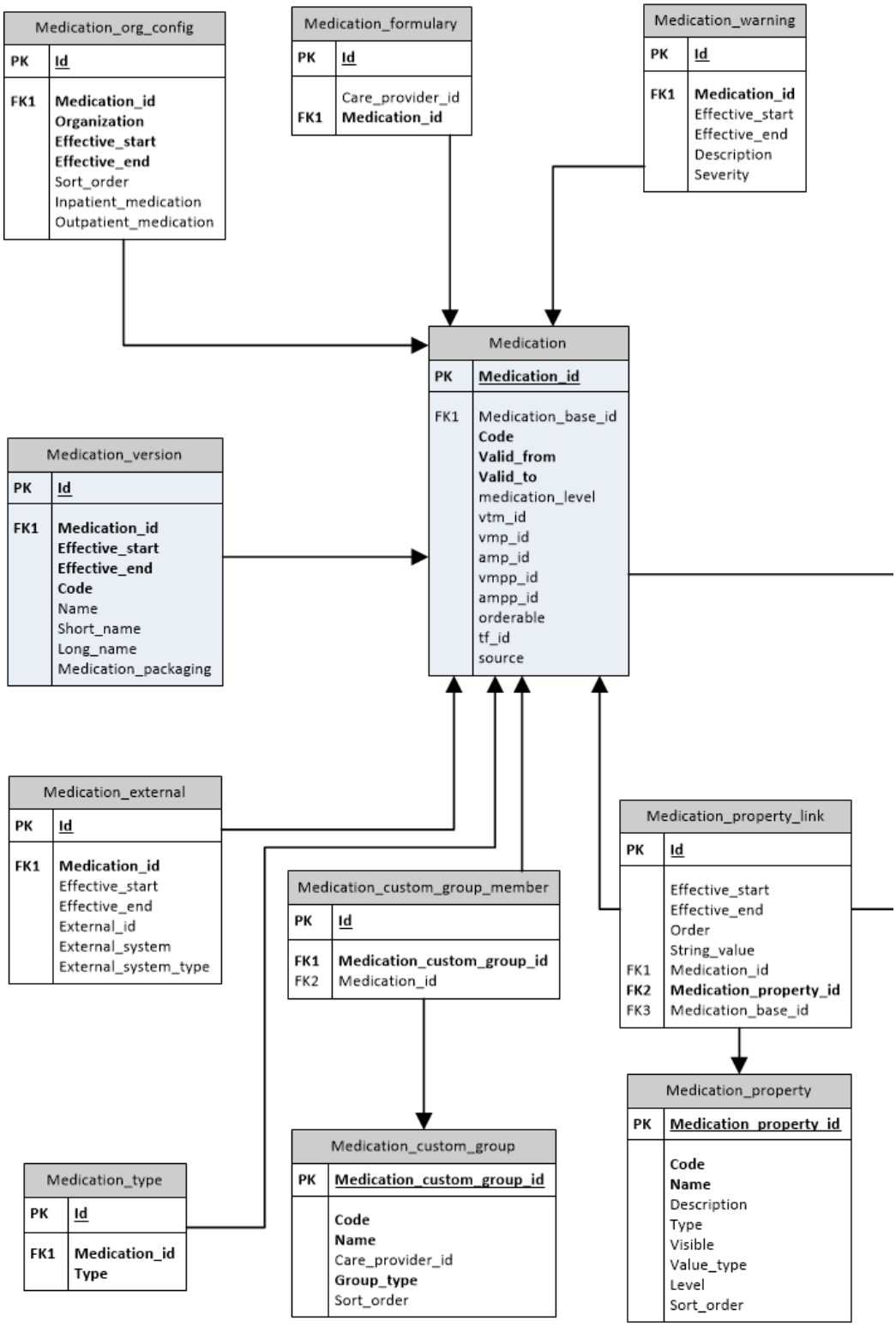
- Microsoft SQL Server,
- Oracle Database,
- PostgreSQL.

V razvoju pa se uporablja Oracle Database. Z relacijami med tabelami se

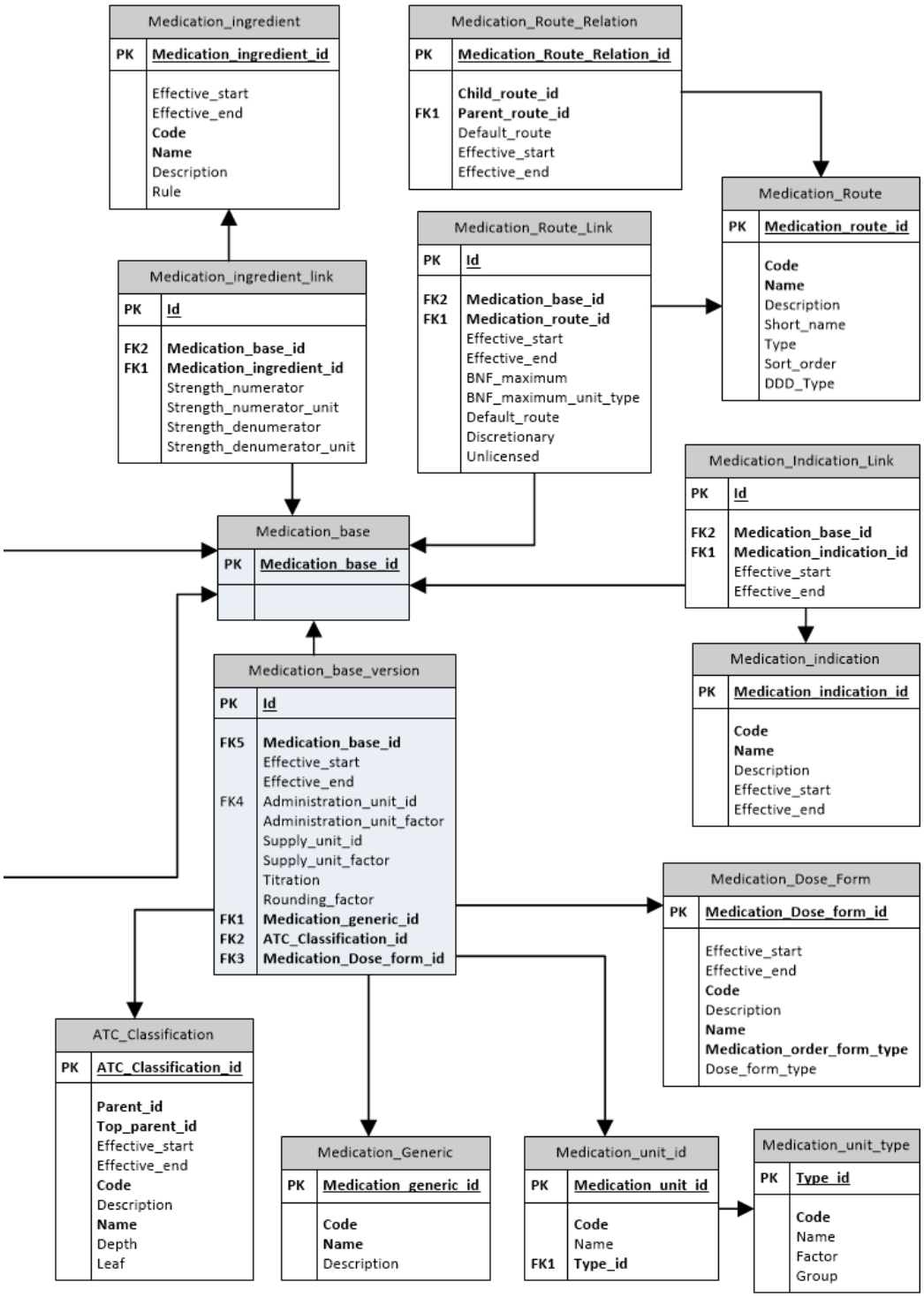
rešimo problema podvojenih podatkov. Tako so popravki na šifrantu veliko lažje izvedljivi saj so ti izolirani.

Struktura modela šifranta kot ga uporablja Think!Meds aplikacija lahko vidimo na slikah 3.4 in 3.5. Glavni podatki zdravila so predstavljeni v relacijskih tabelah povezanih na tabelo *MEDICATION BASE*, ki je zdravilom določena na VMP nivoju, kar posledično prinese naslednja opažanja in omejitve:

- Zdravila na VTM nivoju nosijo minimalni nabor podatkov. Predpisovanje zdravil na tem nivoju je omejeno. Zdravilo ne vsebuje dovolj podatkov za apliciranje. Manjka moč, razmerje učinkovin.
- Zdravila na AMP nivoju dedujejo podatke iz pripadajoče VMP starševske relacije.



Slika 3.4: Prikaz Think!Meds strukture šifranta - prvi del



Slika 3.5: Prikaz Think!Meds strukture šifranta - drugi del

3.1.3 Učinkovine

Učinkovine so eden izmed glavnih podatkov zdravila, saj definirajo njegovo moč. Moč zdravila mora biti vedno izražena kot vsota njegovih učinkovin. To pomeni, da mora vsaka učinkovina imeti definirano moč in njeno enoto. Ker nacionalni viri šifrantov zdravil take strukture v večini ne poznajo, je potrebno definirati lokalno shemo šifranta učinkovin, ki ga aplikacija uporablja.

Zdravilo je lahko sestavljeno iz več učinkovin. Povezava zdravila z učinkovino določa tudi njegovo količino. Količina je lahko predstavljena v več enotah. Najpogosteje se učinkovine določa v kombinacijah:

- liquid unit/mass unit (recimo 10 mg/1 ml) v primeru mešanic,
- mass unit/mass unit (recimo 5mg/10 mg) v primeru krem,
- mass unit (recimo 100 mg) v primeru tablet ali kapsul.

Potrebno je omeniti, da je število enot, v katerih je predstavljena učinkovina v zdravilu lahko več. Vira šifrantov CBZ in DMD uporabljata le dve. Ker model Think!Meds aplikacije izhaja iz DMD modela, se enoti imenujeta kar števec (angl. numerator) in imenovalec (angl. denominator). *MEDICATION INGREDIENT LINK* je torej sestavljen iz:

- moči števca (angl. numerator),
- enote števca (angl. numerator unit),
- moči imenovalca (angl. denominator),
- enote imenovalca (angl. denominator unit).

3.1.4 Nabor glavnih učinkovin zdravila

Nabor učinkovin v enem zdravilu ni vedno enakovreden. Učinkovine lahko razdelimo na glavne učinkovine in ostale učinkovine - recimo dodatki ali

mešanice. Taka razporeditev je zelo pomembna, to je najlažje predstavljeno s primerom.

Vzemimo za primer zdravilo **Paracetamol 500mg / Caffeine 65mg tablets**

Gre za zdravilo iz DMD šifranta na VMP nivoju. Zdravilo se predpisuje in aplicira v obliki tablete. Zdravilo sestavljata dve učinkovini:

- *Paracetamol (z močjo 500mg),*
- *Caffeine (z močjo 65mg).*

V obrazcu za predpisovanje zdravila bi lahko najenostavneje prikazali polje za vnos količine v enoti mg. Vnešena vrednost bi predstavljala vsoto učinkovin zdravila, ki ga moramo aplicirati. Taka implementacija za predpisovanje se izkaže za nevarno. Konkretno zdravilo se namreč predpisuje glede na količino učinkovine paracetamol, ki je v tem primeru glavna učinkovina zdravila. Opazimo, da moramo v modelu omogočiti označevanje glavnih učinkovin. Vsota teh predstavlja moč zdravila glede na katero predpisujemo. To dosežemo z atributom **main** na *MEDICATION INGREDIENT LINK*.

V našem primeru:

- *Paracetamol (z močjo 500mg)* - **main**,
- *Caffeine (z močjo 65mg).*

Če uporabnik vnese 1000mg vemo, da gre za predpis z 1000mg učinkovine *Paracetamol* in 130mg učinkovine *Caffeine*.

3.1.5 Enota predpisovanja zdravila

V teoriji bi, če želi uporabnik aplikacije predpisati izbrano zdravilo, na zaslonu ponudili okno za izbiro moči, ki bi bilo predstavljeno v enoti, kot je definirana na njegovih učinkovinah. V praksi se izkaže, da enota učinkovin ni vedno primerna za predpisovanje. V ta namen model vsebuje atributa:

- *administration unit* - zdravilu določimo enoto v kateri bo to predstavljeno pri predpisovanju in apliciranju zdravila,
- *administration unit factor* - zdravilu določimo razmerje med enoto za predpisovanje in učinkovinami zdravila.

Enota za predpisovanje ima vedno prednost pred enotami učinkovin - se upošteva, v kolikor je določena. Uporaba enote za predpisovanje je uporabna pri zdravilih, ki se predpisujejo v vnaprej določenih oblikah. To so bodisi tablete, kapsule, žličke... Primer izračuna:

Vzemimo za primer zdravilo Paracetamol 500mg / Caffeine 65mg tablets

Gre za zdravilo iz DMD šifranta na VMP nivoju. Zdravilo se predpisuje in aplicira v obliki tablete. Zdravilo sestavljata dve učinkovini:

- *Paracetamol (z močjo 500mg)* - **main**,
- *Caffeine (z močjo 65mg).*

Zdravilo ima določeno enoto za predpisovanje, in sicer tablet z faktorjem **1**. Pri predpisu tega zdravila moč določamo v obliki enote *tablet* z razmerjem:

1 tablet = 500mg Paracetamol + 65mg Caffeine

Če želimo podatke o predpisu uporabiti v preračunu lahko preko atributov *administration unit* in *administration unit factor* enostavno pridemo do količine predpisanih učinkovin.

3.2 Administracije zdravil

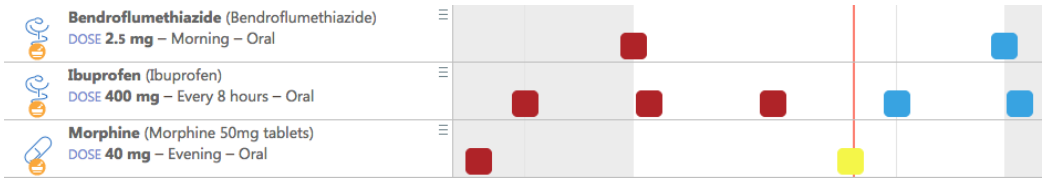
3.2.1 Kreiranje planiranih administracij zdravil

Rezultat predpisa zdravila je potrebno shraniti v strukturo, ki se odraža kot navodilo za apliciranje zdravila. V aplikacijah za predpisovanje zdravil

tako strukturo predstavimo z **opravili** (angl. task) za apliciranje zdravil. Eno opravilo predstavlja eno planirano apliciranje zdravila. Vsako opravilo vsebuje podatke:

- **predpis** h kateremu spada opravilo (ID predpisa),
- **zdravilo**, ki ga je potrebno aplicirati (ID zdravila),
- **čas** predvidene aplikacija zdravila,
- **moč** zdravila (hitrost zdravila, v kolikor je predpis infuzija ali terapija s trajanjem).

Prikaz kreiranih opravil za planirane administracije zdravil je na sliki 3.6.



Slika 3.6: Prikaz kreiranih opravil za planirane administracije zdravil

Opravila so shranjena v SQL bazi. Za strukturo in implementacijo shranjevanja se v Think!Meds aplikaciji uporablja Camunda [1] in bazira na BPMN modelu.

3.2.2 Avtomatsko kreiranje planiranih administracij

Kreiranje opravil se sproži avtomatsko takoj po predpisu terapije. Opravila se vedno kreirajo vse do konca trajanja terapije. V kolikor terapija nima določenega trajanja se opravila kreirajo za 7 dni vnaprej.

Za avtomatsko kreiranje opravil skrbi proces, ki se izvaja vsakih 24 ur. Ta naloži vse predpise, pogleda stanje pripadajočih administracij in kreira manjkajoča opravila v prihodnosti. Razred *AdministrationAutoTaskCreator.java* skrbi za avtomatsko kreiranje, algoritem je prikazan na sliki 3.7.

```java
public class AdministrationAutoTaskCreator
{
  private static final Logger LOG = LoggerFactory.getLogger(AdministrationAutoTaskCreator.class);

  private MedicationsOpenEhrDao medicationsOpenEhrDao;
  private AdministrationAutoTaskCreatorHandler administrationAutoTaskCreatorHandler;
  private MedicationsBo medicationsBo;

  @Autowired
  public void setMedicationsOpenEhrDao(final MedicationsOpenEhrDao medicationsOpenEhrDao)
  {
    this.medicationsOpenEhrDao = medicationsOpenEhrDao;
  }

  @Autowired
  public void setAdministrationAutoTaskCreatorHandler(
      final AdministrationAutoTaskCreatorHandler administrationAutoTaskCreatorHandler)
  {
    this.administrationAutoTaskCreatorHandler = administrationAutoTaskCreatorHandler;
  }

  @Autowired
  public void setMedicationsBo(final MedicationsBo medicationsBo) { this.medicationsBo = medicationsBo; }

  @Transactional
  @Scheduled(cron = "${meds.auto-task-creator-cron}")
  public void run()
  {
    SecurityContextHolder.getContext().setAuthentication(new StaticAuth("Think!Meds", "Think!Meds,", "Think!Meds,"));
    createAdministrationTasks(new DateTime());
  }

  private void createAdministrationTasks(final DateTime time)
  {
    final long startMillis = new DateTime().getMillis();

    final Map<MedicationOrderComposition, String> nonSuspendedInstructionsMap =
        medicationsOpenEhrDao.getActiveInstructionPairsWithPatientIds(time)
            .entrySet()
            .stream()
            .filter(entry -> !medicationsBo.isTherapySuspended(entry.getKey().getFirst(), entry.getKey().getSecond()))
            .collect(Collectors.toMap(e -> e.getKey().getFirst(), Map.Entry::getValue));

    final List<AutomaticAdministrationTaskCreatorDto> autoAdministrationTaskCreatorDtos =
        administrationAutoTaskCreatorHandler.getAutoAdministrationTaskCreatorDtos(time, nonSuspendedInstructionsMap);

    final Set<String> processedPatientIds = new HashSet<>();
    int count = 0;
    for (final AutomaticAdministrationTaskCreatorDto dto : autoAdministrationTaskCreatorDtos)
    {
      try
      {
        administrationAutoTaskCreatorHandler.createAdministrationTasksOnAutoCreate(dto, time);
        processedPatientIds.add(dto.getPatientId());
        count++;
      }
      catch (final Throwable t)
      {
        final String compUid = Opt.resolve(() -> dto.getTherapyDto().getCompositionUid()).get();
        LOG.error("Failed creating tasks - patientId: " + dto.getPatientId() + " compositionUid:"
                  + compUid + "\n" + ExceptionUtils.getFullStackTrace(t));
      }
    }

    for (final String patientId : processedPatientIds)
    {
      try
      {
        triggerEventForCache(patientId);
      }
      catch (final Throwable t)
      {
        LOG.error("Failed invalidation patient cache - patientId : " + patientId);
      }
    }

    final long endMillis = new DateTime().getMillis();
    LOG.debug("Successfully processed " + count + " therapies - TOOK: " + (endMillis - startMillis) + " ms");
  }
```

Slika 3.7: Razred *AdministrationAutoTaskCreator.java* skrbi za avtomatsko kreiranje planiranih administracij

3.2.3 Potrjevanje administracij zdravil

Vsako opravilo zahteva potrditev in tako postane zaključeno. Zaključenost opravila se odraža v različnih stanjih:

- *COMPLETED* (aplikacija je bila uspešna),
- *DEFERED* (aplikacija je bila preložena),
- *NOT GIVEN* (aplikacija ni bila izvedena).

Ko je opravilo aplikacije potrjeno, se njegovi podatki zmapirajo v mednarodno strukturo, ki se zapiše v openEHR. S tem podatki postanejo klinični. Pri potrditvi aplikacije ima uporabnik možnost spremembe podatkov. Opravilo lahko potrdi ob drugačnem času, z drugo močjo zdravila.

3.2.4 Predstavitev moči zdravila v administracijah

Podobno kot je definirana moč zdravila v šifrantu, je definirana tudi moč aplikacije. Ta je sestavljena v dveh enotah (numerator in denominator). Predstavlja jo razred *TherapyDoseDto.java* katerega atributi so pokazani na sliki 3.8.

```
public class TherapyDoseDto extends DataTransferObject
{
  private TherapyDoseTypeEnum therapyDoseTypeEnum;
  private Double numerator;
  private String numeratorUnit;
  private Double denominator;
  private String denominatorUnit;
```

Slika 3.8: Predstavitev moči aplikacije zdravila z razredom *TherapyDoseDto.java*

Za shranjevanje potrjenih aplikacij se uporablja EHR (angl. Electronic health record) predloga (angl. template):

- *Meds - Medication Administration*

Template definira arhetip:

- *openEHR-EHR-COMPOSITION.encounter.v1.*

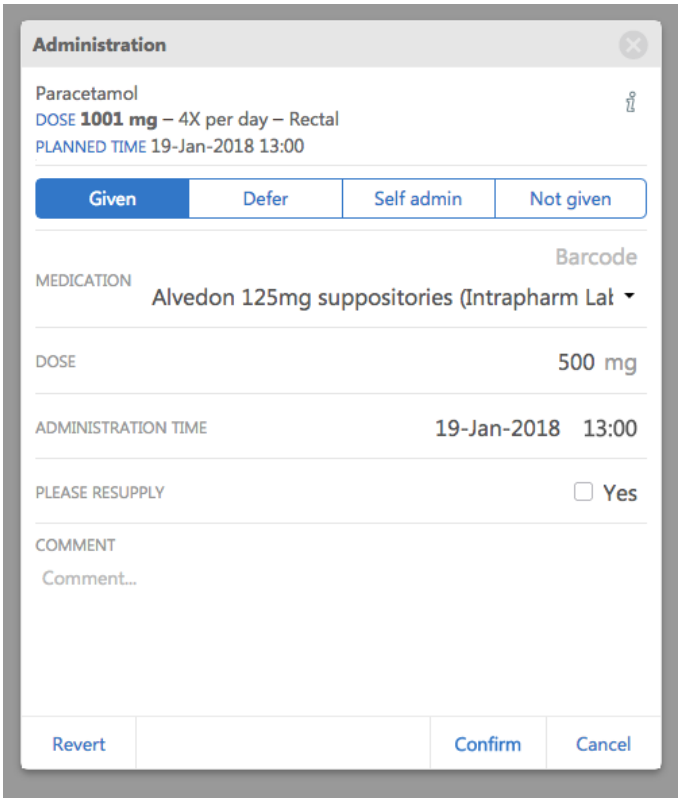
Primer mapiranja podatkov v EHR strukturo je prikazan na sliki 3.9.

```
final TherapyDoseDto therapyDose = administrationUtils.getTherapyDose(administrationDto);
if (therapyDose ≠ null && therapyDose.getNumerator() ≠ null)
{
  if (therapyDose.getTherapyDoseTypeEnum() == TherapyDoseTypeEnum.QUANTITY)
  {
    medicationAction.setStructuredDose(MedicationsEhrUtils.buildStructuredDose(
        therapyDose.getNumerator(),
        therapyDose.getNumeratorUnit(),
        therapyDose.getDenominator(),
        therapyDose.getDenominatorUnit(),
        null,
        null));
  }
  else if (therapyDose.getTherapyDoseTypeEnum() == TherapyDoseTypeEnum.RATE)
  {
    final InfusionAdministrationDetailsCluster infusionDetails = getInfusionDetails(therapyDose);
    if (!medicationAction.getAdministrationDetails().getInfusionAdministrationDetails().isEmpty())
    {
      infusionDetails.setPurposeEnum(
          medicationAction.getAdministrationDetails().getInfusionAdministrationDetails().get(0).getPurposeEnum());
    }
    medicationAction.getAdministrationDetails().getInfusionAdministrationDetails().clear();
    medicationAction.getAdministrationDetails().getInfusionAdministrationDetails().add(infusionDetails);
  }
  else if (therapyDose.getTherapyDoseTypeEnum() == TherapyDoseTypeEnum.VOLUME_SUM)
  {
    medicationAction.setStructuredDose(MedicationsEhrUtils.buildStructuredDose(
        therapyDose.getNumerator(),
        therapyDose.getNumeratorUnit()));
  }
}
else if (medicationAction.getStructuredDose() ≠ null
    && MedicationsEhrUtils.isDoseRangeStructuredDose(medicationAction.getStructuredDose()))
{
  medicationAction.setStructuredDose(null);
}
```

Slika 3.9: Mapiranje podatkov moči administracije v EHR strukturo

Enkratna aplikacija

Enkratna aplikacija je sestavljena iz enega arhetipa. Moč je predstavljena v največ dveh enotah (števec in imenovalec). Potrjevanje enkratne administracije v aplikaciji Think!Meds je predstavljeno na sliki 3.10.



Slika 3.10: Potrjevanje enkratne administracije v aplikaciji Think!Meds

Tipi kontinuiranih aplikacij

Kontinuirana aplikacija je sestavljena iz več arhetipov. Z različnimi statusi arhetipov lahko manipuliramo trajanje in količino apliciranja. V ta namen Think!Meds aplikacija definira naslednje tipe aplikacij:

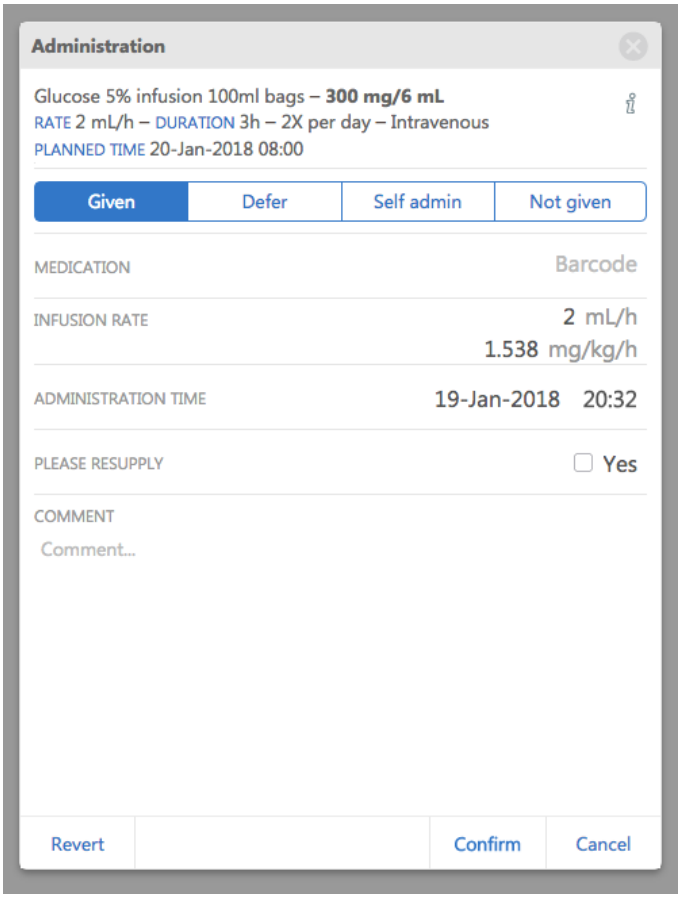
- *START* (določa začetek kontinuirane aplikacije),
- *STOP* (določa konec kontinuirane aplikacije),
- *ADJUST* INFUSION (določa spremembo hitrosti).

Prikaz planiranih kontinuiranih aplikacij je predstavljen na sliki 3.11.

Slika 3.11: Prikaz planiranih kontinuiranih administracij v aplikaciji Think!Meds

Moč kontinuiranih terapij je izražena s hitrostjo. Hitrost je definirana le z eno enoto. Zaradi večje varnosti se za apliciranje zdravil vedno uporablja predpis v mililitrih na uro *ml/h*. EHR template, ki shranjuje konec kontinuirane terapije (*STOP*), podatka o moči ne vsebuje. Ta služi le kot informacija o zaključku infuzije - infuzija je bila ustavljena.

Potrjevanje kontinuirane administracije v aplikaciji Think!Meds je predstavljeno na sliki 3.12.



Slika 3.12: Potrjevanje kontinuirane administracije v aplikaciji Think!Meds

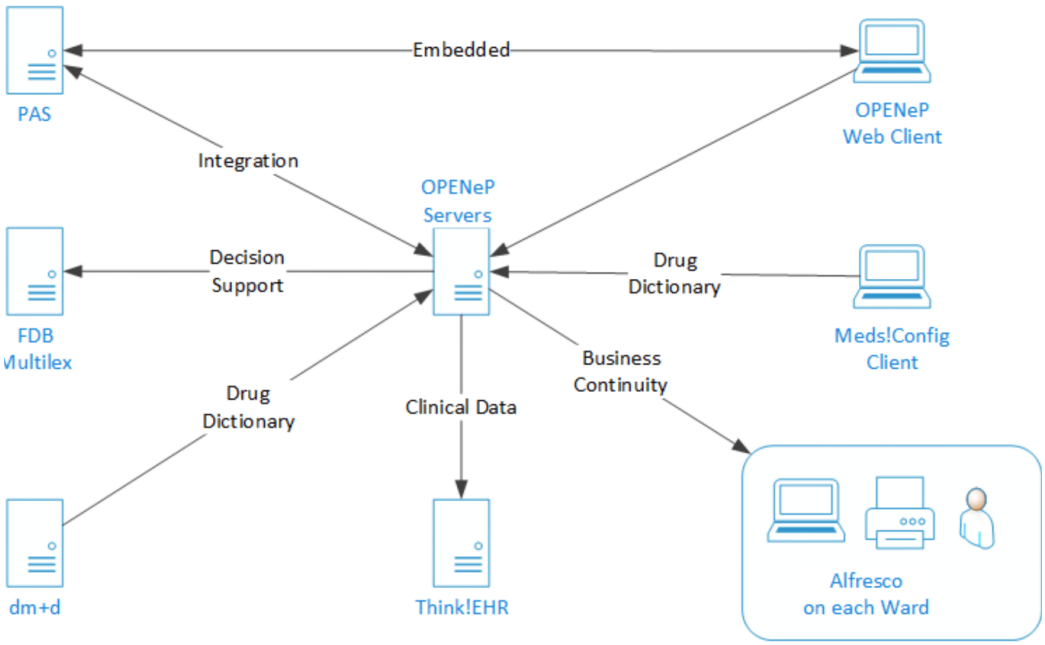
Aplikacije so v openEHR podatkovni bazi grupirane po predpisih. Kontinuirane aplikacije pa so opremljene še z dodatnim grupiranjem, ki določa en proces infuzije. Tako imamo ob analizi posamezne aplikacije, z enostavnim povzetkom v podatkovno bazo, na voljo vse aplikacije iste skupine.

3.3 Arhitekturna struktura produkta Think!Meds

Think!Meds je produkt podjetja Marand d.o.o, ki rešuje problem elektronskega predpisovanja in apliciranja zdravil. Produkt je ločen na:

- storitve, ki rešujejo prikazovanje podatkov - odjemalci (angl. frontend applications),
- storitve, ki skrbijo za procesiranje podatkov - strežniki (angl. backend applications).

Celotna struktura je definirana z diagramom 3.13.



Slika 3.13: Arhitekturna struktura Thnik!Meds produkta

3.3.1 OPENeP

Predstavlja skupino vseh strežnikov. Njihova arhitektura sledi arhitekturi mikrostoritev (angl. microservice architecture). Uporabljajo se kot zbirka ohlapno vezanih storitev, ki izvajajo poslovno logiko. Takšna arhitektura omogoča neprekinjeno dostavo/nameščanje (angl. Continious Delivery/Deploy) [8]. Strežniki so napisani v objektno usmerjenem programskem jeziku Java v okolju Spring Boot. Rešitve kot so izenačevanje obremenitve (angl.

load balancing), storitev odkrivanja (angl. service discovery) pa so implementirane z uporabo infrastrukture produkta Spring Cloud . Za to skrbijo odprtokodna ogrodja, ki jih je razvil Netflix [12]:

- protokol odkrivanja storitev (angl Service Discovery): *Eureka*,
- varovala (angl Circuit Breaker): *Hystrix*,
- deklerativni REST (angl Representational State Transfer) odjemalec: *Feign*,
- zunanja konfiguracija (angl External Configuration): *Spring Config Server*.

Skupino OPENeP sestavljajo naslednje mikrostoritve:

App server - skrbi za vso poslovno logiko aplikacije.

Bpm server - skrbi primarno za kreiranje planiranih administracij zdravil. Bazira na BPMN (angl. Business Process Model and Notation) modelu razvitem v produktu Camunda [1].

Meds!Config server - skrbi za urejanje šifranta zdravil, uvoz in mapiranje zdravil iz nacionalnega šifranta v lokalni model. Meds!Config upravlja z ločeno SQL bazo, kjer se shranjujejo vse verzije in popravki na zdravilih. Manipulacija s podatki poteka preko CRUD (angl. create, read, update, and delete) operacij.

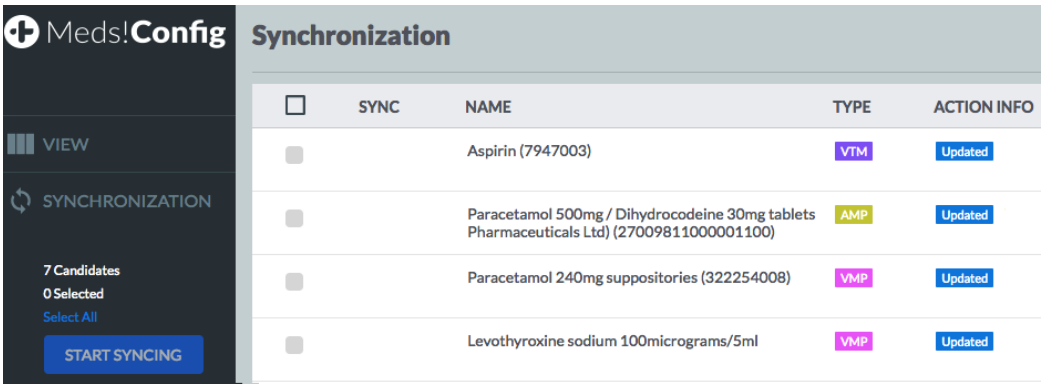
3.3.2 Meds!Config Client

Predstavlja odjemalsko aplikacijo ki je povezana z Meds!Config strežnikom. Preko spletne aplikacije uporabniki uredijo zdravila. Zdravila je možno opremljati tudi z nekliničnimi podatki.

Aplikacija ponuja sinhronizacijo zdravil iz Meds!Config SQL baze v OPENeP SQL bazo. Zdravila se najprej validirajo. Če je zdravilo veljavno

(ustreza vsem pravilom ki opisujejo veljavno strukturo zdravila), se iz baze prebere zadnja verzija zdravila in se preko REST klica pošlje na App server. Ta prebere zadnjo verzijo tega zdravila v OPENeP bazi in ga ustrezno posodobi.

Ena postavitev Meds!Config aplikacije lahko sinhronizira na različne OPENeP postavitve, v kolikor več bolnišnic uporablja isti šifrant zdravil. Primer vrste za sinhronizacijo je prikazan na sliki 3.14.



Slika 3.14: Primer vrste za sinhronizacijo v aplikaciji Meds!Config

3.3.3 FDB

FDB (angl. First Data Bank) [6] predstavlja zunanji vir podatkov za CDS (angl. Clinical decision support) omogoča prikazovanje opozoril pri predpisovanju in administraciji zdravil. Podpira opozorila o [5]:

- alergijah,
- kontraindikacijah,
- interakcijah z zdravili,
- podvojenih zdravilih.

Poglavje 4

Implementacija sistema za preračun učinkovin

4.1 Uvedba šifranta enot

Enostavna predstava šifranta enot je ena glavnih pomanjkljivosti šifrantov zdravil. Najpogosteje je enota predstavljena samo z imenom. Enote v takih baznih shemah ne poznajo nobene relacije kar onemogoča pretvorbo med njimi. Ker so enote med zdravili večinoma različne (eno zdravilo je lahko definirano v miligramih in drugo v gramih) smo brez konverzije enot zelo omejeni in ne moremo izvajati preračunov.

Tak šifrant enot nam onemogoča analizo porabe posamezne učinkovine in ga je potrebno dopolniti. Potrebno je implementirati šifrant enot, ki omogoča pretvorbo. Think!Meds aplikacija uporablja predstavitev enot, ki nam omogoča konverzijo enot znotraj iste skupine. Takšna predstavitev je sestavljena iz naslednjih glavnih gradnikov:

- skupina enote (*UNIT GROUP*),
- tip enote (*UNIT TYPE*),
- enota (*UNIT*).

Začetni nabor glavnih gradnikov je določen vnaprej. Z upoštevanjem določenih pravil, pa zasnova šifranta omogoča kasnejše dodajanje vseh lastnih gradnikov.

Začetni nabor skupin je sestavljen iz:

- *LIQUID UNIT* - enote tekočin,
- *MASS UNIT* - masne enote,
- *TIME UNIT* - časovne enote,
- *SURFACE UNIT* - enote površin.

Koncept skupine nam določa opcijo za medsebojno pretvorbo. Enoti sta med sabo konvertibilni samo če pripadata isti skupini.

Začetni nabor tipov enot je sestavljen iz:

- skupina *LIQUID UNIT* (L, DL, CL, ML, MICRO L),
- skupina *MASS UNIT* (G, KG, MG, MICRO G, NANO G),
- skupina *TIME UNIT* (D, H, MIN, S),
- skupina *SURFACE UNIT* (M2).

Vsak tip enote pripada največ eni skupini. Pripadnost skupini ni obvezna, vendar v tem primeru konverzije ni mogoče izvajati. Tip enote poleg skupine vsebuje še faktor in ime. Faktor je ključni podatek, ki omogoča konverzijo. Znotraj ene skupine obstaja en tip enote, katerega faktor je enak 1. To je tip, ki predstavlja normiranje. Smiselno je, da je normiranec tip enote, ki je najpogosteje uporabljen, ni pa nujno - to sta recimo L, KG.

Ime predstavlja privzeto ime za prikaz enote. Ta se prikaže v aplikaciji za predpisovanje zdravil. Uporaba atributa za ime enote nam omogoča, da lahko isti šifrant enot uporabimo v različnih jezikih (tudi v primerih, kjer se ne uporabljajo mednarodne oznake enot, recimo v Ruščini)

Enota predstavlja razširitev tipa enote. To lahko zmodeliramo le kot drugačen zapis enote (recimo mikrogram in mg) ali pa kot predstavitev nove enote, ki ne spada v nobenega izmed tipov enot.

4.2 Pretvorba enot

Enote zdravil in njihovih učinkovin niso vedno skladne. Vpeljava šifranta enot znotraj podatkovne baze zdravil rešuje ta problem. Za preračun med dvema vrednostima, ki sta predstavljeni v enoti je dovolj, če sta enoti skladni zgolj v skupini enote. Rešitev pretvorbe na strežniku je prikazana na sliki 4.1.

Enote so predstavljene kot tekstovni tip (*String*). Za konverzijo enot je potrebno iz podatkovne baze najprej naložiti obe enoti, preveriti ali sta konvertabilni in izvesti konverzijo. Razred *MedicationUnitDto.java*, ki predstavlja posamezno enoto, nosi vse podporne metode za konverzijo (slika 4.2).

```java
public interface UnitsConverter
{

  /**
   * Convert {@param value} represented in {@param from} unit to {@param to} known unit type
   *
   * @throws IllegalStateException if units cannot be converted, verify conversion
   * using {@link UnitsConverter#isConvertible} first.
   */
  double convert(double value, @Nonnull String from, @Nonnull KnownUnitType to);

  /**
   * Convert {@param value} represented in {@param from} known unit type to {@param to} unit
   *
   * @throws IllegalStateException if units cannot be converted, verify conversion
   * using {@link UnitsConverter#isConvertible} first.
   */
  double convert(double value, @Nonnull KnownUnitType from, @Nonnull String to);

  /**
   * Convert {@param value} represented in {@param from} unit to {@param to} unit
   *
   * @throws IllegalStateException if units cannot be converted, verify conversion
   * using {@link UnitsConverter#isConvertible} first.
   */
  double convert(double value, @Nonnull String from, @Nonnull String to);

  /**
   * Convert {@param value} represented in {@param from} known unit type to {@param to} known unit
   *
   * @throws IllegalStateException if units cannot be converted, verify conversion
   * using {@link UnitsConverter#isConvertible} first.
   */
  double convert(double value, @Nonnull KnownUnitType from, @Nonnull KnownUnitType to);

  /**
   * Checks if unit {@param unit1} and unit {@param unit2} are convertible
   */
  boolean isConvertible(@Nonnull String unit1, @Nonnull String unit2);

  /**
   * Checks if unit {@param unit} and known unit type {@param knownType} are convertible
   */
  boolean isConvertible(@Nonnull String unit, @Nonnull KnownUnitType knownUnit);

  /**
   * Checks if unit {@param unit} is of group type {@param group}
   */
  boolean isGroupType(@Nonnull String unit, @Nonnull UnitGroupEnum group);

  /**
   * Checks if unit {@param unit} is of known unit type {@param knownUnit}
   */
  boolean isKnownUnit(@Nonnull String unit, @Nonnull KnownUnitType knownUnit);
}
```

Slika 4.1: Razred, ki implementira interface *UnitsConverter.java* skrbi za konverzijo med enotami

```
public boolean isConvertible(@Nonnull final MedicationUnitTypeDto unit)
{
  Preconditions.checkNotNull(unit, "unit");
  return group == unit.getGroup();
}

public double convert(final double value, @Nonnull final MedicationUnitTypeDto to)
{
  Preconditions.checkNotNull(to, "to");
  Preconditions.checkNotNull(factor, "units conversion factor for " + displayName + " is null");

  if (isConvertible(to))
  {
    return value * factor / to.getFactor();
  }
  throw new IllegalStateException("Units " + getName() + ", " + to.getName() + " are not compatible!");
}
```

Slika 4.2: Razred *MedicationUnitDto.java*, ki predstavlja posamezno enoto, nosi vse podporne metodo za konverzijo

4.3 Doza za predpisovanje zdravil

Uvedba doze za predpisovanje zdravil nam določa strukturo, ki je enotna za vsa zdravila. Kreiranje doze zdravila je na strežniku centralizirano v enem razredu. Razred *DoseUtils.java* se uporablja za kreiranje doze za predpisovanje posameznega zdravila in ob tem upošteva pravila opisana v poglavju 3.1.3. V kolikor so enote učinkovin v zdravilu različne, vendar so del iste skupine, se rezultat pretvori v enoto prve učinkovine. Prav tako se upošteva nabor glavnih učinkovin, če te obstajajo. Doza za predpisovanje zdravil predstavlja celotno moč zdravila in je pomembna pri preračunu. Na sliki 4.3 vidimo algoritem izračuna doze za predpisovanje zdravila iz nabora njenih učinkovin.

```
public PrescribingDoseDto buildDoseFromIngredients(@Nonnull final List<MedicationIngredientDto> ingredients)
{
  checkNotNull(ingredients, "ingredients");

  if (ingredients.isEmpty())
  {
    return null;
  }

  final MedicationIngredientDto first = ingredients.get(0);
  final String mainNumeratorUnit = first.getStrengthNumeratorUnit();

  return ingredients
      .stream()
      .map(i -> buildDoseFromIngredient(mainNumeratorUnit, i))
      .reduce(
          (d1, d2) ->
          {
            if (d1.getDenominator() == null ^ d2.getDenominator() == null)
            {
              throw new IllegalStateException(
                  "Either all, or non of ingredients denominators must exist!");
            }

            return new PrescribingDoseDto(
                d1.getNumerator() + d2.getNumerator(),
                d1.getNumeratorUnit(),
                d1.getDenominator(),
                d1.getDenominatorUnit());
          })
      .orElseThrow(() -> new IllegalStateException("Error building dose from ingredients!"));
}

private PrescribingDoseDto buildDoseFromIngredient(
    final String toNumeratorUnit,
    final MedicationIngredientDto ingredient)
{
  final String numeratorUnit = ingredient.getStrengthNumeratorUnit();

  if (!unitsConverter.isConvertible(numeratorUnit, toNumeratorUnit))
  {
    throw new IllegalStateException(
        "Units " + numeratorUnit + " and " + toNumeratorUnit + " are not convertible!");
  }

  final double numerator = unitsConverter.convert(
      ingredient.getStrengthNumerator(),
      numeratorUnit, toNumeratorUnit);

  return new PrescribingDoseDto(
      numerator,
      toNumeratorUnit,
      ingredient.getStrengthDenominator(),
      ingredient.getStrengthDenominatorUnit());
}
```

Slika 4.3: Algoritem izračuna doze za predpisovanje zdravila iz nabora njenih učinkovin

Primeri izračunov:

Učinkovine zdravila so izražene samo v eni enoti

Nabor učinkovin:

- *učinkovina 1 (z močjo 500mg)* - **main**,
- *učinkovina 2 (z močjo 65mg).*

Doza za predpisovanje znaša 500mg.

Učinkovine zdravila so izražene samo v dveh enotah

Nabor učinkovin:

- *učinkovina 1 (z močjo 5mg/1ml)* - **main**,
- *učinkovina 2 (z močjo 10mg/1ml)* - **main**,
- *učinkovina 3 (z močjo 20mg/1ml).*

Doza za predpisovanje znaša 15mg/1ml.

4.4 Validacije podatkov v šifrantu

Če želimo podatke v šifrantu zdravil uporabiti za preračun, moramo zagotoviti, da so podatki vedno veljavni. Za to potrebujemo definirana pravila. DMD določa nekaj pravil, ki poenostavijo ureditev šifranta in hkrati zmanjšujejo možnost napak pri predpisovanju in apliciranju zdravil. Spodnja pravila delujejo kot predpogoj za možnost preračuna aplicirane doze učinkovine:

- enote števcev učinkovin v enem zdravilu morajo biti del iste skupine,
- enote imenovalcev učinkovin v enem zdravilu morajo biti enake,
- vse enote učinkovin v enem zdravilu so normirane glede na imenovalec.

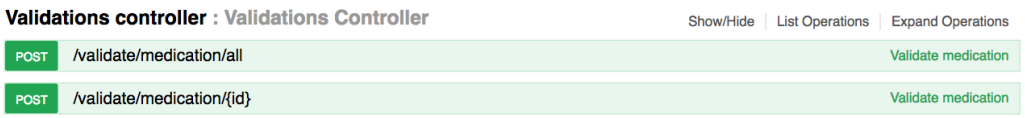
Pri uvažanju in urejanju šifranta je potrebno zagotoviti, da so podatki vedno validirani. Definiramo seznam validacij in pravil, ki jih je potrebno v šifrantu zagotoviti. Validacije pa izvajamo ob vsakem popravku v šifrantu.

Enostavne validacije pravil lahko podpremo že z uporabo anotacij v razredni strukturi šifranta. To dosežemo z uporabo knjižnice *Hibernate Validator*, ki deluje po principu *Bean Validation*. Primer uporabe anotacij za validiranje na razredu *MedicationBaseDto.java* je prikazan na sliki 4.4.

```
@ValidBase
public class MedicationBaseDto extends IdentityDto implements JsonSerializable, SharedDto
{
  private String uuid;

  @Valid private List<IngredientLinkDto> ingredients = new ArrayList<>();
  @Valid private List<IndicationLinkDto> indications = new ArrayList<>();

  @NotEmpty @Valid private List<RouteLinkDto> routes = new ArrayList<>();
  @Valid private List<PropertyLinkDto> properties = new ArrayList<>();
  @Valid private List<WarningDto> warnings = new ArrayList<>();

  @NotNull @Valid private MedicationBaseVersionDto baseVersion;
```

Slika 4.4: Primer uporabe anotacij za validiranje na razredu *MedicationBaseDto.java*

Slika 4.5 prikazuje strukturo REST API (angl. Representational State Transfer Application Program Interface) metod, na sliki 4.6 pa vidimo primer rezultata klica na REST API ob prisotnosti napak pri validaciji v JSON (angl. JavaScript Object Notation) strukturi.



Slika 4.5: Struktura REST API metod za validiranje podatkov o zdravilih

```
{
  "violations": [
    {
      "path": "baseVersion.doseForm",
      "message": "may not be null"
    },
    {
      "path": "",
      "message": "Either Administration unit or ingredients must be defined"
    },
    {
      "path": "routes",
      "message": "may not be empty"
    }
  ],
  "error": null,
  "objectId": 149189,
  "objectType": "MEDICATION",
  "time": "2018-01-21T14:24:07.949+01:00"
}
```


Slika 4.6: Primer rezultata klica na REST API metodo za validiranje podatkov o zdravilih, ob prisotnosti napak pri validaciji v JSON strukturi

4.5 Preračun aplicirane količine iz administracij zdravil

Za preračun aplicirane količine učinkovine v intervalu uporabljamo dva vira podatkov:

- podatki o zdravilih (vir: *šifrant zdravil*),
- podatki o aplikacijah zdravil (vir: *openEHR* - klinični podatki).

Za preračun so potrebni samo štirje vhodni podatki:

- pacient, za katerega preračun izvajamo,
- časovni interval v katerem izvajamo preračun,
- učinkovina, katere aplicirana količina nas zanima,
- enota v kateri želimo meriti podatke.

Prvi korak je pridobitev vseh aplikacij, ki so bile končane znotraj željenega intervala. Podatke lahko nato združimo na med samo neodvisne dele. Tak del je definiran kot:

- **enkratna aplikacija**,
- **kontinuirani tipi aplikacije iste skupine**.

Vsakega izmed delov lahko obravnavamo neodvisno in na koncu rezultate seštejemo.

4.5.1 Enkratne aplikacije zdravil

Zdravilo ima definirano enoto za predpisovanje

1. Naložimo zdravilo, ki je aplicirano in preberemo definirano enoto za predpisovanje in njen faktor.
2. Iteriramo skozi vse učinkovine zdravila, ki ustrezajo iskani učinkovini, jih konvertiramo v željeno enoto in seštejemo. S tem dobimo dozo iskanih učinkovin za predpisovanje zdravila.
3. Dozo učinkovin za predpisovanje zdravila zmnožimo z aplicirano dozo in njenim faktorjem. Dobljeno vrednost konvertiramo v željeno enoto.

Zdravilo nima definirane enote za predpisovanje

1. Naložimo zdravilo, ki je aplicirano in izračunamo dozo za predpisovanje zdravila.
2. Iteriramo skozi vse učinkovine zdravila, preberemo njihove moči, jih konvertiramo v željeno enoto. S tem dobimo osnovo za predpisovanje.
3. Izračunamo razmerje željene učinkovine v zdravilu. V kolikor zdravilo vsebuje le eno učinkovino je razmerje enako 1, sicer razmerje izračunamo. To je možno zaradi pravil šifranta, ki določajo,

da so enote moči vseh učinkovin med samo konvertabilne. Izračun razmerja učinkovine je viden na sliki 4.7.

4. Aplicirano dozo konvertiramo v iskano enoto in zmnožimo z razmerjem iskane učinkovine.

```
private double getIngredientPercentage(
    final MedicationDataDto medicationDataDto,
    final Long ingredientId,
    final MedicationRuleEnum ingredientRuleEnum,
    final KnownUnitType knownUnit)
{
  final List<MedicationIngredientDto> ingredients = medicationDataDto.getMedicationIngredients();
  if (ingredients.size() == 1)
  {
    final MedicationIngredientDto ingredient = ingredients.get(0);
    return isSearchIngredient(ingredientId, ingredientRuleEnum, ingredient) ? 1.0 : 0.0;
  }

  double ingredientQuantitySum = 0.0;

  final PrescribingDoseDto prescribingDose = medicationDataDto.getPrescribingDose();
  if (prescribingDose != null && unitsConverter.isConvertible(
      prescribingDose.getNumeratorUnit(),
      knownUnit))
  {
    final double quantitySum = unitsConverter.convert(
        prescribingDose.getNumerator(),
        prescribingDose.getNumeratorUnit(),
        knownUnit);

    for (final MedicationIngredientDto ing : ingredients)
    {
      if (isSearchIngredient(ingredientId, ingredientRuleEnum, ing)
          && unitsConverter.isConvertible(ing.getStrengthNumeratorUnit(), knownUnit))
      {
        final double ingredientQuantity = unitsConverter.convert(
            ing.getStrengthNumerator(),
            ing.getStrengthNumeratorUnit(),
            knownUnit);

        ingredientQuantitySum += ingredientQuantity;
      }
    }

    return ingredientQuantitySum / quantitySum;
  }

  return ingredientQuantitySum;
}
```

Slika 4.7: Izračun razmerja učinkovine v enem zdravilu

4.5.2 Kontinuirani tipi aplikacije zdravil iste skupine

Kontinuirane tipe aplikacije je potrebno obravnavati skupno, saj so med sabo odvisne. Vsaka kontinuirana skupina administracij se začne z administracijo tipa *START* in konča s tipom *STOP*. Vmes pa lahko obstaja poljubno število *ADJUST INFUSION* tipov.

Računanje trajanja aplikacije

Trajanje aplikacij je predstavljano z časovnim razmikom posameznih aplikacij. Če je bila *START* aplikacija aplicirana ob 8:00 in je naslednja *ADJUST INFUSION* aplikacija potrjena ob 10:00, trajanje znaša 2 uri.

Računanje trajanja infuzij je vidno na sliki 4.8.

```
private int calculateRateDurationInterval(
    final AdministrationDto administrationDto,
    final Interval searchInterval,
    final String therapyId,
    final Multimap<String, AdministrationDto> administrationDtoMap)
{
  if (administrationDto.getAdministrationTime().isAfter(searchInterval.getEnd()))
  {
    return 0;
  }

  final List<AdministrationDto> laterAdministrationsForTherapy =
      getLaterAdministrationsForTherapy(
          therapyId,
          administrationDto,
          administrationDtoMap);

  if (laterAdministrationsForTherapy.isEmpty())
  {
    final Interval administrationDurationInterval =
        new Interval(administrationDto.getAdministrationTime(), searchInterval.getEnd());

    final Interval overlapInterval = searchInterval.overlap(administrationDurationInterval);
    return Minutes.minutesIn(overlapInterval).getMinutes();
  }
  else
  {
    DateTime intervalEnd = searchInterval.getEnd();

    for (final AdministrationDto laterAdministrationDto : laterAdministrationsForTherapy)
    {
      if (laterAdministrationDto.getAdministrationType() == STOP
          || laterAdministrationDto.getAdministrationType() == ADJUST_INFUSION)
      {
        final DateTime laterAdministrationTime = laterAdministrationDto.getAdministrationTime();
        if (laterAdministrationTime != null && laterAdministrationTime.isBefore(intervalEnd))
        {
          intervalEnd = laterAdministrationTime;
        }
      }
    }

    final Interval administrationDurationInterval = new Interval(
        administrationDto.getAdministrationTime(),
        intervalEnd);

    final Interval overlapInterval = searchInterval.overlap(administrationDurationInterval);
    return Minutes.minutesIn(overlapInterval).getMinutes();
  }
}
```

Slika 4.8: Izračun trajanja kontinuirane administracije zdravila

Računanje moči aplicirane učinkovine

Izračunamo količino učinkovine v enem mililitru (slika 4.9), skupaj s podatkom o hitrosti aplikacije nato izračunamo količino iskane učinkovine. Označimo V kot hitrost, D trajanje aplikacije v minutah, $\mathcal{Q}$ količino iskane učinkovine in I količino iskane učinkovine v enem mililitru. Izračun:

$$\mathcal{Q} = \frac{IVD}{60}$$

```
private double getIngredientQuantityInOneMl(
    final MedicationDataDto medication,
    final Long ingredientId,
    final MedicationRuleEnum ingredientRuleEnum,
    final KnownUnitType knownUnit)
{
  double ingredientQuantity = 0.0;
  for (final MedicationIngredientDto ingredient : medication.getMedicationIngredients())
  {
    if (isSearchIngredient(ingredientId, ingredientRuleEnum, ingredient)
        && unitsConverter.isConvertible(ingredient.getStrengthNumeratorUnit(), knownUnit)
        && ingredient.getStrengthDenominator() != null)
    {
      final double ingredientInUnit = unitsConverter.convert(
          ingredient.getStrengthNumerator(),
          ingredient.getStrengthNumeratorUnit(),
          knownUnit);

      if (unitsConverter.isGroupType(ingredient.getStrengthDenominatorUnit(), LIQUID_UNIT))
      {
        final double ingredientMl = unitsConverter.convert(
            ingredient.getStrengthDenominator(),
            ingredient.getStrengthDenominatorUnit(),
            ML);

        ingredientQuantity += ingredientInUnit / ingredientMl;
      }
    }
  }

  return ingredientQuantity;
}
```

Slika 4.9: Izračun količine učinkovine v enem mililitru

4.5.3 Kontinuirani tipi aplikacije zdravil iste skupine na robu intervala

Aplikacije kontinuiranih tipov, ki se nahajajo na robih intervala zahtevajo dodatno delo.

V kolikor se aplikacija tipa *ADJUST INFUSION* ali *STOP* nahaja ob robu začetka intervala in ta ne vsebuje pripadajoče aplikacije tipa *START* je potreben ponovni povzetek v openEHR bazno strukturo, ki vrne manjkajoče aplikacije. Tak povzetek je enostaven in hiter, saj so aplikacije iste skupine med sabo povezane.

Nato se izračuna količina celotne skupine in odšteje razmerje količine, ki je bila aplicirana zunaj intervala.

Podobno se reši tudi aplikacije ob robu konca intervala.

4.6 Arhitekturna rešitev preračuna

Implementacijo preračuna lahko generaliziramo z enim razredom (slika 4.10).

```
public interface IngredientCalculator
{
  CalculateResult calculate(
      @Nonnull final Interval interval,
      @NotEmpty final Set<Long> ingredientIds,
      final Set<String> patientIds,
      final Set<Long> careProviderIds);
}
```

Slika 4.10: Generalizacija preračuna na strežniku

Dodatni vhodni parametri omogočajo razširitve preračuna:

- **patientIds**: Izvajanje preračuna za enega ali več pacientov hkrati. V primeru praznega seznama, se preračun izvaja za vse paciente.
- **ingredientIds**: Omogoča izvajanje preračuna za več učinkovin hkrati. Tako dobimo vsoto aplicirane količine vseh podanih učinkovin.

- **careProviderIds**: Omogoča izvajanje preračuna samo za izbrane bolnišnične oddelke.

Z dodatnimi parametri *patientIds*, *ingredientIds*, *careProviderIds* lahko preračun izvajamo za več pacientov, učinkovin ali samo za določene bolnišnične oddelke.

4.6.1 Uporaba dodatnih pravil

Rezultat aplicirane učinkovine lahko dopolnimo z dodatnimi pravili, ki ga ustrezno priredijo ali dopolnijo. To dosežemo z implementacijami razredov, ki razširjajo razred *IngredientRuleCalculator.java* (slika 4.11).

Če imamo dostop do demografskih podatkov v openEHR lahko implementiramo pravilo, ki določa maksimalno dovoljeno količino učinkovine glede na starost in težo pacienta (slika 4.12)

```java
@Component
public abstract class IngredientRuleCalculator
{
  private IngredientCalculator ingredientCalculator;

  public void setIngredientCalculator(final IngredientCalculator ingredientCalculator)
  {
    this.ingredientCalculator = ingredientCalculator;
  }

  /**
   * Apply custom rules to calculated result
   */
  protected abstract CalculateResult applyRule(CalculateResult calculateResult);

  public CalculateResult calculate(
      @Nonnull final Interval interval,
      @NotEmpty final Set<Long> ingredientIds,
      final Set<String> patientIds,
      final Set<Long> careProviderIds)
  {
    return applyRule(
        ingredientCalculator.calculate(
            interval,
            ingredientIds,
            patientIds,
            careProviderIds));
  }
}
```

Slika 4.11: Razred *IngredientRuleCalculator.java* omogoča uporabo dodatnih pravil pri preračunu

```
public class PatientAgeAndWeightRule extends IngredientRuleCalculator
{
  private final int patientAge;
  private final double patientWeight;

  private final int patientAgeLimit;
  private final double patientWeightLimit;

  private final int patientAgeDoseLimit;
  private final double patientWeightDoseLimit;

  public PatientAgeAndWeightRule(
      final int patientAge,
      final double patientWeight,
      final int patientAgeLimit,
      final int patientAgeDoseLimit,
      final double patientWeightLimit,
      final double patientWeightDoseLimit)
  {
    this.patientAge = patientAge;
    this.patientWeight = patientWeight;
    this.patientAgeLimit = patientAgeLimit;
    this.patientAgeDoseLimit = patientAgeDoseLimit;
    this.patientWeightLimit = patientWeightLimit;
    this.patientWeightDoseLimit = patientWeightDoseLimit;
  }

  @Override
  protected CalculateResult applyRule(final CalculateResult calculateResult)
  {
    if (patientAge > patientAgeLimit
        && patientAgeDoseLimit > calculateResult.getQuantity())
    {
      return new LimitCalculateResult(
          calculateResult,
          patientAgeLimit,
          true,
          "Patient age limit exceeded");
    }

    if (patientWeight > patientWeightLimit
        && patientWeightDoseLimit > calculateResult.getQuantity())
    {
      return new LimitCalculateResult(
          calculateResult,
          patientWeightLimit,
          true,
          "Patient weight limit exceeded");
    }

    return calculateResult;
  }
}
```

Slika 4.12: Implementacija pravila, ki določa maksimalno dovoljeno količino učinkovine glede na starost in težo pacienta

4.6.2 Ravnanje ob napakah med preračunavanjem

Prisotnosti napak v preračunavanju ne moremo izključiti. Ne glede na vzrok napak je potrebno, da je posledica le te čim bolj izolirana. Ker preračun aplicirane učinkovine razdelimo na več delov, tako da vsak del predstavlja izračun aplicirane učinkovine tekom ene aplikacije, potem so deli med sabo neodvisni. Prisotnost napake v enem delu preračuna ne vpliva na preračun ostalih delov.

Del, znotraj katerega je prišlo do napake in se preračun ne more izvesti, se označi kot neveljaven. Neveljavnost in s tem neupoštevanje posameznih aplikacij je ključna informacija, ki je uporabniku vidna poleg standardnega opozorila. Vsota preračunov vseh ostalih delov se sešteje.

Razred, ki predstavlja strukturo rezultata preračuna je prikazan na sliki 4.13. Vse napake med preračunom so shranjene v polju *errors*, izključene iz končne vsote in prikazane uporabniku (slika 4.14). Prisotnost napak v preračunu izoliramo z uporabo *CompletableFuture.java* (slika 4.15)

```java
public class CalculateResult
{
  private final Double quantity;
  private final String quantityUnit;
  private final List<CalculateError> errors = new ArrayList<>();

  public CalculateResult(final Double quantity, final String quantityUnit)
  {
    this.quantity = quantity;
    this.quantityUnit = quantityUnit;
  }

  public static CalculateResult error(
      final Throwable throwable,
      final String administrationId)
  {
    final CalculateResult result = new CalculateResult(null, null);
    result.addError(new CalculateError(administrationId, throwable.getMessage()));
    return result;
  }

  public Double getQuantity()
  {
    return quantity;
  }

  public String getQuantityUnit()
  {
    return quantityUnit;
  }

  public void addError(final CalculateError error)
  {
    this.errors.add(error);
  }

  public List<CalculateError> getErrors()
  {
    return Collections.unmodifiableList(errors);
  }
}
```

Slika 4.13: Razred, ki predstavlja strukturo rezultata preračuna

```
public class CalculateError
{
  private final String administrationId;
  private final String error;

  public CalculateError(final String administrationId, final String error)
  {
    this.administrationId = administrationId;
    this.error = error;
  }

  public String getAdministrationId()
  {
    return administrationId;
  }

  public String getError()
  {
    return error;
  }
}
```

Slika 4.14: Razred, ki predstavlja strukturo napake med preračunom

```
CompletableFuture
    .supplyAsync(() → calculateIngredientQuantity(administration))
    .exceptionally(throwable → CalculateResult.error(
        throwable,
        administration.getAdministrationId()));
```

Slika 4.15: Možnost napak v preračunu izoliramo z uporabo *CompletableFuture.java*

Poglavje 5

Evaluacija in testiranje sistema

5.1 Uporaba preračunov v rešitvah za predpisovanje zdravil

Največja uporabnost aplicirane količine učinkovine se v rešitvah za predpisovanje zdravil pojavi v dveh osnovnih scenarijih:

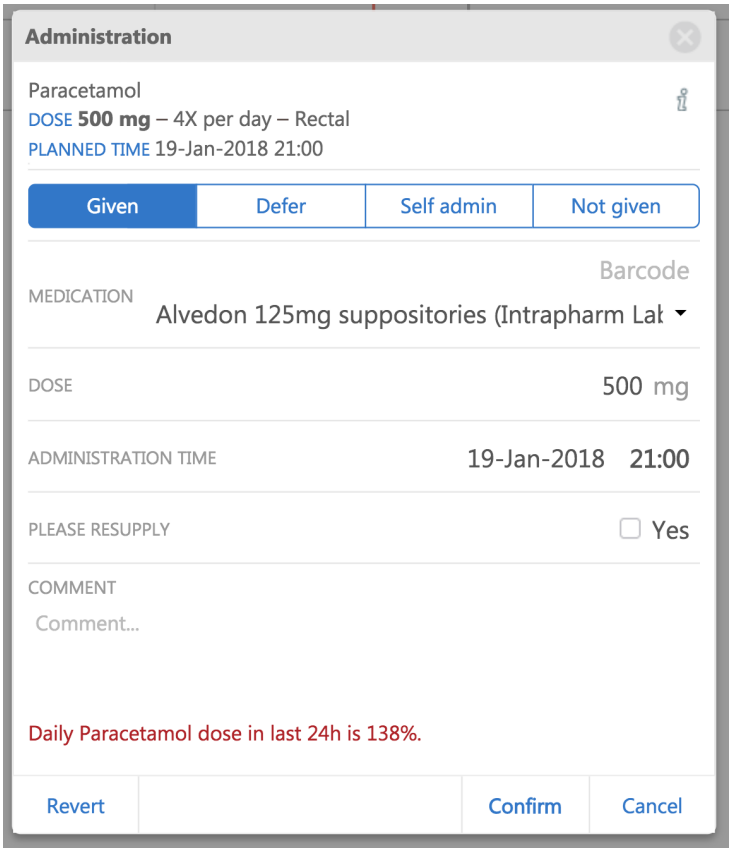
- **med predpisovanjem zdravila** (pred potrditvijo predpisa),
- **med apliciranjem zdravila** (pred potrditvijo aplikacije).

Rezultati preračuna se uporabniku pokažejo v obliki opozorila (slika 5.1). Nalaganje takšnih opozoril mora biti restriktivno - dokler opozorila niso prikazana na zaslonu mora biti akcija s katero bo uporabnik potrdil aplikacijo ali predpis zdravila onemogočena.

Če v preračun vključimo tudi trenutni predpis/aplikacijo, s tem predvidimo spremembo aplicirane količine, ki jo bo trenutna akcija povzročila.

5.2 Restrikcije ob preseženi maksimalni dozi

Opozorila delimo na različne stopnje glede na pomembnost njihove vsebine. S takšno delitvijo lahko določimo naslednja pravila [20]:

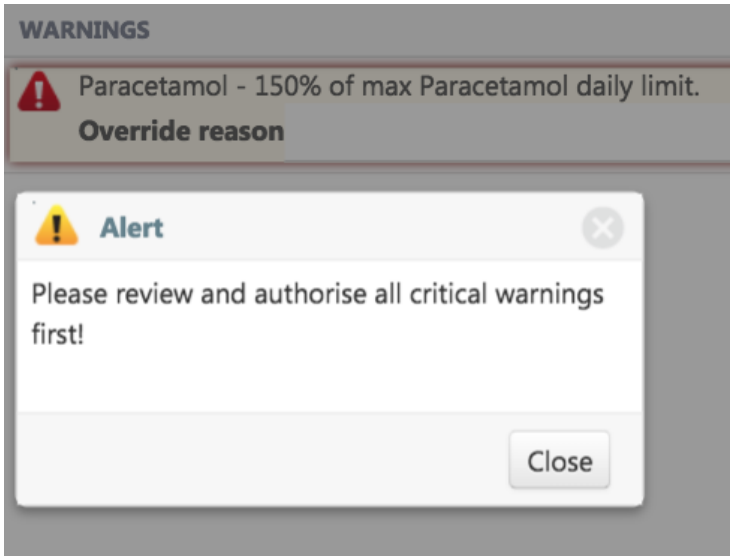


Slika 5.1: Prikaz opozorila o preseženi maksimalni dozi paracetamola v aplikaciji Think!Meds

- **Majhna stopnja pomembnosti** (angl. low severity) - Opozorilo nima nobene restrikcije.

- **Srednja stopnja pomembnosti** (angl. medium severity) - Uporabnik mora, če želi z akcijo nadaljevati, vnesti razlog za preklic opozorila (slika 5.2). Ta se poleg ostalih kliničnih podatkov shrani v openEHR.

- **Visoka stopnja pomembnosti** (angl. high severity) - Opozorilo uporabniku preprečuje nadaljevanje akcije.



Slika 5.2: Opozorilo v primeru nadaljevanja potrjevanja predpisa brez vnešenega razloga za preklic opozorila

5.3 Uporaba preračuna paracetamola v bolnišnici

Produkt Think!Meds uporablja sistem za preračun za učinkovino paracetamol. Preračuni se izvajajo ob postopku predpisa zdravila in ob administracijah. V kolikor zdravilo, ki ga uporabnik predpisuje/administrira vsebuje učinkovino paracetamol, se na strežnik pošlje zahtevek po preračunu aplicirane količine paracetamola v intervalu zadnjih 24 ur. Pri preračunu se upošteva tudi trenutni predpis/administracija.

Zgornja meja dovoljene količine paracetamola v 24 urah je določena glede na težo in starost pacienta. Tekom enega zahtevka za preračun se lahko tako primerja tudi 2 zgornji meji. Rezultat zahtevka predstavlja tisti izračun, katerega razmerje aplicirane količine je glede na dovoljeno količino največje. Algoritem je naslednji:

Zgornja meja glede na pacientovo starost

- pacient je mlajši od 18 let: *60mg/kg/dan*

Zgornja meja glede na pacientovo težo

- pacientova teža je večja od 50kg: *4000g*
- pacientova teža manjša ali enaka 50kg: *2000g*

Implementacija sistema za preračun dnevne doze paracetamola pa je v bolnišnici v uporabi od meseca maja 2016.

5.4 Analiza zahtevkov za preračun paracetamola na realnih podatkih

Opravljena je bila analiza podatkov o zahtevkih za preračun paracetamola na realnih podatkih v bolnišnici. Glavni cilj analize je bilo pridobiti podatek o razmerju zahtevkov med naslednjima stanjema:

- v preračunu **je bila** presežena maksimalna doza paracetamola v zadnjih 24 urah,
- v preračunu **ni bila** presežena maksimalna doza paracetamola v zadnjih 24 urah.

Statistična obdelava je bila izvedena na podatkih od *1.6.2017* do *1.12.2017*.

Potrebno je omeniti, da se zahtevek za preračun sproži ob vsaki akciji predpisa ali administracije učinkovine paracetamol, še preden se akcija potrdi. To pomeni da število rezultatov o preseženi maksimalni dozi ni enako številu apliciranih ali predpisanih zdravil, ki so presegli to mejo - uporabnik je akcijo namreč lahko preklical.

5.4.1 Rezultati analize

Obravnavanih je bilo **59.191** zahtevkov za preračun administriranje doze.

- V **57.475** primerih (97,1%), je bila administrirana doza **pod 100%** maksimalne dovoljene doze
- V **1.716** primerih (2,9%), je bila administrirana doza **nad 100%** maksimalne dovoljene doze

Opažanja so naslednja:

Razpršenost rezultatov o preseženi maksimalni dozi glede na čas v dnevu (glej sliko 5.3)

Pojavitev rezultatov o preseženi maksimalni dozi je najbolj zgoščena med *10:00* in *13:00*. Opazimo tudi nekaj velikih odstopanj, ko je bila presežena maksimalna doza izračunana med 600% in 1.600% - te izjeme najverjetneje predstavljajo napake v predpisu/administraciji zdravila. Primeri kot je administriranje/predpis zdravila v napačni enoti (mg, g).

Največkrat uporabljena zdravila, pri katerih je prišlo do presežene maksimalne doze (glej sliko 5.4)

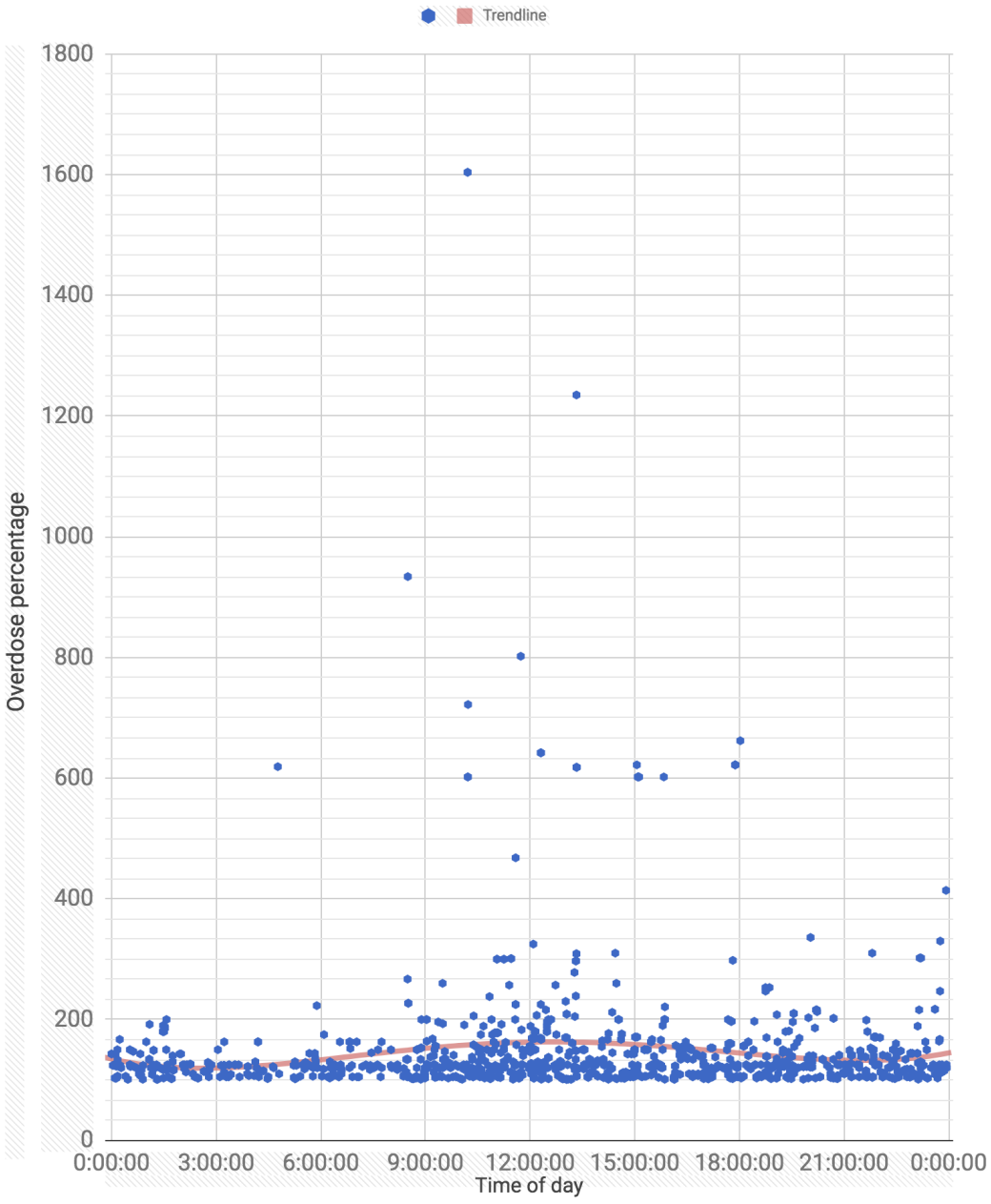
Rezultati kažejo, da so primeri kontinuiranih administracij zdravil najbolj pogosti (zdravila aplicirana v infuzijah). Opazimo tudi nekaj administracij mešanic zdravil.

Histogram procentualne pojavitve presežene maksimalne doze (glej sliko 5.5)

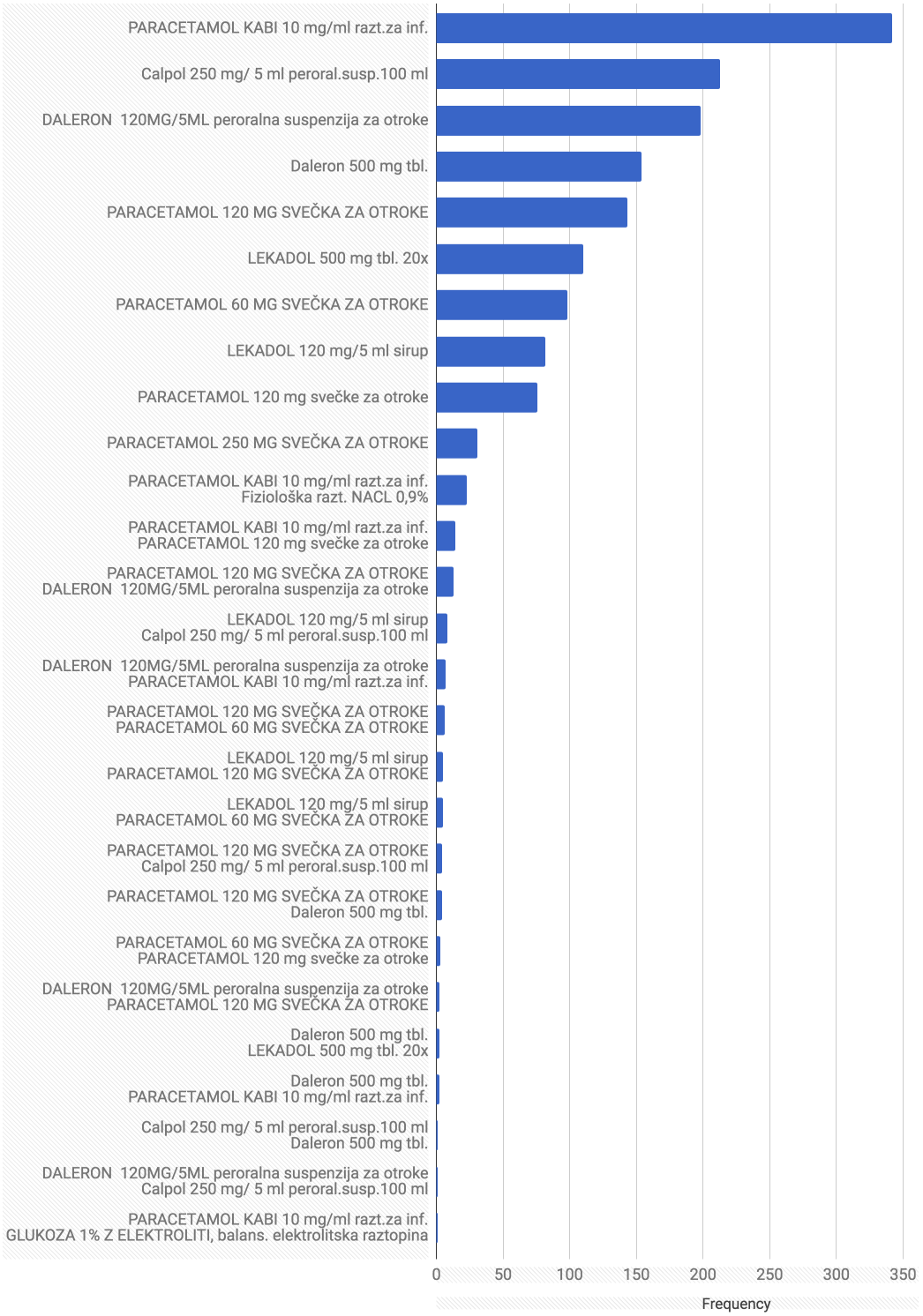
Najpogostejša je pojavitev 100% do 150% maksimalne dovoljene doze. Sledi 150% do 200%. Izjemnih vrednosti je pretežno malo.

Razmerje pravil, katerih rezultat je presegel maksimalno dozo (glej sliko 5.6)

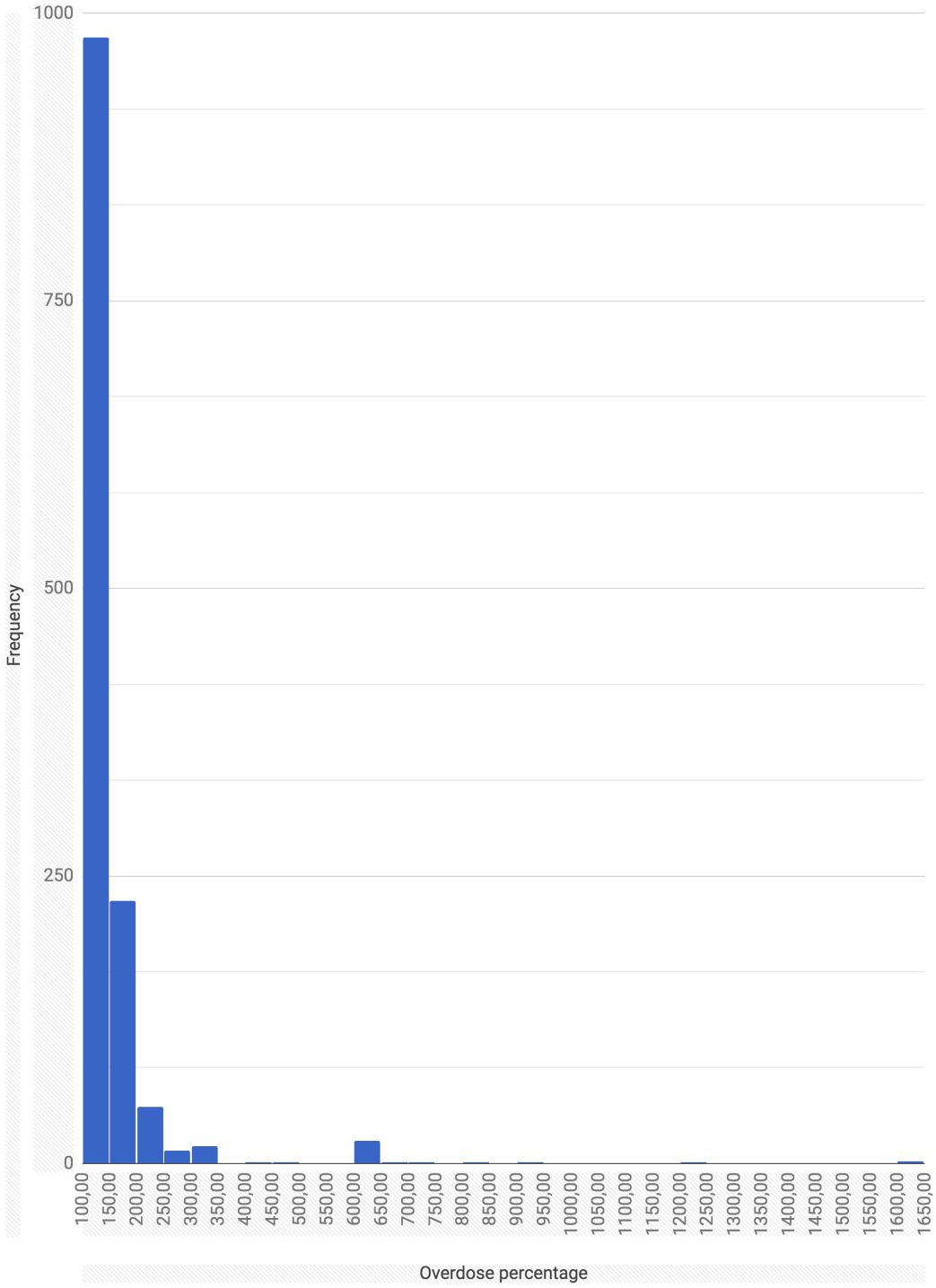
Največkrat se pojavi prekoračitev meje, ki jo določa pravilo *60/mg/kg/dan*, kar 84,1%. Sledi pravilo *2g na dan* z 11,6%.



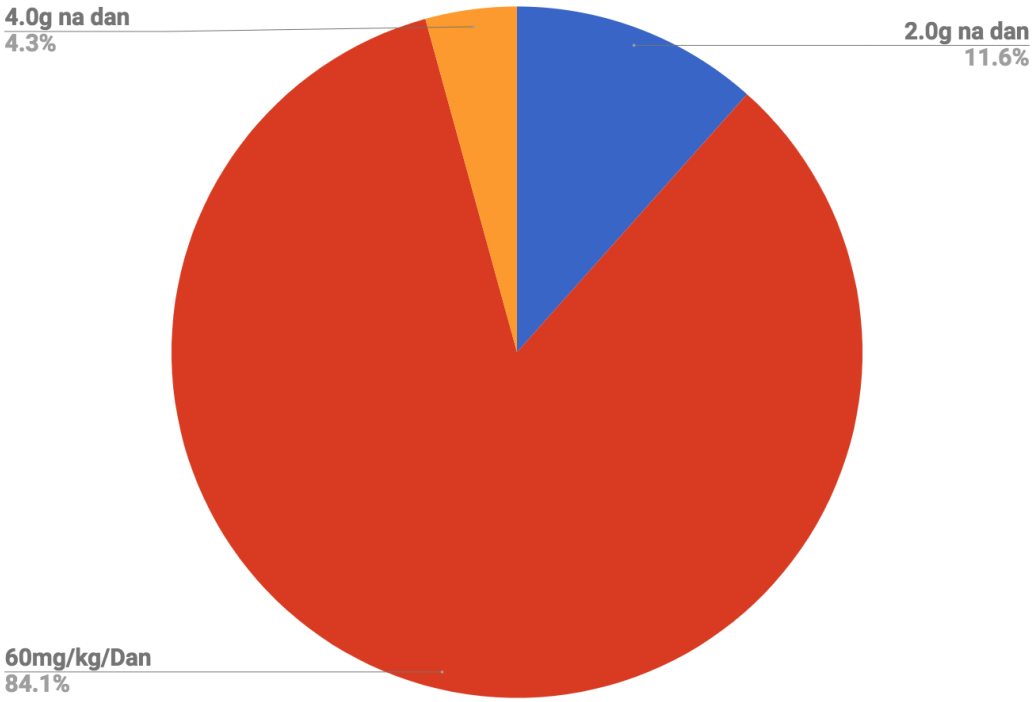
Slika 5.3: Razpršenost rezultatov o preseženi maksimalni dozi glede na čas v dnevu



Slika 5.4: Največkrat uporabljena zdravila, pri katerih je prišlo do presežene maksimalne doze



Slika 5.5: Histogram procentualne pojavitve presežene maksimalne doze



Slika 5.6: Razmerje pravil, katerih rezultat je presegel maksimalno dozo

Poglavje 6

Zaključek

6.1 Sklepne ugotovitve

V diplomskem delu sem predstavil rešitev uvedbe sistema za preračun učinkovine zdravila v aplikacijo za predpisovanje zdravil. Opisal sem nadgradnjo teh elementov, da zadostujejo kot predpogoj za implementacijo sistema. Predstavljena je bila uvedba šifranta enot, medsebojne konverzije enot in dodatnih struktur (doza za predpisovanje zdravil). Razložena je bila arhitekturna implementacija preračuna, ki omogoča enostavno nadgradnjo.

Takšen sistem lahko služi kot podpora za klinične odločitve (angl Clinical decision support) pri predpisovanju in administriranju zdravil. Sistem je enostavno nadgradljiv z dodatnimi pravili, kot so demografski podatki o pacientu. Tako lahko implementiramo preračun prekomerne aplicirane doze učinkovine in uporabnikom prikažemo opozorilo pred potrditvijo predpisa ali administracije zdravila. Opisana je implementacija takega sistema znotraj produkta Think!Meds z učinkovino paracetamol.

Analiza uporabe tega sistema dokazuje, da so takšna opozorila koristna. **2,9** odstotkov zahtevkov za preračun morebitne presežene doze paracetamola je v rezultatu vrnilo preseženo aplicirano dozo. Ugotovili smo, da je pravilo *60/mg/kg/dan* tisto, ki določa mejo, zaradi katere največkrat pride do prekoračitve. V analizi smo odkrili tudi nekaj izjem v rezultatih, ki nakazujejo

morebitne napake pri predpisovanju in apliciranju zdravil (zamenjava enote miligram in mikrogram).

Uvedba takšnega sistema za preračun izboljša varnost predpisovanja in apliciranja zdravil. Omogoča enostavno implementacijo pravil, ki se lahko uporabljajo kot določanje največje dovoljene doze učinkovin v poljubnem intervalu.

6.2 Možnost izboljšav

6.2.1 Vključitev planiranih administracij zdravil

Z upoštevanjem ne le potrjenih aplikacij vendar tudi predvidenih aplikacij lahko dosežemo preračun, ki **predvideva** količino aplicirane učinkovine v prihodnosti. To je funkcionalnost, ki ponuja zelo koristne informacije o morebitni preveliki količini aplikacije.

Recimo, da nas zanima:

- količina aplicirane učinkovine **U**,
- v intervalu dolžine **I,**
- kjer je zgornja meja količine aplicirane učinkovine **M**.

Če preračun izvajamo ob času **T** moramo izvesti preračune za vse možne intervale dolžine **I**, ki vsebuje čas **T**.

To so vsi intervali od **T - I** do **T + I**. Kot rezultat vrnemo tisti preračun, v katerem je količina aplicirane učinkovine največja.

Literatura

[1] Camunda. Dosegljivo: https://camunda.org. [Dostopano: 12. 2. 2018].

[2] Centralna baza zdravil. Dosegljivo: https://partner.zzzs.si/wps/portal/portali/aizv/e-poslovanje/centralna_baza_zdravil/. [Dostopano: 12. 2. 2018].

[3] Dictionary of medicines and devices. Dosegljivo: http://dmd.medicines.org.uk/. [Dostopano: 12. 2. 2018].

[4] Dmd medicines. Dosegljivo: http://dmd.medicines.org.uk/DesktopDefault.aspx?tabid=2. [Dostopano: 12. 2. 2018].

[5] Fdb multilex. Dosegljivo: http://www.fdbhealth.com/multilex-decision-support/. [Dostopano: 12. 2. 2018].

[6] First data bank. Dosegljivo: https://en.wikipedia.org/wiki/First_Databank. [Dostopano: 12. 2. 2018].

[7] Hibernate validator. Dosegljivo: http://hibernate.org/validator/. [Dostopano: 12. 2. 2018].

[8] Microservices. Dosegljivo: http://microservices.io/. [Dostopano: 12. 2. 2018].

[9] Snomed ct. Dosegljivo: https://www.snomed.org/snomed-ct/what-is-snomed-ct. [Dostopano: 12. 2. 2018].

[10] Spring boot. Dosegljivo: https://projects.spring.io/spring-boot/#quick-start. [Dostopano: 12. 2. 2018].

[11] Spring cloud. Dosegljivo: http://projects.spring.io/spring-cloud/. [Dostopano: 12. 2. 2018].

[12] Spring cloud netflix. Dosegljivo: https://cloud.spring.io/spring-cloud-netflix/. [Dostopano: 12. 2. 2018].

[13] Spring overview. Dosegljivo: https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/overview.html. [Dostopano: 12. 2. 2018].

[14] What is openehr. Dosegljivo: https://www.openehr.org/what_is_openehr. [Dostopano: 12. 2. 2018].

[15] Introducing openehr. Dosegljivo: http://www2.openehr.org/releases/1.0/openEHR/introducing_openEHR.pdf, 2. [Dostopano: 12. 2. 2018].

[16] Vesna Dejak. Nova arhitektura celovitega sistema za upravljanje z zdravili. Dosegljivo: http://eprints.fri.uni-lj.si/3550/1/63990172%2DVESNA_DEJAK%2DNova_arhitektura_celovitega_sistema_za_upravljanje_z_zdravili.pdf, 2016. [Dostopano: 12. 2. 2018].

[17] Kate L. Lapane, Molly E. Waring, Karen L. Schneider, Catherine Dubé, and Brian J. Quilliam. A mixed method study of the merits of e-prescribing drug alerts in primary care. *Journal of General Internal Medicine*, 23(4):442–446, Apr 2008.

[18] Marand. Think!ehr platform, marand. Dosegljivo: http://www.marand.com/thinkehr/. [Dostopano: 12. 2. 2018].

[19] Marand. Think!meds platform, marand. Dosegljivo: http://www.marand.com/thinkmeds/. [Dostopano: 12. 2. 2018].

[20] P G Nightingale, D Adu, N T Richards, and M Peters. Implementation of rules based computerised bedside prescribing and administration: intervention study. *BMJ*, 320(7237):750–753, 2000.

[21] Charles P. Schade, Frank M. Sullivan, Simon de Lusignan, and Jean Madeley. e-prescribing, efficiency, quality: Lessons from the computerization of uk family practice. *Journal of the American Medical Informatics Association*, 13(5):470–475, 2006.

[22] Jonathan M. Teich, Jerome A. Osheroff, Eric A. Pifer, Dean F. Sittig, and Robert A Jenders. Clinical decision support in electronic prescribing: Recommendations and an action planreport of the joint clinical decision support workgroup. *Journal of the American Medical Informatics Association*, 12(4):365–376, 2005.