Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Primož Delopst

Podpora pri odločanju z uporabo standarda openEHR

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Dejan Lavbič

Ljubljana, 2018

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Semantična integracija obstoječih poslovnih informacijskih sistemov je velik izziv pri razvoju programske opreme. Pri obvladovanju kompleksnosti razvoja poslovnih aplikacij obstajajo številni pristopi, kjer je ključna težava predvsem vzdrževanje obstoječih rešitev in omogočanje semantične interoperabilnosti. Omenjeni problemi veljajo tudi v zdravstvu, kjer so prisotne zelo stroge zahteve, zato je potrebno biti pri izbiri ustreznih tehnologij in pristopov še posebej previden. V okviru diplomskega dela predstavite možnost uporabe sistema za obvladovanje poslovnih pravil pri razvoju kompleksnih aplikacij. Prikažite dodano vrednost omenjenega pristopa, ki zmanjša kompleksnost sistema in omogoča podporo odločanju. Predstavite tudi spremenjene vloge v procesu razvoja, zaradi vpeljave poslovnih pravil in predlagan pristop kritično ovrednotite na izbranem primeru.

Zahvaljujem se mentorju doc. dr. Dejanu Lavbiču za mentorstvo, vodenje in pomoč pri izdelavi diplomskega dela, sodelavcem in podjetju Marand d. o. o. za pomoč in podporo v času pisanja diplomskega dela, družini ter moji Anji za pomoč in podporo v času študija.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
ADL	Archetype definition language	Arhetipski definicijski jezik
AM	Archetype model	Arhetipski model
AOM	Archetype object model	Arhetipski objektni model
API	Application Programming Interface	Programski vmesnik
AQL	Archetype query language	Arhetipski poizvedovalni jezik
CKD	Chronic kidney disease	Kronična ledvična bolezen
CKM	Clinical knowledge manager	Klinični upravljalec znanja
CRUD	Create, read, update and delete	Kreiranje, branje, posodabljanje in brisanje
CSD	Clinical decision support	Klinična podpora pri odločanju
DRL	Drools rule language	Jezik pravil Drools
EDA	Event driven architecture	Dogodkovno usmerjena arhitektura
EHR	Electronic health record	Elektronski zdravstveni zapis
EMR	Electronical medical record	Elektronski medicinski zapis
GDL	Guideline definition language	Jezik za definicijo pravil
HDF	HL7 Version 3 Development Framework	Ogrodje za razvoj HL7 verzije 3
HL7	Health Level-7	Zdravstveni nivo 7
HTTP	Hypertext Transfer Protocol	Protokol za izmenjavo hipertexta

kratica	angleško	slovensko
IHE	Integrating the Healthcare Enterprise	Integracija zdravstvenega podjetja
JDK	Java Development kit	Java razvojno orodje
JMS	Java Message System	Sporočilni sistem Java
JRE	Java Runtime Environment	Okolje za izvajanje Jave
JSON	JavaScript Object Notation	JavaScript objektna anotacija
KDIGO	Kidney Disease, Improving Global Outcomes	Bolezen ledvic, izbolšanje globalnih rezultatov
LIS	Laboratory information system	Laboratorijski informacijski sistem
MSH	Message Header	Glava sporočila
OPT	Operational template	Operativna predloga
REST	Representational state transfer	Reprezentativno stanje prenosa
RIM	Reference Information Model	Referenčni informacijski model
RM	Reference model	Referenčni model
RMI	Risk Management Incident	Tvegano upravljanje problemov
SM	Service model	Storitveni model
SQL	Structured Query Language	Strukturirani povpraševalni jezik
XML	Extensible Mark-up Language	Razširljiv označevalni jezik
XPath	XML Path Language	XML jezik poti

Povzetek

Naslov: Podpora pri odločanju z uporabo standarda openEHR

Avtor: Primož Delopst

Informatizacija in razvoj programske opreme doživljata velik vzpon. Sistemi postajajo vse kompleksnejši, posledično se veča tudi število poslovnih pravil. Slednji so težko obvladljivi in razpršeni po sistemu. Ker so napisani v programskih jezikih, jih lahko vzdržujejo le razvijalci programske opreme. Poleg vseh naštetih problemov se v nekaterih pojavlja tudi potreba po podpori pri odločanju. Nekatere od teh problemov rešujejo sistemi za upravljanje poslovnih pravil, ki jih standard openEHR uporablja kot osnovo jezika za definiranje navodil GDL. Enako se dogaja tudi v zdravstveni panogi. V diplomskem delu sem podrobno predstavil standard openEHR, nad katerim je postavljena platforma Think!EHR Platform™, standard HL7 za izmenjavo sporočil v zdravstvu in jezik GDL. Na primeru sem predstavil njegovo delovanje. Ob teoretični razlagi sem predstavil prototip sistema, katerega komponente temeljijo nad standardi in arhitekturo platforme Think!EHR Platform™ ter uporabljajo njegovo podporo pri kliničnem odločanju. Podal sem primer ugotavljanja anemije pacienta s kronično ledvično boleznijo v sistemu Think!Clinical. S predstavljeno tematiko, teoretično podlago, prototipom in primerom uporabe sem skušal predstaviti in ponuditi rešitve za nekatere od problematik pri razvoju zdravstvenih informacijskih sistemov.

Ključne besede: openEHR, GDL, podpora pri odločanju, kronična ledvična bolezen.

Abstract

Title: Decision support using openEHR standard

Author: Primož Delopst

Computerization and software development are experiencing a great rise. Systems are becoming increasingly complex, and consequently the number of business rules increases. The latter are difficult to handle because they are scattered across the system. They are written in programming languages, therefore only software developers can maintain them. In addition to all of these problems, some of them require a support in decision-making. Some of these problems are solved by business management systems that are used as implementations by the language to define GDL instructions. The same goes for the healthcare industry. The Bachelor's degree presents the openEHR standard on which the Think!EHR Platform™ platform is based on. Further, the HL7 standard for the exchange of messages in health, and the language of GDL are presented. I demonstrated its functioning with an example. Besides the theoretical basis, the prototype of the system was presented. Components of the Think!EHR Platform™ platform are based on the standards and architecture of the platform and use its support in clinical decision-making. It supports the case of finding an anaemia of a patient with chronic kidney disease in the Think!Clinical system. With the presented topic, theoretical basis, prototype, and a case study I attempted to present and offer solutions to some of the problems in the development of health information systems.

Keywords: openEHR, GDL, decision support, chronic kidney disease.

Poglavje 1

Uvod

1.1 Motivacija

Razvoj programske opreme doživlja v zadnjem času velik vzpon, prav tako tudi razvoj programske opreme v zdravstveni panogi. Posledično se zdravstvo vedno bolj informatizira. Na nivoju posamezne bolnišnice obstajajo številne aplikacije [38]. Vsaka od teh ima podatkovno bazo in operira s svojimi podatki. Ta problem rešuje standard openEHR. Na podlagi le-tega je podjetje Marand d. o. o. razvilo platformo Think!EHR Platform™, ki omogoča, da so lahko aplikacije zgrajene nad skupno arhitekturo, kar posledično pomeni, da so podatki prenosljivi med temi aplikacijami. Z razvojem vedno večjih in bolj kompleksnih sistemov se posledično povečuje kompleksnost programske kode. Pojavlja se vedno več poslovnih pravil, ki še dodatno povečujejo njeno kompleksnost. Takšna koda je težka za vzdrževanje. Vzdržujejo jo lahko le razvijalci programske opreme. Zaradi tega je težje razvijati dodatno funkcionalnost, podjetja tako razvito programsko opremo težje prodajo. Vedno večja je tudi potreba po podpori pri odločanju. Za reševanje nekaterih od teh problemov so bili razviti številni sistemi za upravljanje s poslovnimi pravili, ki jih uporabljajo že v številnih industrijskih panogah. So tudi ena od osnov za implementacijo modula za izvajanje navodil GDL, ki temelji na standardu openEHR.

1.2 Cilji

Z razvojem zdravstvenih informacijskih sistemov narašča kompleksnost kode in število poslovnih pravil. Vedno večja je potreba po klinični podpori pri odločanju. Ker diplomsko delo temelji na standardu openEHR, se najprej osredotoča na njegovo predstavitev in poda teoretično podlago za standard HL7, s katerim zagotovimo konsistentnost komunikacije med različnimi zdravstvenimi sistemi. V literaturi in praksi se velikokrat zasledi pojem "Sistem za upravljanje poslovnih pravil", ki prispeva k zmanjševanju kompleksnosti sistema ter odstranitve množice pogojev iz programske kode, služi pa tudi za podporo pri odločanju [7]. Sistem za upravljanje poslovnih pravil analitikom omogoča določanje pravil, kar lahko izboljša kakovost informacijskega sistema ter obremenjenost razvijalcev. Cilj diplomskega dela je na konkretnem primeru pokazati, kako zmanjšati kompleksnost programske kode ter izboljšati njeno kakovost, zato sem se v diplomskem delu osredotočil na opis jezika za definicijo navodil GDL, ki temelji na standardu openEHR. Za svojo implementacijo lahko uporabi enega od različnih sistemov za upravljanje s poslovnimi pravili. Njegovo uporabo sem nazorno prikazal na primeru anemije pacienta s kronično ledvično boleznijo. Predstavil sem tudi prototip na podlagi standarda openEHR, ki navodila uporablja. Na koncu sem v diplomskem delu predstavil prototip rešitve z uporabo arhitekture dogodkov, protokola REST, standardov openEHR in HL7, informacijskega sistema in platforme Think!EHR Platform™ in na primeru uporabe ugotavljanja anemije v kliničnem informacijskem sistemu Think!Clinical nazorno prikazal njegovo delovanje.

1.3 Struktura diplomskega dela

Diplomsko delo je razdeljeno na štiri dele. V drugem poglavju so predstavljeni standardi openEHR, jezik GDL in HL7, sledi predstavitev uporabljenih tehnologij in predstavitev kronične ledvične bolezeni. Predstavljena je Think!EHR Platform™ platforma podjetja Marand d. o. o., njeno proženje

dogodkov in izvajanje navodil, sistem za upravljanje poslovnih pravil, sistem Apache Kafka in kliničnI informacijskI sistem Think!Clinical. Diplomsko delo nato predstavi način kreiranja novega GDL navodila na primeru ugotavljanja anemičnosti pacienta s kronično ledvično boleznijo. Na koncu predstavi prototip, ki je v grobem razdeljen na dva dela. V prvem delu so predstavljene posamezne komponente prototipa, v drugem pa njegovo delovanje. V zadnjem delu je delovanje prototipa predstavljeno na primeru ugotavljanja anemije pacienta s kronično ledvično boleznijo. Podane so prednosti in slabosti uporabe prototipa. V zaključku so predstavljene ugotovitve diplomskega dela in možnosti za nadaljnje delo.

Poglavje 2

Opis standardov za zapis in prenos zdravstvenih podatkov

2.1 Standard OpenEHR za zapis zdravstvenih podatkov

2.1.1 Elektronski zdravstveni zapis

Elektronski zdravstveni zapis (angl. Electronic health record) ali EHR je elektronski zdravstveni zapis pacienta [36]. Je uradni zdravstveni dokument posameznika, ki je deljen med različnimi zdravstvenimi ustanovami. Dokument vsebuje klinične podatke, kot so: obiski zdravstvenih ustanov, alergije, zgodovina bolezni, zdravila, informacije o boleznih, vitalne znake, laboratorijske preiskave, slike radioloških preiskav ... Vsebuje lahko tudi nekatere druge podatke, kot so številka zdravstvenega zavarovanja, demografske podatke in celo podatke iz nekaterih naprav. Glavne prednosti elektronskega zdravstvenega zapisa so možnost souporabe in posodabljanje zdravstvenih zapisov med različnimi ustanovami, boljše shranjevanje in pridobivanje, lažje standardiziranje storitev, možnost odločitvene podpore za zdravstveno osebje in potencialno tudi znižanje stroškov razvoja zdravstvenih sistemov.

2.1.2 Razlika med EHR in EMR

V nekaj zadnjih letih je zdravstvena industrija z informatizacijo doživela korenite spremembe. Posledično se je zgodil prehod iz papirja na digitalni zapis zdravstvenih podatkov. V literaturi in novicah se je začelo pojavljati veliko zapisov o elektronskih medicinskih zapisih in elektronskih zdravstvenih zapisih [15]. Termina se pogosto pomensko zamenjujeta, vendar med njima obstajajo pomembne razlike.

Elektronski medicinski zapis ali EMR predstavlja digitalno nadomestilo papirnatih zapisov v zdravstvu [16]. Vsebuje medicinske podatke ter podatke o zgodovini zdravljenja pacientov. Zapis informacij in njihovo shranjevanje omogoča lažje modeliranje zdravja populacije, sledenje zdravstvenim težavam in njihovo iskanje. Deljenje oz. prenos podatkov je otežen, saj ne omogoča deljenja zunaj ordinacije drugače kot s tiskanjem na papir.

Elektronski zdravstveni zapis ali EHR omogoča več funkcionalnosti kot elektronski medicinski zapis, saj se osredotoča na pacientovo celotno zdravje. Omogoča souporabo informacij (informacije so dostopne zunaj ordinacije) in posodabljanje v realnem času [15, 16]. Omogoča več orodij, ki zdravstvenim delavcem pomagajo pri sprejemanju odločitev.

2.1.3 Standard openEHR

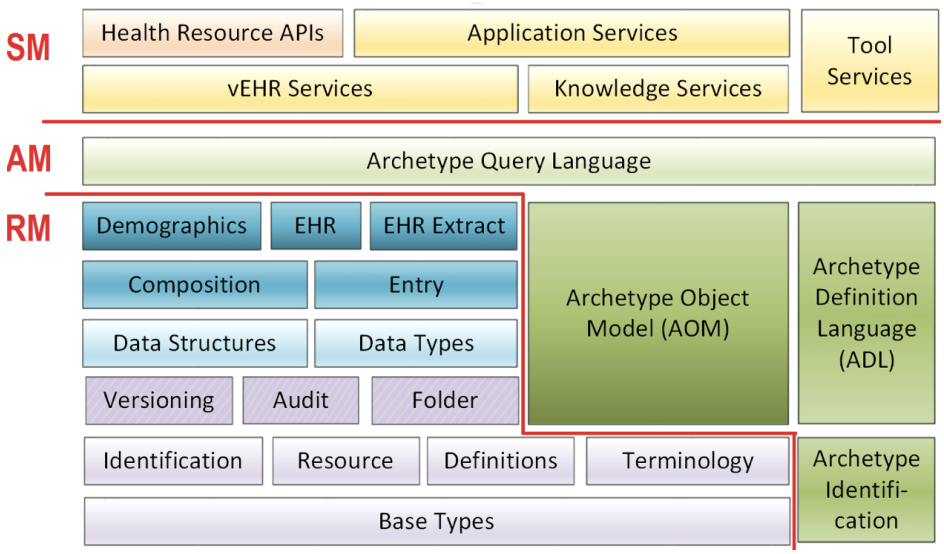
Fundacija openEHR je neprofitno podjetje s sedežem v Združenem kraljestvu na Univerzi v Londonu, ki stremi k napredku in zagotavljanju elektronskega zdravstvenega zapisa visoke kakovosti [27, 29, 41]. Njen namen je prenos zdravstvenih podatkov iz fizične oblike v elektronsko in zagotavljanje interoperabilnosti med vsemi oblikami elektronskih zdravstvenih podatkov. Je skupnost več kot 1200 ljudi, ki se ukvarjajo z vzdrževanjem specifikacije openEHR, odprto kodno implementacijo ter razvojem več orodij za modeliranje (npr. ADL designer). Svoje delo objavlja pod odprto kodno licenco OpenSource in s tem nadaljuje tradicijo projekta GEHR, iz katerega je nastala [41]. Temelji na podlagi standardov GEHR in CEN [9, 29, 43]. Podjetje sta usta-

novila University College London (UCL) in podjetje Ocean Informatics na podlagi objavljenega članka Davida Ingrama o potrebi po klinično fokusirani organizaciji [29].

OpenEHR specifikacija je odprt standard, ki opisuje upravljanje, shranjevanje, pridobivanje interoperabilnih zdravstvenih podatkov, neodvisnih od organizacije in tehnologije v obliki elektronskega zdravstvenega zapisa, ki so bili zapisani s strani fundacije openEHR [29, 44]. Standard temelji na elektronskem zdravstvenem zapisu EHR. Identifikator EHR pacienta se imenuje EhrId. Ena izmed njegovih vlog je vloga ključa za povezavo pacientovih podatkov v drugih sistemih. Po standardu openEHR so demografski podatki pacienta ločeni od ostalih podatkov, saj to omogoča boljšo varnost in težjo sledljivost. Arhitektura EHR zapisa je predstavljena na sliki 2.1 in je sestavljena iz več delov. Glavni razred je razred EHR. Ta je unikaten na podlagi EHR identifikatorja. Komponenta dostopa (angl. Access) vsebuje zapise o pravicah dostopa do podatkov o pacientu. Komponenta status (angl. Status) vsebuje podatke o zadnjih zapisih, stanju, eksternem identifikatorju itd. Komponenta direktorija je opcijska komponenta, ki se uporablja za logično urejevanje podatkovnih zapisov. Zapis EHR je sestavljen iz več podatkovnih zapisov. Vsak podatkovni zapis vsebuje identifikator sistema, ki ga je kreiral. Vsi, ki prispevajo ali spreminjajo podatkovni zdravstveni zapis pacienta, so zapisani v razredu CONTRIBUTION.

Arhitektura openEHR specifikacije temelji na večnivojski arhitekturi in je sestavljena iz štirih glavnih delov. Prikazana je na sliki 2.1:

- Referenčni model (RM), ki vsebuje osnovne razrede openEHR sistema.
- Arhetipski model (AM).
- Arhetipski poizvedovalni jezik (AQL), ki je zasnovan za iskanje in pridobivanje kliničnih podatkov v elektronskem zdravstvenem zapisu.
- Storitveni model (SM), ki vsebuje definicije osnovnih storitev.



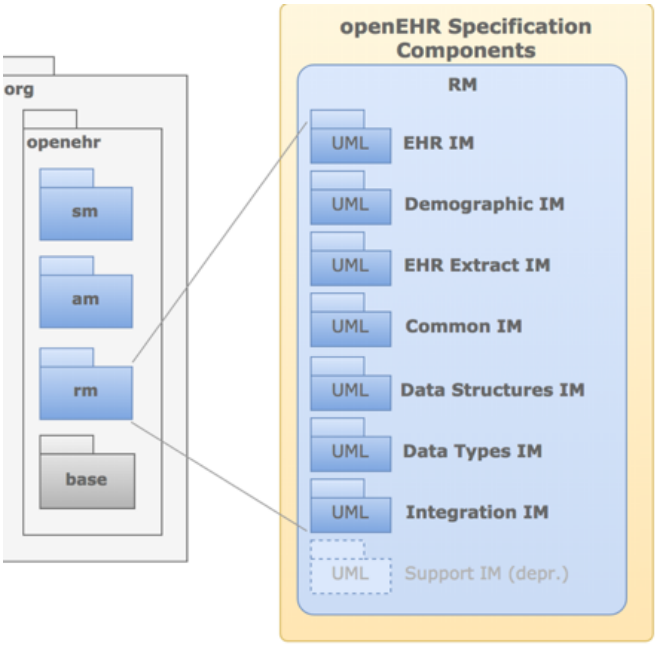
Slika 2.1: Bločni diagram strukture openEHR komponent

Standard openEHR ločuje med vsebinskimi in jezikovnimi pravili. Temu večkrat pravijo dvonivojsko modeliranje. Ena od ključnih prednosti standarda je odprtost, kar omogoča hitro razvijanje najrazličnejših arhetipov in predlog, s katerimi so predstavljeni zdravstveni podatki. Razvite strukture so prosto dostopne. Večina jih je mednarodno potrjenih s strani zdravstvenih organizacij. Uporabnikom je omogočeno kreiranje, deljenje, razpravljanje in potrjevanje le-teh v skupnem repozitoriju, imenovanem upravljavec kliničnega znanja CKM (angl. Clinical Knowledge Manager) [10].

2.1.4 Referenčni model

Referenčni model definira podatkovne razrede, logične povezave med njimi, osnovne podatkovne tipe, dostope in verzioniranje [28]. Opisuje demografski model. Med pomembnejše podatkovne tipe spadajo tipi za zapis tekstovnih podatkov (DV_TEXT in DV_CODED_TEXT), zapisi logičnih vrednosti (DV_BOOLEAN), zapisi količinskih podatkov (DV_DURATION, DV_QUANTITY, DV_COUNT) in zapisi časovnih podatkov (DV_DATE, DV_TIME, DV_DATETIME). Poleg zgoraj opisanih razredov EHR in CONTRIBUTION

je med pomembnejšimi razred COMPOSITION. V referenčnem modelu so opisani štirje glavni tipi razredov arhetipov. To so COMPOSITION, SECTION, ENTRY in CLUSTER. Vsak je uporabljen za predstavitev različne klinične vsebine in kliničnih procesov. Referenčni model specifikacije openEHR predvideva, da je razred COMPOSITION predstavljen kot neke vrste dokument, ki vsebuje več organiziranih razredov SECTION. Razredi SECTION nato vsebujejo več razredov tipa ENTRY. V razredu COMPOSITION je shranjenih največ podatkov. Predstavlja eno enoto, ki se shrani v openEHR sistemu [23]. Vsebuje najpogostejše klinične dogodke in dokumente, kot so: obisk, rezultati testov, radiologija, operacija, predpis zdravil itd. Razred lahko vsebuje več sekcij. Vsaka vsebuje del klinične vsebine. Sekcije so predstavljene z razredom SECTION. Ne nosijo pomembne vsebine. Najbolj nazoren primer so vitalni znaki. Služijo kot vsebniki za manjše razrede, ki nosijo več klinične vsebine, kot sta ENTRY in CLUSTER. Razred ENTRY predstavlja informacije, ki so grupirane med seboj. Uporabimo jih lahko samostojno. Primera le-teh sta krvni pritisk ali diagnoza. Razred ENTRY je abstrakten razred. Delimo ga na štiri razrede. Razred OBSERVATION vsebuje neobdelane informacije. Te lahko vsebujejo poročila pacienta, kot so simptomi in rezultati preiskav. Delimo ga na štiri dele: DATA, STATE, PROTOCOL in HISTORY. Primeri razreda so teža, višina, krvni sladkor itd. Razred EVALUATION se uporablja za klinično obdelane informacije. Primer je diagnoza. Razred INSTRUCTION nosi podatke o tem, kar naj bi se zgodilo v prihodnosti. Primer je naročilo. Razreda ACTION in INSTRUCTION se pogosto uporabljata skupaj. Akcije dopolnjujejo navodila ter zapišejo rezultat navodila (npr. končano). Razred ACTION vsebuje informacije o tem, kar se je dejansko zgodilo. Razred CLUSTER se uporablja v razredu ENTRY ali drugih razredih CLUSTER. Uporablja se v primerih, ko je potrebna rekurzivnost.



Slika 2.2: Referenčni model openEHR specifikacije

2.1.5 Arhetipi

Množica arhetipov temelji na referenčnem modelu v obliki pravil in omejitev [6]. Določa pravila med njegovimi podatkovnimi strukturami. Formalno gledano je vsak arhetip specifikacija za specifičen klinični koncept. Vsebuje vse elemente, ki imajo smisel zanj. Na ta način lahko pokrivajo splošno klinično vsebino. Zgrajeni so tako, da omogočajo deljenje in uporabo v različnih situacijah, kliničnih vsebinah, jezikih in EHR sistemih. To pomeni, da bo njihov pomen ostal enak. Napisani so v jeziku ADL (angl. Archetrype definition language). Njihovo strukturo opisuje AOM (angl. Archetype object model). Ta struktura je neodvisna od sistema. Njen glavni namen je določiti, kako naj razvijalci razvijajo orodja za arhetipe. Glede na referenčni model so predstavljeni s štirimi razredi, ki so že predstavljeni v poglavju 2.1.4. Vsak arhetip je sestavljen iz identifikatorja, jezika, specializacije, opisa, definicije, pravila, terminologije, zaznamkov in zgodovine sprememb. V jeziku je določen jezik,

v katerem je arhetip napisan in vsi njegovi prevodi. V opisu so opisani arhetipi in vse njegove uporabe. Arhetipi uporabljajo različno terminologijo, ki je definirana v delu njegove terminologije. V definiciji so zapisane podatkovne strukture in podatkovni tipi. V zgodovini sprememb se hranijo vse spremembe, ki so bile izvedene nad arhetipom. Zapisana so pravila, ki naj veljajo za posamezen arhetip (npr. minimalne vrednosti). Z njihovo pomočjo lahko izvajamo validacijo. Arhetipi naj bi bili zgrajeni na stabilen način, ki omogoča dolgoročno uporabo brez potrebe po popravljanju. Zaradi tega je zaželeno, da so potrjeni s strani zdravstvenih strokovnjakov in organizacij. Dostopni so na upravljavcu kliničnega znanja CKM. Služijo kot osnova za arhetipski poizvedovalni jezik.

2.1.6 Predloge

Predloga je struktura, ki je sestavljena iz več arhetipov. Arhetipe lahko dodajamo v za to namenjene reže [37]. Ker je arhetip namenjen splošnemu primeru uporabe, ni potrebe, da predloga vsebuje njegovo celotno množico podatkov. Iz posameznega arhetipa lahko pri gradnji predloge uporabimo samo podmnožico podatkov, potrebnih pri tem primeru. Spremenjeni so lahko podatkovni tipi posameznega arhetipa. Določimo mu tip podatka, številnost, vrednost, meje in terminologijo. Predlogo zgradimo v več korakih. Najprej si moramo izbrati korenski arhetip. S pomočjo rež določimo arhetipe. S tem dobimo strukturo predloge. Iz posameznega arhetipa nato izberemo podmnožico podatkov, ki jih potrebujemo. To storimo tako, da nepotrebnim podatkom nastavimo pojavitev na 0..0 (podatek se v tem primeru ne bo pojavil). Izbrani podmnožici podatkov lahko specificiramo še mejo, številnost, obveznost ... S tem je konstruiranje nove predloge končano. Predloge so zapisane v datotekah s končnico OPT (angl. Operative Template) v jeziku XML. Za izgradnjo arhetipov in predlog obstaja nekaj urejevalnikov. Na ta način so lahko predloge zgrajene hitreje in lažje, razvijajo jih lahko analitiki s pomočjo zdravstvenih strokovnjakov. Za razliko od arhetipov, ki so zgrajeni za uporabo na nacionalni ravni, so predloge najpogosteje zgrajene za uporabo

v posameznih podjetjih.

2.1.7 Arhetipski poizvedovalni jezik - AQL

Poizvedovalni jezik je računalniški jezik, ki se uporablja za poizvedovanje po podatkovnih bazah in informacijskih sistemih. Obstaja množica različnih poizvedovalnih jezikov, kot so SQL, XQuery ali objektno usmerjeni poizvedovalni jeziki. Problem se pojavi, ker so odvisni od podatkovne strukture sistema, kar pomeni, da se isti stavek ne more uporabiti v drugih sistemih z različno podatkovno strukturo, zato noben od naštetih ni primeren za poizvedovanje po EHR-ju.

Arhetipski poizvedovalni jezik (AQL) je poizvedovalni jezik, zasnovan za iskanje in pridobivanje kliničnih podatkov v elektronskem zdravstvenem zapisu in temelji na specifikacijah openEHR [5]. Za razliko od ostalih poizvedovalnih jezikov, ki temeljijo na nivoju instance, AQL temelji na poizvedovanju na nivoju arhetipov (nivo semantike), kar mu omogoča prenosljivost poizvedb med različnimi ponudniki EHR. Ima veliko podobnosti z ostalimi poizvedovalnimi jeziki, kot so: aritmetične operacije (seštevanje, odštevanje, deljenje, množenje), relacijske operacije (=, !=, <, <=, >, >=), logične operacije(AND, OR, XOR, NOTA), agregacijske funkcije (MAX, MIN, AVG), paginacijo, vrstni red vračanja in nekatere funkcije, ki jih najdemo v ostalih jezikih. Jezik temelji na dveh poizvedovalnih jezikih: SQL-u in XPath. Po jeziku SQL je prevzel nekaj ključnih stavkov: SELECT, FROM, WHERE, ORDER BY itd. Za naslavljanje podatkovnega elementa znotraj podatkovnega zapisa se pri SQL stavkih uporablja ime tabele in stolpca. Pri jeziku AQL se namesto tega za naziv posameznega podatkovnega elementa znotraj podatkovnega zapisa uporablja XPath. S potjo lahko določimo arhetip, element ali vrednost, ki jo želimo vrniti.

Stavek SELECT določa množico podatkov, ki jih bo poizvedba vrnila. Stavek FROM določa vir rezultata (npr. EHR ali pa Composition) ter pripadajoči pogoj vsebovanosti (EHR[ehr_id=$ehrId] CONTAINS COMPOSITION c[openEHR-EHR COMPOSITION.encounter.v1]). S stavkom WHER-

E določimo pogoje iskanja. S stavkom ORDER BY uredimo podatke, ki bodo vrnjeni v rezultatu. Jezik ima nekatere posebnosti, kot je na primer stavek CONTAINS. Pozna tri vrste predikatov: standardni, vozliščni in arhetipski. Standardni predikat ima vedno levi operand, desni operand in operator ([ehr_id/value='123456']). Levi del predikata je po navadi openEHR pot, desni del pa vrednost. Za operator so lahko uporabljene le relacijske operacije. Arhetipski predikat lahko vsebuje le levi del predikata ([openEHR-EHR-COMPOSITION.encounter.v1]). Uporabljen je lahko le v WHERE stavku za določanje obsega podatkovnih virov, iz katerih se lahko vrnejo podatki. Vozliščni predikat ima lahko naslednje oblike:

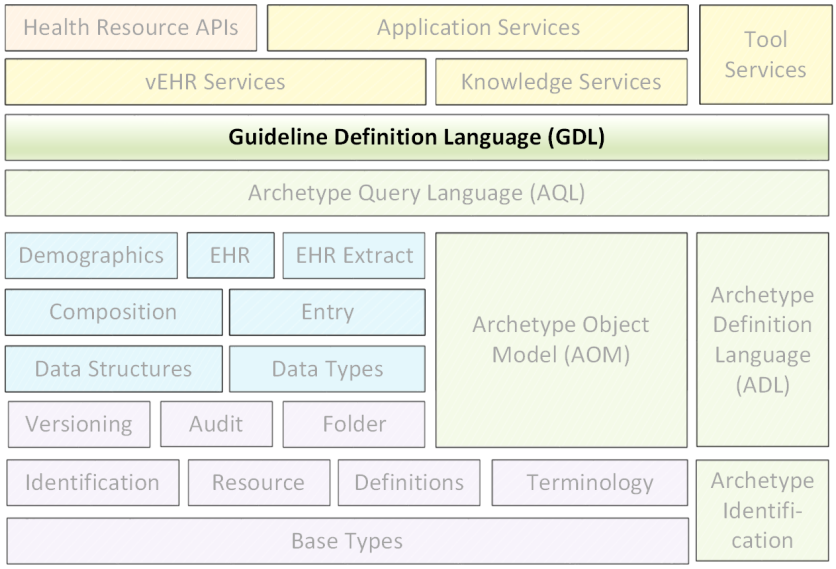
- Vsebuje lahko le identifikator vozlišča arhetipa ([at0057]).
- Vsebuje lahko identifikator vozlišča arhetipa in standardni predikat ([at0057 and name/value='value']).
- Vsebuje lahko identifikator vozlišča arhetipa in bližnjico standardnega predikata ([at0057, 'value']).

```
select
    distinct obs/data[at0001]/events[at0002]/data[at0003]/items[
        at0095]/items[at0096 and name/value='S-Feritin']/items[
        at0078]/value/magnitude,
    obs/data[at0001]/events[at0002]/data[at0003]/items[at0095]/
        items[at0096 and name/value='S-Feritin']/items[at0078]/
        value/normal_range/lower_unbounded,
    obs/data[at0001]/events[at0002]/data[at0003]/items[at0095]/
        items[at0096 and name/value='S-Feritin']/items[at0078]/
        value/normal_range/upper_unbounded,
    obs/data[at0001]/events[at0002]/data[at0003]/items[at0095]/
        items[at0096 and name/value='S-Feritin']/items[at0078]/
        value/normal_range/lower/magnitude,
    obs/data[at0001]/events[at0002]/data[at0003]/items[at0095]/
        items[at0096 and name/value='S-Feritin']/items[at0078]/
        value/normal_range/upper/magnitude
from EHR [ehr_id/value='7d91a75f-56cv-d4g5-g28g-b4agba8899g4']
contains COMPOSITION c[openEHR-EHR-COMPOSITION.report-mnd.v1]
contains OBSERVATION obs[openEHR-EHR-OBSERVATION.lab_test.v1]
where
    c/name/value='Laboratory_report' and
    obs/data[at0001]/events[at0002]/data[at0003]/items[at0095]/
        items[at0096]/name/value='S-Feritin'
order by obs/data[at0001]/events[at0002]/time/value desc
limit 1
```

Slika 2.3: AQL stavek vrača magnitudo, spodnjo in zgornjo mejo ter podatek o tem, ali zgornja in spodnja meja sploh obstajata, zadnjega shranjenega podatkovnega zapisa, ki v vozlišču at0096 vsebuje ime S-Feritin. Iz poizvedbe so vidne vse tri vrste predikatov.

2.1.8 Jezik za definiranje pravil GDL

Jezik za definiranje pravil (angl. Guidline Definition Language) je formalizem, ki na podlagi definiranega jezika omogoča podporo pri odločanju [17]. Opisan je v standardih openEHR. Nastal je zaradi potrebe po izvajanju klinične podpore pri odločanju na podlagi različnih programskih jezikov, tehnologij in sistemov. Njegov namen je izražanje klinične logike na podlagi pravil. Navodila lahko izboljšajo klinične podatke, zagotovijo varnost pacienta, zmajnšajo stroške in povečajo zaupanje v zdravstvo [39]. OpenEHR arhetipi so uporabljeni za vhodne in izhodne podatke. Jezik mora znati definirati klinično logiko za odločanje z uporabo le-teh. To mu daje neodvisnost od naravnih jezikov in terminologije. Omogočati mora urejanje metapodatkov navodila (npr. avtorstvo, namen, reference, verzija) neodvisno od naravnega jezika. Poimenovanje posameznega pravila mora biti edinstveno. Verzioniranje pravil, izvajanje več pravil za klinično odločanje zapovrstjo, omogoča izvajanje preproste in kompleksne logike, uporabo navodil v različnih aplikacijah in sistemih. Prevedemo ga lahko v več jezikov brez spremembe logične definicije navodila. Neodvisnost od terminologije je zagotovljena na podlagi tega, da je možno povezati lokalno definiran izraz v navodilu s konceptom, ki je definiran s strani enega ali več sistemov ali pa zunanjo referenco terminologije. Strukturiran je iz dveh modulov: modula navodil (angl. Guide modul) in modula izražanja (angl. Expression mode) [17]. Oba sta sestavljena iz več razredov, ki so predstavljeni v poglavjih 2.1.8.1 in 2.1.8.2.



Slika 2.4: Umestitev GDL v arhitekturo openEHR

2.1.8.1 Modul navodil

Modul navodil je sestavljen iz več razredov in njihovih atributov, ki so prikazani na sliki 2.5).

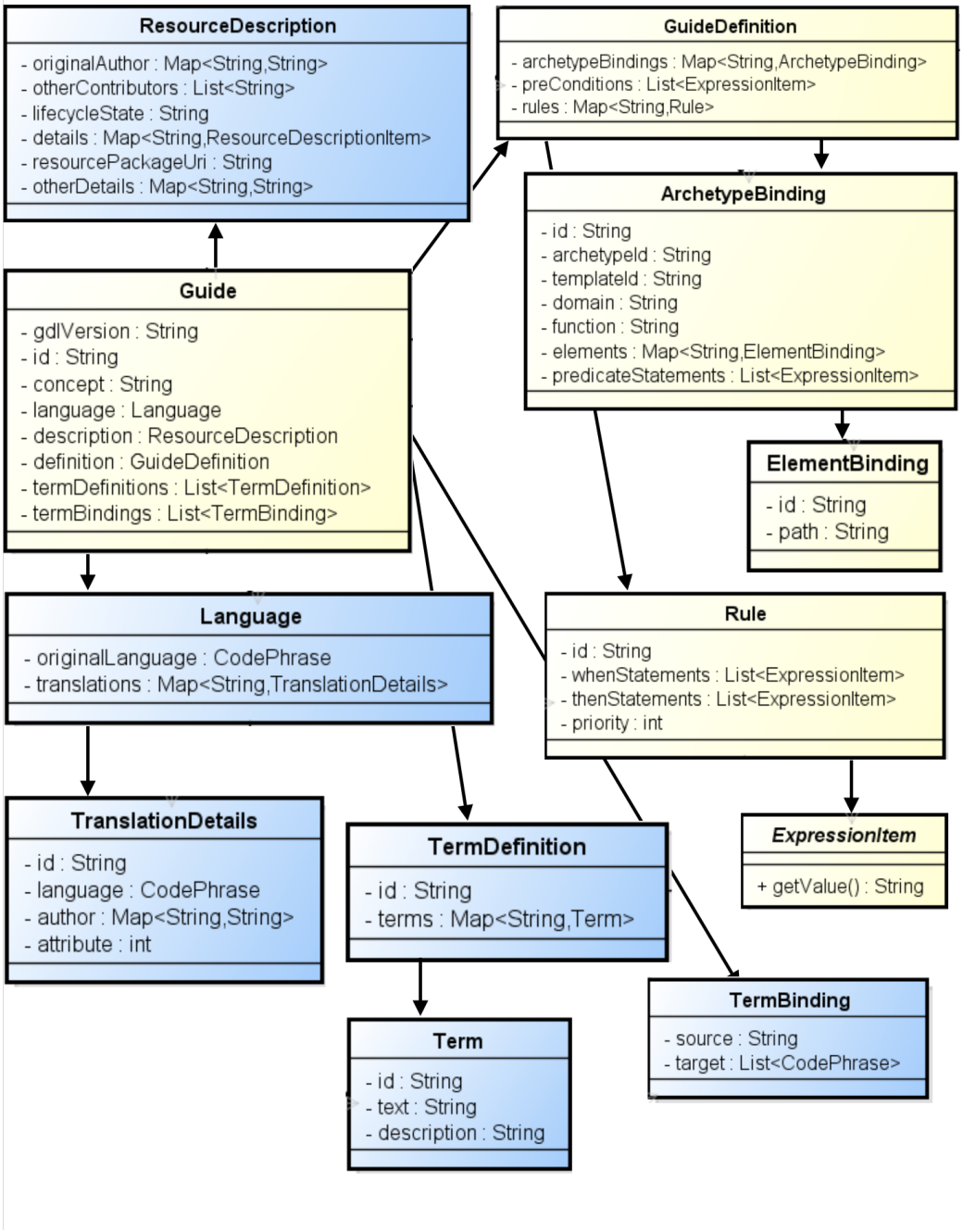
- GUIDE je razred navodila. Definira verzijo GDL, v katerem je navodilo napisano, njegov enolični identifikator, pomen, opis in definicijo. Opis vsebuje podatke o avtorju, uporabi, povezavah in naravnem jeziku navodila. Definicija vsebuje podatke o definiciji pravil, povezavah arhetipov in podatke o ontologiji navodila.

- GUIDE_DEFINITION je razred definicij. Vsebuje listo povezav arhetipov, mapo pravil in listo predpogojev. Lista arhetipov definira elemente, ki bodo v pravilu uporabljeni. Mapa pravil vsebuje pravila, ki so indeksirana po GT kodi. Lista predpogojev vsebuje predpogoje, ki se morajo izvesti pred izvedbo navodila.

- ARCHETYPE_BINDING definira listo povezav elementov med arhetipom ali predlogo in lokalnimi GT kodami. Vsebuje identifikator arhetipa in opcijsko tudi identifikator predloge, iz katere so vzeti elementi in domena, ki definira vir spremenljivke (EHR pomeni, da je vrednost spremenljivke dobljena iz EHR, CSD pomeni, da je spremenljivka vstavljena v sistem za izvajanje), mapo elementov (za ključ uporabljajo lokalno GT kodo) in listo predikatov, ki se morajo izvesti, preden se bodo lahko izvedle EHR poizvedbe.

- ELEMENT_BINDING definira povezavo med arhetipom in lokalno spremenljivko v pravilu. Vsebuje lokalno GT kodo elementa in pot do elementov v arhetipu.

- RULE predstavlja pravilo v navodilu. Sestavljen je iz lokalne GT kode, seznama izrazov, ki morajo biti izračunani, preden se to pravilo lahko izvede in seznama izrazov, s katerimi je sestavljen rezultat pravila.

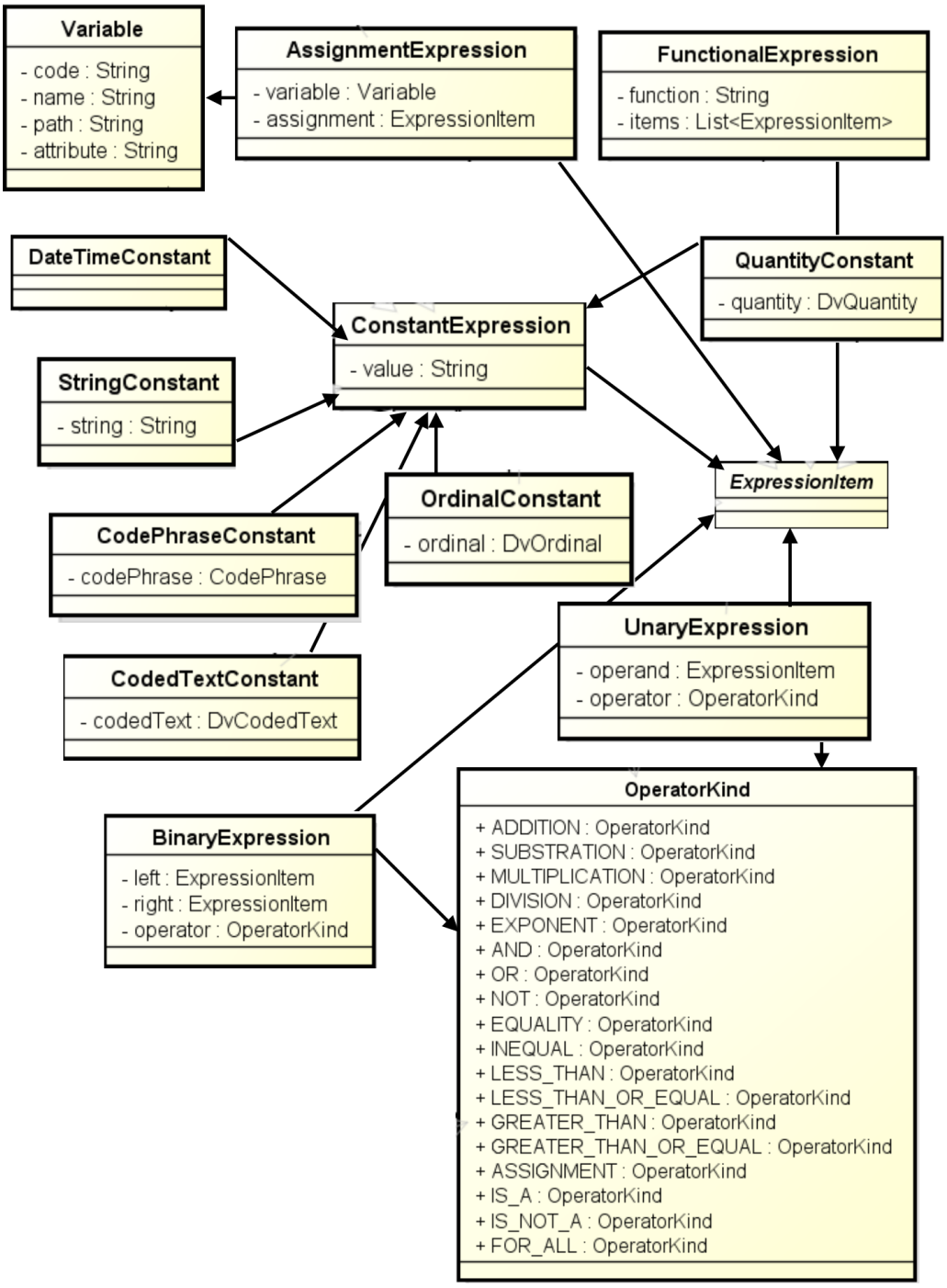


Slika 2.5: Grafični prikaz razredov modula navodil

2.1.8.2 Modul izrazov

Modul izrazov je predstavljen iz več razredov in atributov, ki so predstavljeni na sliki 2.6).

- EXPRESSION_ITEM je abstraktna tabela izraza. Ne vsebuje nobenega atributa. UNARY_EXPRESSION, BINARY_EXPRESSION, ASSIGMENT_EXPRESSION, FUNCITIONAL_EXPRESSION in CONSTANT_EXPRESSION dedujejo od nje.
- UNARY_EXPRESSION vsebuje operand tipa EXPRESSION_ITEM in operator tipa OPERTOR_KIND izraza.
- BINARY_EXPRESSION vsebuje levi operand in desni operand, ki sta tipa EXPRESSION_ITEM in operator tipa OPERATOR_KIND.
- ASSIGMENT_EXPRESSION vsebuje atributa variable in assignment. Prvi predstavlja GT kodo spremenljivke, ki se ji bo vrednost priredila. Druga je tipa EXPRESSION_ITEM in predstavlja izraz, iz katerega je vrednost pridobljena.
- FUNCTIONAL_EXPRESSION vsebuje podatek o tem, kakšen tip funkcije se bo izvedel in listo elementov funkcije tipa EXPRESSION_ITEM.
- OPERATOR_KIND vsebuje tip operatorja, njegovo ime in simbol. Definirane so naslednje vrste operatorjev: logični, aritmetični, relacijski in terminološki.
- FUNCTION_KIND vsebuje tip funkcije in funkcijo, ki je definirana kot parameter, ki jo opiše.



Slika 2.6: Grafični prikaz modula izrazov

2.1.8.3 Implementacija GDL

Jezik je neodvisen od implementacijske tehnologije. Zato ga lahko implementiramo z različnimi platformami za upravljanje poslovnih pravil. Standard v specifikacijah navaja, da je za prvo implementacijo modula za izvajanje GDL izbral platformo Drools [13]. Predstavljena je v poglavju 3.8.1. Sistem omogoča pretvorbo jezika GDL v jezik DRL (angl. Drools rule language). Za implementacijo z izbrano platformo so navedene naslednje predpostavke:

- Vsako pravilo v jeziku GDL kreira pravilo v jeziku DRL.
- Ime vsakega pravila je sestavljeno iz identifikatorja navodila in GT kodo.
- Vsako pravilo bo imelo atribute brez zanke. S tem onemogočimo izvajanje teh v pravilu.
- Trenutni čas je definiran z uporabo razreda DV_DATETIME.
- Definicija vsakega elementa v izrazu bo preverjena zaradi izvedbe.
- Vsaka modifikacija elementa bo izvedena znotraj metode modify z namenom širjenja baze znanja.
- Elementov ne moremo definirati zunaj obsega pravila. To ni mogoče zaradi platforme Drools.
- Definicija predpogojev ni možna. Če bo obstajala, se bo kopirala v vsako pravilo posebej.

2.1.8.4 Urejevalnik GDL

Fundacija je za potrebe pisanja in urejevanja razvila odprtokodni program GDL editor (angl. Guideline Definition Language Editor) [18]. GDL urejevalnik je namizna aplikacija, razvita za delovanje na več platformah. Uporabniku omogoča kreiranje, posodabljanje in zaganjanje datotek GDL na

podlagi definiranih arhetipov. Urejevalnik je prikazan na sliki 4.3. Arhetipi morajo biti uvoženi v program, ki je sestavljen iz vrstice menija, gumbov (omogočajo nalaganje, shranjevanje, dodajanje pravil, dodajanje povezav in zaganjanje) in iz devetih osnovnih oken. Ta omogočajo podajanje osnovnih informacij navodila, definiranje povezav do arhetipov in predpogojev, upravljanje s pravili, upravljanje s predpogoji, gradnjo novih pravil, upravljanje s terminologijo, izgradnjo GDL navodila in pregled izvajanja GDL datoteke v obliki HTML.

2.2 Standard HL7

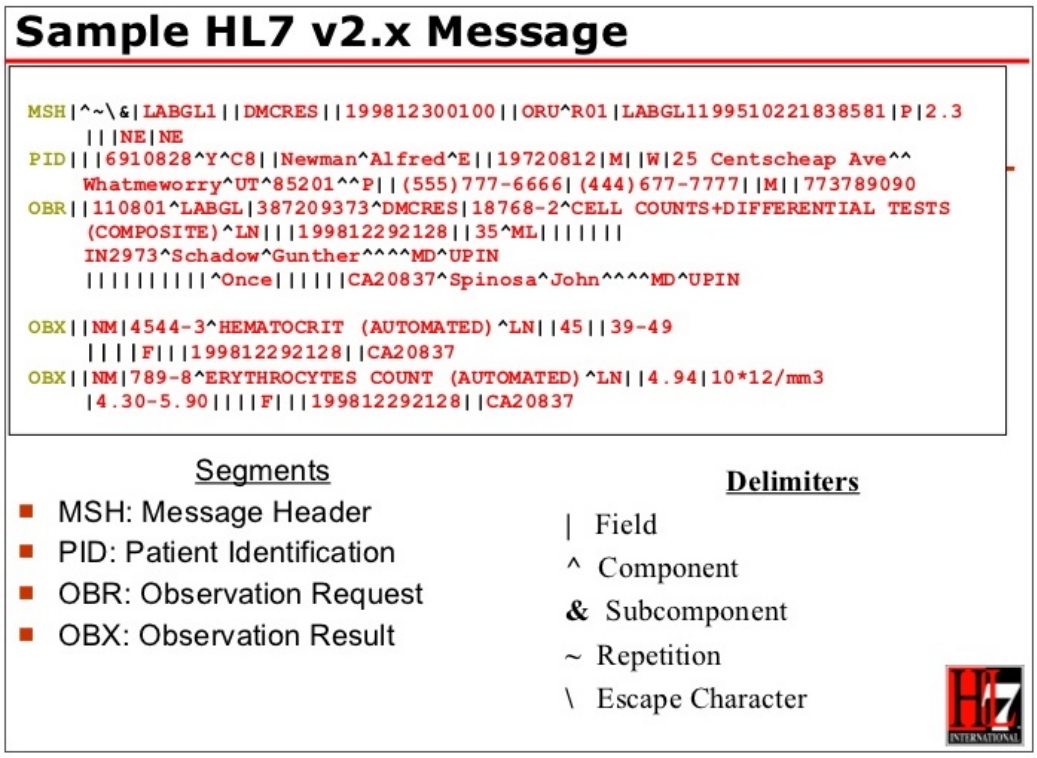
HL7 (Health Level-7) je množica mednarodnih standardov za prenašanje kliničnih in administrativnih podatkov med različnimi aplikacijami [19, 21, 42]. Standard in pripadajoča orodja skrbijo za izmenjavo, integracijo, deljenje in pridobivanje elektronskih zdravstvenih podatkov. Za standard skrbi mednarodna organizacija Health level seven international. V zdravstvenih ustanovah uporabljajo veliko število različnih aplikacij (tudi do več 100), ki morajo komunicirati med sabo. HL7 specificira več standardov, metodologij, pravil in navodil za komunikacijo različnih sistemov. To omogoča, da se informacije delijo in procesirajo v standardni in konsistentni obliki. Standardi so razdeljeni v 7 skupin [21]:

- Primarni standardi vsebujejo standarde za integracijo in skladnost.
- Temeljni standardi, ki definirajo osnovna orodja in temelje za definicijo standardov in infrastrukture, ki jo morajo uporabljati implementatorji.
- Standardi klinične in administrativne domene vsebujejo sporočila in dokumente za klinične posebnosti in skupine.
- Standardi EHR profila vsebujejo funkcionalne modele in profile, ki omogočajo upravljanje z elektronskim zdravstvenim zapisom.

- Implementacijska navodila vsebujejo navodila ter dokumente za podporo, ki se uporabljajo skupaj z obstoječimi standardi.
- Pravila in reference, ki vsebujejo tehnične specifikacije, programske strukture in navodila za razvijanje aplikacij in standardov.
- Izobrazba in zavedanje, ki ponuja orodja za uporabo s poskusno dobo, uporabne vire in orodja za razumevanje in uporabo standardov.

2.2.1 HL7 sporočilo verzije 2

HL7 standard verzije 2 ali Pipehat je bil ustvarjen leta 1989 [19, 20]. Definira množico elektronskih sporočil za podporo kliničnim, administrativnim, finančnim in logističnim procesom v zdravstvu. To omogoča interoperabilnost med različnimi sistemi, kot so: sistem za administracijo pacientov, laboratorijski informacijski sistem, farmacevtski sistem, računovodski sistem, sistem elektronskega medicinskega zapisa in sistem elektronskega zdravstvenega zapisa. Skozi leta je bilo izdanih več verzij 2.x, novejše verzije so kompatibilne s starejšimi. Sporočilo uporablja segmentno strukturo v obliki vrstic in enočrkovne ločilne znake. Posamezen segment ima več polj, ki so ločena z ločilnim znakom polja. Posamezno polje lahko vsebuje več podpolj, ki so ločeni z ločilnim znakom podpolja. Tudi posamezno podpolje lahko ima več podpolj. Za ločevanje segmentov se uporablja znak "|", za ločevanje polj znak "^", za ponavljanje pa znak "~". Vsak segment se začne z nizom treh črk, ki definira tip segmenta. Prvi segment sporočila je vedno MSH (angl. Message Header), ki vključuje sporočilo, ki določa, kakšni tipi segmentov se nahajajo v sporočilu.



Slika 2.7: Primer HL7 sporočila verzije 2

2.2.2 HL7 sporočilo verzije 3

Standard HL7 verzije 3 se je začel razvijati leta 1995, izdan je bil leta 2005 [19]. Temelji na formalni metodologiji HDF in objektno usmerjenih principih. HDF definira tudi procese, orodja, pravila in akterje, ki so potrebni za razvoj standarda. Temelj razvoja in ključni del standarda je RMI (angl. Risk Management Incident), ki definira potrebno vsebino v posameznih kliničnih in administrativnih področjih. Izraža semantične povezave med informacijami v poljih HL7 sporočila. Ta so v obliki XML sintakse. Standard se še vedno razvija ter poskuša olajšati interoperabilnost med različnimi zdravstvenimi sistemi.

Poglavje 3

Uporabljene tehnologije

3.1 Dogodkovno usmerjena arhitektura

Dogodkovno usmerjena arhitektura ali EDA (angl. Event driven arhitecture) je programski arhitekturni vzorec, ki sestoji iz pošiljateljev dogodka (kreirajo dogodek ali pretok dogodka) in prejemnikov (dogodke sprejemajo) [11]. Pošiljatelj in obenem tudi kreator dogodka edini ve, da se je dogodek zgodil. Najpogosteje se uporablja v aplikacijah in sistemih, ki posredujejo dogodke med različnimi komponentami. Te so med seboj šibko sklopljene. Razvoj takšnih aplikacij in sistemov omogoča, da so ti bolj odzivni in asinhroni. Takšna arhitektura rešuje probleme distribuiranega upravljanja podatkov in pošiljanja dogodkov več prejemnikom.

Dogodek je definiran kot sprememba stanja (npr. bolniku se spremeni stanje temperature iz 37,6 na 38,0 stopinj celzija). Velikokrat je razlog povzročitve dogodka ali pa je pri tem prispevalo sporočilo, vsebovano v dogodku. Zato takšno arhitekturo pogosto imenujemo sporočilna arhitektura (angl. Message driven arhitecture) [11]. Dogodek je sestavljen iz dveh delov. V glavi dogodka se nahajajo podatki o dogodku, kot so ime dogodka in čas kreiranja dogodka. V telesu dogodka je njegova vsebina. Ta je lahko poljubna [12]. Prednosti dogodkovne arhitekture so procesiranje dogodkov v realnem času in enostavno dodajanje novih prejemnikov v sistem brez inte-

gracije. Takšen sistem je skalabilen in enostaven za distribucijo. Pošiljatelji in prejemniki so ločeni. Dogodki so dostavljeni v razmeroma hitrem času. Zato se lahko prejemniki hitro odzovejo nanj.

Dogodkovna arhitektura uporablja dva modela:

- Model pošiljatelj/prejemnik uporablja strategijo sledenja prijavljenim prejemnikom. Ko je dogodek poslan v temo (angl. topic), ga pošlje vsakemu prejemniku, prijavljenemu na njo.

- Model dogodkovnega pretoka (angl. stream), pri katerem se dogodki zapisujejo v arhiv. Takšen tip dogodka je trajen. To pomeni, da se lahko prejemniki kadarkoli prijavijo v pretok in berejo dogodke naprej od zadnjega stanja. Dogodki znotraj pretoka so urejeni. Prejemniki sami vzdržujejo položaj v pretoku.

Glede na to, kako lahko prejemnik procesira dogodke, ločimo:

- Enostavno procesiranje dogodkov. Dogodek proži akcijo, ki jo izvede prejemnik.

- Kompleksno procesiranje dogodkov. Prejemnik proži akcijo na množici dogodkov. S pomočjo akcije išče vzorec v njihovih podatkih.

- Pretočno procesiranje dogodkov. Za takšen tip procesiranja dogodkov se uporabljajo pretočne platforme. Eden od primerov je sistem Apache Kafka.

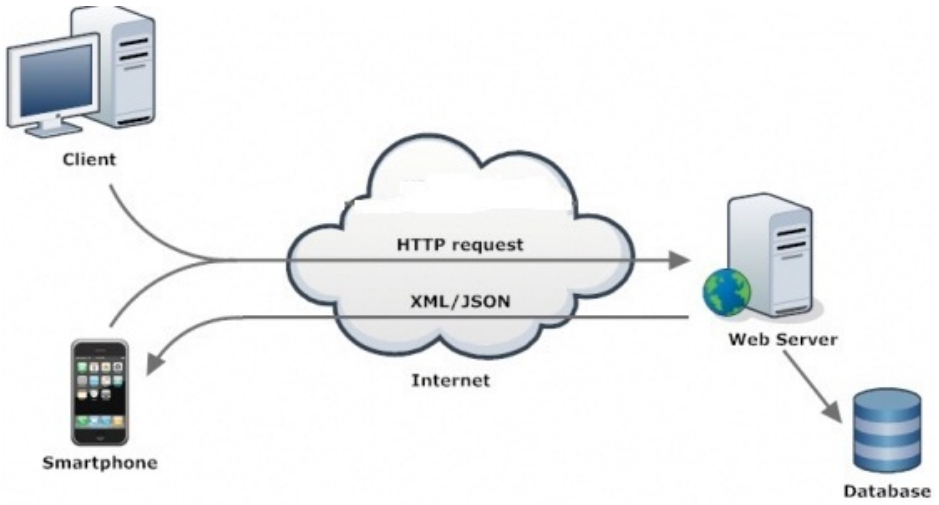
Arhitektura z uporabo dogodkovnega pretoka je še posebej primerna za razvoj prototipa in komunikacijo med posameznimi deli prototipa, saj platforma Think!EHR Platform™ in njena arhitektura omogočata, da lahko več aplikacij upravlja in operira z istimi podatki (semantična interoperabilnost). V realnosti to pomeni, da lahko z uporabo dogodkovno usmerjene arhitekture več aplikacij nad platformo procesira enake dogodke.

3.2 Protokol REST

Spletna storitev je predstavljena kot množica standardov in protokolov, ki se uporabljajo za izmenjavo podatkov med aplikacijami in sistemi, ki so napisani v različnih programskih jezikih [35]. Te aplikacije in sistemi uporabljajo spletne storitve za izmenjavo podatkov. Protokol REST (angl. Representational State Transfer) je arhitekturni vzorec, ki ga je prvi predstavil Roy Fielding leta 2000 in je namenjen za razvoj spletnih storitev [32]. Ključni element pri protokolu so njegovi viri. Strežnik in odjemalec si izmenjujeta reprezentacijo teh v različnih oblikah (npr. JSON, Text ali pa XML). Drugi pomemben del protokola so metode, ki so definirane nad viri. Komunikacija za izmenjavo podatkov med odjemalcem in strežnikom temelji na protokolu HTTP [33]. Protokol REST je protokol brez stanja (angl. stateless protocol). Najpogosteje uporabljene HTTP metode v arhitekturi REST so:

- GET, ki omogoča bralni dostop do vira.
- PUT, ki omogoča kreiranje novega vira.
- DELETE, ki omogoča brisanje vira.
- POST, ki omogoča posodabljanje obstoječega vira ali kreiranje novega.
- OPTION, ki omogoča pridobitev liste možnih operacij nad virom.

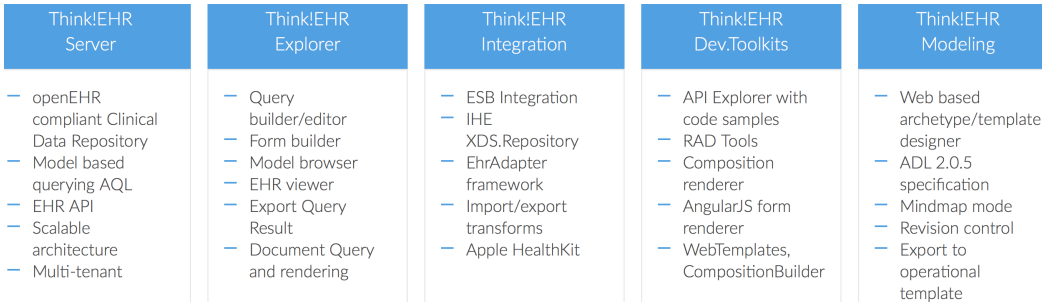
REST storitve so osnova za gradnjo programskih vmesnikov. API je skupek protokolov in orodij za razvoj programske opreme [3]. Določa operacije, vhodne in izhodne podatke nad posamezno komponento. Omogoča enostavno razširljivost, skalabilnost, varnost in neodvisnost. Zaradi enostavnosti in konsistentnosti se REST storitve vedno bolj uveljavljajo in uporabljajo pri razvoju programske opreme.



Slika 3.1: Arhitektura REST spletnih storitev

3.3 Platforma Think!EHR Platform™

Think!EHR Platform™ platforma je produkt podjetja Marand d. o. o.. Je visoko performančna rešitev, ki omogoča shranjevanje, poizvedovanje, upravljanje in pridobivanje strukturiranih elektronskih zdravstvenih podatkov, ki temeljijo na zadnjem standardu openEHR specifikacije [34]. Vsi klinični podatki so shranjeni v arhetipe in predloge, kar omogoča standardiziran zapis, pridobivanje podatkov, njihovo terminološko validacijo in podporo več jezikom, terminologijam in enotam. Aplikacije lahko dostopajo do platforme na različne načine, kot so: REST, SMART, HL7 in FHIR. Platforma omogoča hitrejše razvijanje zdravstvenih aplikacij naslednje generacije, ki temeljijo na že obstoječih arhetipih in predlogah, izboljšanje sprejemanja odločitev, možnosti raziskovanja in izgradnje podatkovnih skladišč, ki temeljijo na standardnih podatkovnih modelih.



Slika 3.2: Sestava Think!EHR Platform™ platforme

Glavni del platforme je Think!EHR Platform™ strežnik, ki omogoča razvijalcem shranjevanje elektronskih zdravstvenih zapisov in izvajanje AQL poizvedb.

3.4 Proženje dogodkov s pomočjo Think!EHR Platform™ platforme

Podobno kot proženje dogodkov v relacijskih podatkovnih bazah se pojavi potreba po proženju dogodkov kot reakcija na določen podatkovni zapis, ki bo shranjen, izbrisan ali posodobljen. Na platformi so definirane AQL poizvedbe. Ob vsakem brisanju, posodabljanju ali kreiranju podatkovnega zapisa se proži vsaka od njih. Če vrne neprazen rezultat, se kreira in pošlje dogodek. Podatki, ki jih vrne dogodek, so opredeljeni z AQL poizvedbo. To pomeni, da bo podatke, ki jih vrača poizvedba, vračal tudi dogodek. Dogodek vrača stare podatke (to so podatki, ko se AQL poizvedba izvede nad obstoječim podatkovnim zapisom) in nove podatke (to so podatki, ko se AQL izvede kot posodabljanje podatkovnega zapisa). Potrebno je opozoriti, da bodo stari podatki ob kreiranju, novi pa ob brisanju vedno imeli vrednost null. Poleg podatkov, ki jih vrača poizvedba, dogodek vsebuje tudi nekatere druge podatke, kot npr. EHR identifikator. Dogodek je opredeljen z eno izmed dveh faz. Ta je lahko PRE_COMMIT ali POST_COMMIT. PRE_COMMIT faza je sinhrona. Ima dostop do podatka podatkovnega za-

pisa, določenim delom lahko preprečijo, da se shranijo (podatkovni zapis se shrani šele, ko je prejemnik uspešno prejel dogodek. Vsaka napaka prepreči shranjevanje). Dogodek v POST_COMMIT fazi se izvede šele, ko je podatkovni zapis uspešno shranjen. Think!EHR Platform™ strežnik podpira tri tipe dogodkov: JavaScript, Push, and Queue.

3.4.1 JavaScript dogodki

Tak tip dogodkov naj bi bili skoraj vedno v fazi POST_COMMIT. Uporabniku omogoča pisanje preprostih JavaScript programov, ki reagirajo na podatke dogodka.

```
select
    a/data[at0002]/events[at0003]/data[at0001]/items[at0004]/
        value
from EHR e
contains OBSERVATION a[openEHR-EHR-OBSERVATION.body_temperature.
    v1]
where
    a/data[at0002]/events[at0003]/data[at0001]/items[at0004]/
        value/magnitude>39 and
    a/data[at0002]/events[at0003]/data[at0001]/items[at0004]/
        value/units='C'
```

Slika 3.3: Primer AQL poizvedbe JavaScript dogodka

```
var temp = eventData.getNewResults().get(0)[0];
var map = helper.createMap();
map.put('subject', 'Temperature over 39');
notifierRegistry.get('mail').notify('physician@hospital.com', '
    Warning! Your patient has %temperature of ' + temp.
    getMagnitude(), map);
```

Slika 3.4: Primer pridobivanja podatkov JavaScript dogodka

Takšen dogodek se zgodi vsakič, ko AQL poizvedba vrne neprazen rezultat. Spremenljivke eventData, helper in notifierRegistry so vstavljene (angl. injected) v JavaScript izvajalno okolje. Iz spremenljivke eventData se lahko pridobijo naslednje informacije:

- String getEhrUid() vrne EHR identifikator, v katerem je ta podatkovni zapis shranjen.
- List<Object[]> getOldResults() vrne rezultat izvajanja AQL poizvedbe dogodka obstoječega podatkovnega zapisa. V primeru akcije kreiranja podatkovnega zapisa funkcija vrača vrednost null.
- List<Object[ ]> getNewResults() vrne rezultat izvajanja AQL poizvedbe dogodka novega podatkovnega zapisa. V primeru akcije brisanja podatkovnega zapisa funkcija vrača vrednost null.

Spremenljivka helper služi za kreiranje tabele fiksne dolžine ali mape. Spremenljivka notifierRegistry služi kot register različnih urejevalcev obvestil (angl. Notification handlers). To omogoča JavaScriptu pošiljanje sms ali email sporočil. Podprti so tudi drugi tipi sporočil.

```
JSEventDto dto = new JSEventDto();
dto.setName("jsevent");
dto.setAqlString("SELECT c/name/value FROM Composition c");
dto.setPhase(Phase.POST_COMMIT);
dto.setActive(false);
dto.setJavascript("print('Hello world!')");
return eventService.addEvent(sessionId, dto);
```

Slika 3.5: Primer kreiranja JavaScript dogodka

3.4.2 Push dogodki

Ko se sproži takšen tip dogodkov, se izvede metoda HTTP POST na specificiran URL v JSON ali XML obliki. Dogodkom v PRE_COMMIT fazi je preprečeno, da je podatkovni zapis v primeru napake shranjen (vključno s komunikacijskimi napakami pri pošiljanju dogodka). Napaka je posredovana ustvarjalcu podatkovnega zapisa. Če se napaka zgodi pri dogodkih v POST_COMMIT fazi, so le-ti shranjeni v vrsto. Izvedejo se pozneje, če jim ne poteče veljavnost. Čas ponovnega pošiljanja je privzeto nastavljen na eno uro, čas veljavnosti pa na en dan.

```
{
  "oldResults": [
      {
        "name": "Vitals",
        "#1": "685940f3-d98a-47fe-a85d-1bf64135bfbe::default::1"
      }
    ],
  "newResults": [
      {
        "name": "Vitals",
        "#1": "685940f3-d98a-47fe-a85d-1bf64135bfbe::default::2"
      }
    ],
    "operation": "update",
    "ehrId": "id"
}
```


Slika 3.6: Primer preprostega posodabljanja podatkovnega zapisa

Spremenljivke ehrId, oldResults in newResults nosijo enake informacije, ki jih vračajo istoimenske funkcije pri JavaScript dogodkih. Spremenljivka operation nosi podatek o tipu operacije, ki je sprožila dogodek (create, update ali delete).

```
PushEventDto dto = new PushEventDto();
dto.setName("pushevent");
dto.setAqlString("SELECT c/name/value as name, c/uid/value FROM
    Composition c");
dto.setPhase(Phase.POST_COMMIT);
dto.setActive(false);
dto.setType("JSON");
dto.setDestination("http://localhost:8080/event/post");
dto.setRetryInterval(Duration.ofMinutes(10L));
dto.setTimeToLive(Duration.ofHours(12L));
return eventService.addEvent(sessionId, dto);
```

Slika 3.7: Kreiranje dogodka tipa Push

3.4.3 Queue dogodki

Ko se zgodi takšen tip dogodkov, je poslan v temo (angl. topic) sistema Apache Kafka. Ta je lahko konfiguriran, da avtomatsko kreira temo, ki še ne obstaja ali pa jo moramo kreirati sami. Ime dogodka se smatra za ključ sporočila. Sporočilo je lahko v JSON ali XML obliki. Določi se mu lahko particija teme. V primeru, da ta ni definirana, se pojavitve dogodkov razpršijo čez vse možne particije s pomočjo algoritma Round-robin. Če so dogodki konfigurirani s PRE_COMMIT fazo, je njihova dostava zagotovljena v smislu, da dokler dostava ni uspešna, podatkovni zapis ni shranjen. Če so dogodki v POST_COMMIT fazi, njihova dostava ni zagotovljena zaradi nedostopnosti sistema Apache Kafka ali pa Think!EHR Platform™ strežnika v majhnem časovnem intervalu med shranjevanjem in odpošiljanjem dogodka. Dogodek bo v takem primeru izgubljen.

```
QueueEventDto dto = new QueueEventDto();
dto.setName("queueevent");
dto.setAqlString("SELECT c/name/value as name, c/uid/value FROM
    Composition c");
dto.setPhase(Phase.POST_COMMIT);
dto.setActive(false);
dto.setType("JSON");
dto.setTopic("event1")
dto.setPartition(null);
return eventService.addEvent(sessionId, dto);
```

Slika 3.8: Kreiranje dogodka tipa Queue

3.5 Izvajanje GDL v platformi Think!EHR Platform™

Platforma omogoča izvajanje GDL navodil kot pomoč pri izvajanju podpore pri odločanju. Dostopna je na podlagi aplikacijsko programskega vmesnika [4]. Do njih dostopamo preko protokola REST. Definirani in dostopni so naslednji viri:

- /guide, nad katerim sta omogočeni metodi GET in POST. GET vrača seznam vseh definiranih GDL datotek. POST omogoča kreiranje nove GDL datoteke.
- /guide/guideId, nad katerim sta omogočeni metodi GET in DELETE. GET vrača specifično GDL datoteko, DELETE jo zbriše.
- /guide/guideId/execute/ehrIds/composition/templateId, nad katerim je omogočena metoda POST. Ta izvede GDL datoteko nad specifičnim podatkovnim zapisom. Spremenljivka templateId predstavlja identifikator vhodne predloge, ehrId pa identifikatorje pacienta.

- /guide/guideId/execute/ehrIds/query, nad katerim je omogočena metoda POST. Ta izvede GDL nad obstoječimi podatkovnimi zapisi.

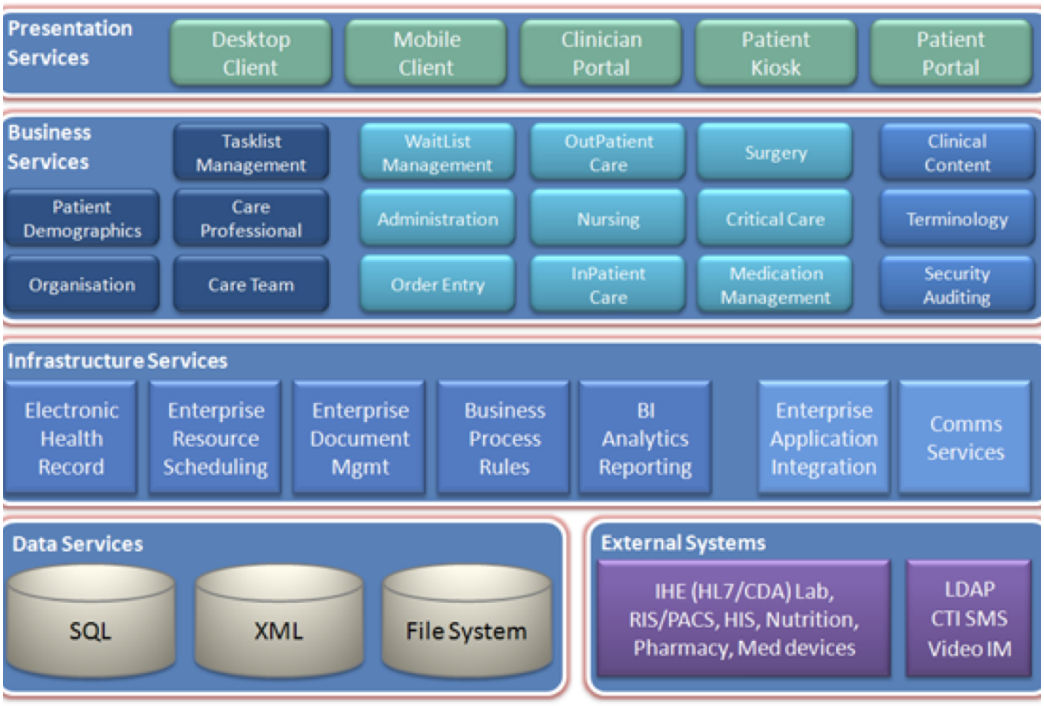
Zadnja dva vira, poleg potrebnih parametrov, omogočata še nekatere dodatne parametre:

- Authorization - za avtorizacijo s Think!EHR Platform™ platformo se uporablja avtorizacija tipa basic. Je obvezen parameter v primeru, da parameter Ehr-Session ni prisoten.
- Ehr-Session - podobno kot Authorization se uporablja za avtorizacijo s platformo Think!EHR Platform™. Obvezen je v primeru, da parameter Authorization ni prisoten.
- CompositionBody - v njem pošljemo podatke podatkovnega zapisa v obliki seznama poti in vrednosti posameznega vozlišča. Podamo ga v obliki JSON formata.
- Persist - določa, ali se bo rezultat pravila shranil v Think!EHR Platform™ platformo.
- Trace – določa, če naj odgovor vsebuje sled izvajanja.
- Format – določa, kakšnega tipa naj bo odgovor.

3.6 Think!Clinical

Think!Clinical je klinični informacijski sistem, ki ga je razvilo podjetje Marand d. o. o. Uporabljata ga Pediatrična klinika in Onkološki inštitut v Ljubljani. Za shranjevanje podatkov poleg relacijske podatkovne baze Oracle uporablja Think!EHR Platform™ platformo. Informacijski sistem vsebuje številne module, ki omogočajo učinkovito izvajanje kliničnih procesov. Nekateri od teh so: administracija uporabinkov, podatki o pacientu, eNaročanje, eRecept, sistem za upravljanje zdravil po prinncipu zaprte zanke, modul za podporo dela medicinskih ekip na Kliniki za nuklearno medicino, modul za

podporo celotnega procesa naročanja laboratorijskih naročil in odvzema biološkega materiala, modul za zdravstveno nego. Moduli so grupirani v tri dele. Prvi del se nanaša na pacienta. Ponuja različne module, ki se nanašajo na klinični proces posameznika. Drugi del so moduli, ki se nanašajo na klinični proces organizacije. Tretji del pokriva upravljanje z aplikacijo in je namenjen IT službi. Omogoča avtomatizacijo delovnega procesa, elektronski zdravstveni zapis podatkov pacienta, prepoznavanje govora, izboljša kakovost informacij ter varnost in zasebnost osebnih podatkov. Integracija z zunanjimi sistemi poteka po informacijskih standardih, kot so openEHR, HL7 in IHE, kar mu omogoča hitro, konsistentno in enostavno integracijo z zunanjimi sistemi in z različnimi napravami. Arhitektura, ki je predstavljena na sliki 3.9, mu omogoča zmogljivost, učinkovitost in razpoložljivost. Sestavljena je iz več strežnikov, ki komunicirajo med seboj. Največji med njimi je aplikacijski strežnik. Integracijski strežnik skrbi za integracijo z zunanjimi sistemi in napravami, BPM (angl. Business Process Managment) strežnik skrbi za poslovne procese, urniški (angl. Scheduling) strežnik skrbi za shranjevanje ter upravljanje urnikov in naročil.

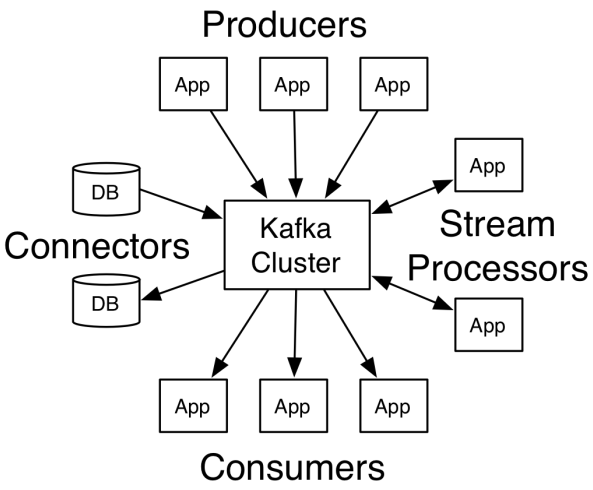


Slika 3.9: Arhitektura sistema Think!Clinical

3.7 Apache Kafka

Apache Kafka je odprtokodna pretočna platforma, ki deluje kot skupek enega ali večih posredniških spletnih strežnikov. Razvilo jo je podjetje Apache Software Foundation v programskem jeziku Java in Scala [2, 40]. Arhitektura aplikacije je prikazana na sliki 3.10. Platforma omogoča objavljanje in prijave na pretok zapisov, shranjevanje ter njegovo procesiranje. To omogoča gradnjo pretočnih aplikacij in sistemov za prenos pretoka podatkov med njimi, njegovo transformacijo in procesiranje. Pretoki podatkov so shranjeni v teme, podatek je sestavljen iz ključa, vrednosti ter zapisa časa kreiranja. Platforma je sestavljena iz štirih ključnih aplikacijsko programskih vmesnikov: API obljavljalcev (angl. Producer API), ki omogoča objavo pretoka dogodkov v temo; API prejemnikov (angl. Consumer API), ki omogoča pri-

javo na teme in procesiranje pretoka zapisov; API pretoka (angl. Stream API), ki deluje kot procesor pretoka, kar mu omogoča, da vhodni pretok teme transformira v izhodni pretok teme; API povezovanja (angl. Connector API), ki omogoča gradnjo aplikacij, ki kot obljavljalci ali prejemniki uporabljajo teme [22]. Prednost uporabe sistema je možnost skaliranja, procesiranja in omogočanje več prejemnikov za vsako posamezno temo. Apache Kafka zagotavlja vrstni red sporočil s pomočjo particij znotraj tem, nizke latence in shranjevanje podatkov na disk. Trenutne verzije že omogočajo pretočno procesiranje. Ker so podatki shranjeni na disk, lahko enako obravnava podatke iz preteklosti in sedanjosti. Vsak prejemnik lahko procesira dogodke od zadnjega prejetega naprej.



Slika 3.10: Arhitektura Apache Kafke

3.8 Sistem za upravljanje poslovnih pravil

Z razvojem kompleksnih sistemov se pojavi vse večja zahteva po sistemih za upravljanje s poslovnimi pravili. Razvijejo se sistemi za upravljanje poslovnih pravil BRMS (angl. Business rule management system). Omogočajo defini-

ranje, kreiranje, izvajanje in upravljanje poslovnih pravil. Da nek sistem imenujemo sistem za upravljanje poslovnih pravil, mora ta vsebovati repozitorij, orodja in okolje za izvajanje pravil [7, 8]. Repozitorij omogoča ločitev logike pravil od logike aplikacije. Vsa logika pravil je zato centralizirana na enem mestu. Orodja omogočajo razvijalcem lažji razvoj in integracijo s sistemom in definiranjem pravil. Ti služijo kot podpora pri odločanju. Okolje za izvajanje poslovnih pravil omogoča njihovo izvajanje znotraj sistema za upravljanje poslovnih pravil. Uporaba takšnih sistemov ima nekaj ključnih prednosti. Pravila so zbrana na enem mestu, saj so ločena od ostale logike sistema. Zaradi tega jih lažje vzdržujemo. Različni sistemi definirajo različne jezike za zapis pravil. Ker so vsa zapisana v enakem jeziku, so zato bolj berljiva. Zaradi naprednih algoritmov je njihovo izvajanje hitrejše. Pravila lahko definira tudi analitik. Kljub kar nekaj prednostim ima takšen sistem tudi nekaj slabosti. Glavna izmed njih je odvisnost enega od sistemov za upravljanje poslovnih pravil, saj je vsak sistem drugače implementiran, uporablja svoj sistem za izvajanje, svoja orodja in svojo sintakso. V literaturi je možno zaslediti več različnih tehnologij za upravljanje poslovnih pravil. Nekatere od teh so IBM Websphere ILOG, Easy rules, RuleBook, OpenL tablets in Drools [7]. Ker openEHR standard v svojih specifikacijah za uporabo navaja Drools, ki ga uporablja tudi GDL editor in platforma Think!EHR Platform™, bom v poglavju 3.8.1 podrobneje predstavil slednjega.

3.8.1 Drools

Drools je odprtokodna platforma za upravljanje poslovnih pravil. Njegova zbirka orodij omogoča kreiranje, posodabljanje in upravljanje s pravili, njihovo izvajanje in nekaj vtičnikov za razvoj [13, 14]. Nekatera izmed orodij so Drools Workbench, ki omogoča kreiranje in upravljanje s pravili, Drools Expert, ki omogoča izvajanje poslovnih pravil, Drools Fusion, ki omogoča delo z dogodki, produkt jBPM, ki omogoča integracijo pravila s procesom, OptaPlaner, ki omogča avtomatsko planiranje in Kie Server, ki omogoča kreiranje nove instance in izvajanje pravil preko protokola REST ali JMS

klicev [14]. Napisan je v programskem jeziku Java. Razvilo ga je podjetje JBoss, divizija Red hat Inc. Podjetje ponuja tudi njegovo razširljivo plačljivo verzijo, imenovano JBoss Enterprise BRMS. Drools uporablja za definicijo pravil svoj jezik, imenovan DRL (angl. Drools rule language). Pravila so zapisana v datotekah s končnico drl. Temeljijo na principu kdaj-nato (angl. when-then). Jezik poleg programskega zapisa pravil podpira tudi odločitvene tabele (angl. Decision tables). Te omogočajo definiranje pravil s pomočjo tabele. So lažje za razumevanje in dodajanje novih pravil. Sistem za izvajanje poslovnih pravil uporablja razširjeno različico Rate algoritma, imenovano RateOO, ki je prilagojena objektno orientiranim sistemom [30, 31]. Njegove glavne prednosti so ločitev programske logike in podatkov, centraliziranost, hitrost, razširitve, nekompleksnost in možnost integracije njegovih orodij pri razvoju. To programerju omogoča hitrejši razvoj, lažje izvajanje kode ter iskanje napak.

Poglavje 4

Izdelava navodila GDL na primeru anemije pri bolnikih s kronično ledvično boleznijo

4.1 Anemičnost pri bolnikih s kronično ledvično boleznijo

4.1.1 Kronična ledvična bolezen

Anemija ali slabokrvnost je zaplet, ki ga lahko pričakujemo pri večini bolnikov s kronično ledvično boleznijo (angl. Chronical kidney disease - CKD), ki jo pogosto odkrijejo zelo pozno [25]. Posledice bolezni so oteženo zdravljenje in odpoved ledvic. Pri bolnikih s sladkorno boleznijo, bolnikih z arterijsko hipertenzijo in bolnikih z boleznimi srca in ožilja je bolezen še pogostejša [25]. Ker ti hodijo na redna zdravljenja, bolezen odkrijemo hitreje. Na njo lahko vplivajo še kajenje, starost in debelost. Pogosteje se lahko pojavi pri bolnikih, katerih sorodniki imajo težave s kronično ledvično boleznijo. Najpogostejši kazatelji kronične ledvične bolezni so visok krvni tlak, prenizka hitrost čiščenja krvi v minuti, srbečica, slabost, spremembe na koži, živčevju in sečilih ter beljakovine v seču. Pogost zaplet, ki se pojavlja pri bolnikih

s kronično ledvično boleznijo, je anemija. Ledvice imajo dve glavni funkciji: endokrina funkcija (izločanje hormonov) in očiščevalna funkcija (čiščenje strupov, soli in vode iz telesa). S slabljenjem ledvic začnejo odmirati celice, ki proizvajajo eritropoetin. Te so ključne za nastajanje rdečih krvničk v kostnem mozgu. Ker se koncentracija eritropoetina niža, je manjša tudi spodbuda za nastajanje rdečih krvničk. To je osnovni mehanizem nastanka ledvične anemije [26]. Ravno zaradi neizrazitih znakov jo težko odkrijemo v zgodnjih fazah. Zaradi tega je za bolnike kronične ledvične bolezni ključno, da hitro odkrijemo bolezen. Ob morebitni odpovedi ledvic je treba bolniku čistiti kri s pomočjo dialize ali presaditi ledvice. Bolezen lahko preprečimo z zdravo prehrano, dietami, vzdrževanjem telesne teže in telesno dejavnostjo.

4.1.2 Ugotavljanje anemije

Za lažje ugotavljanje anemije pri pacientih s kronično ledvično boleznijo se upoštevajo smernice KDIGO (angl. Kidney Disease, Improving Global Outcomes) [24]. Te narekujejo, da če pacient ima diagnozo, ki opredeljuje kronično ledvično bolezen in opredeljeno raven hemoglobina v krvi, je ta ogrožen za anemijo.

Odločitvena tabela predstavlja način ugotavljanja anemije pri bolnikih s kronično ledvično boleznijo. Anemija je pri takšnem bolniku prisotna, če ima eno od petih diagnoz kronične ledvične bolezni, je star več kot dve leti in ima raven hemoglobina manj kot 11gd/dl ali pa je star manj kot dve leti in ima raven hemoglobina majn kot 10.5 g/dl. V nasprotnem primeru anemija pri bolniku ni prisotna.

Diagnoza MKB10	**Raven laboratorijske preiskave K-Hemoglobin**	**Ogroženost za anemijo**
N181: Kronična ledvična bolezen (KLB), stopnja 1	≥2 leta Hb ≤11 g/dL ali <2 let Hb ≤10.5 g/dL	Da
N182: Kronična ledvična bolezen (KLB), stopnja 2	≥2 leta Hb ≤11 g/dL ali <2 let Hb ≤10.5 g/dL	Da
N183: Kronična ledvična bolezen (KLB), stopnja 3	≥2 leta Hb ≤11 g/dL ali <2 let Hb ≤10.5 g/dL	Da
N184: Kronična ledvična bolezen (KLB), stopnja 4	≥2 leta Hb ≤11 g/dL ali <2 let Hb ≤10.5 g/dL	Da
N185: Kronična ledvična bolezen (KLB), stopnja 5	≥2 leta Hb ≤11 g/dL ali <2 let Hb ≤10.5 g/dL	Da
N181: Kronična ledvična bolezen (KLB), stopnja 1	≥2 leta Hb > 11 g/dL ali <2 let Hb > 10.5 g/dL	Ne
N182: Kronična ledvična bolezen (KLB), stopnja 2	≥2 leta Hb > 11 g/dL ali <2 let Hb > 10.5 g/dL	Ne
N183: Kronična ledvična bolezen (KLB), stopnja 3	≥2 leta Hb > 11 g/dL ali <2 let Hb > 10.5 g/dL	Ne
N184: Kronična ledvična bolezen (KLB), stopnja 4	≥2 leta Hb > 11 g/dL ali <2 let Hb > 10.5 g/dL	Ne
N185: Kronična ledvična bolezen (KLB), stopnja 5	≥2 leta Hb > 11 g/dL ali <2 let Hb > 10.5 g/dL	Ne

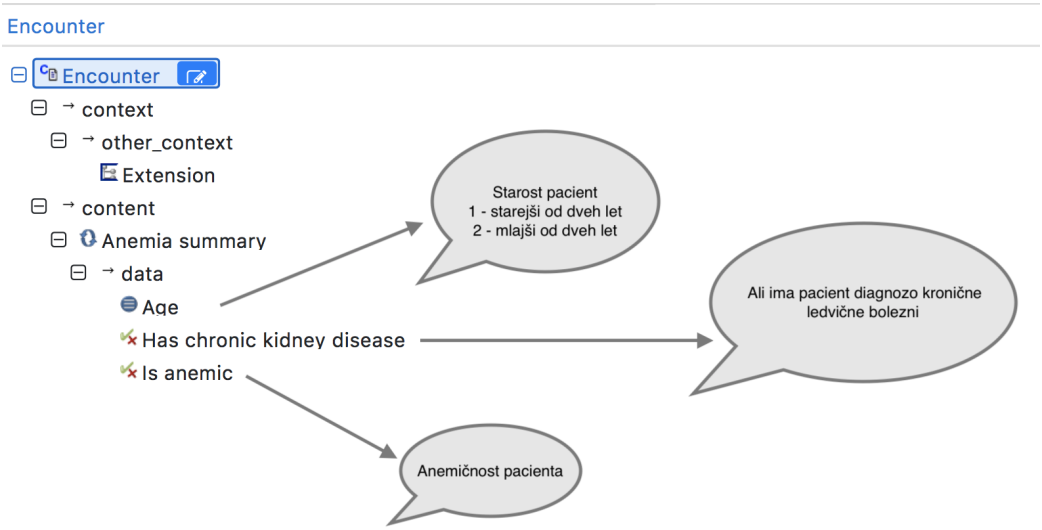
Tabela 4.1: Odločitvena tabela pri določanju prisotnosti anemije pri bolnikih s kronično ledvično boleznijo

4.2 Potek izdelave navodila GDL na primeru anemije

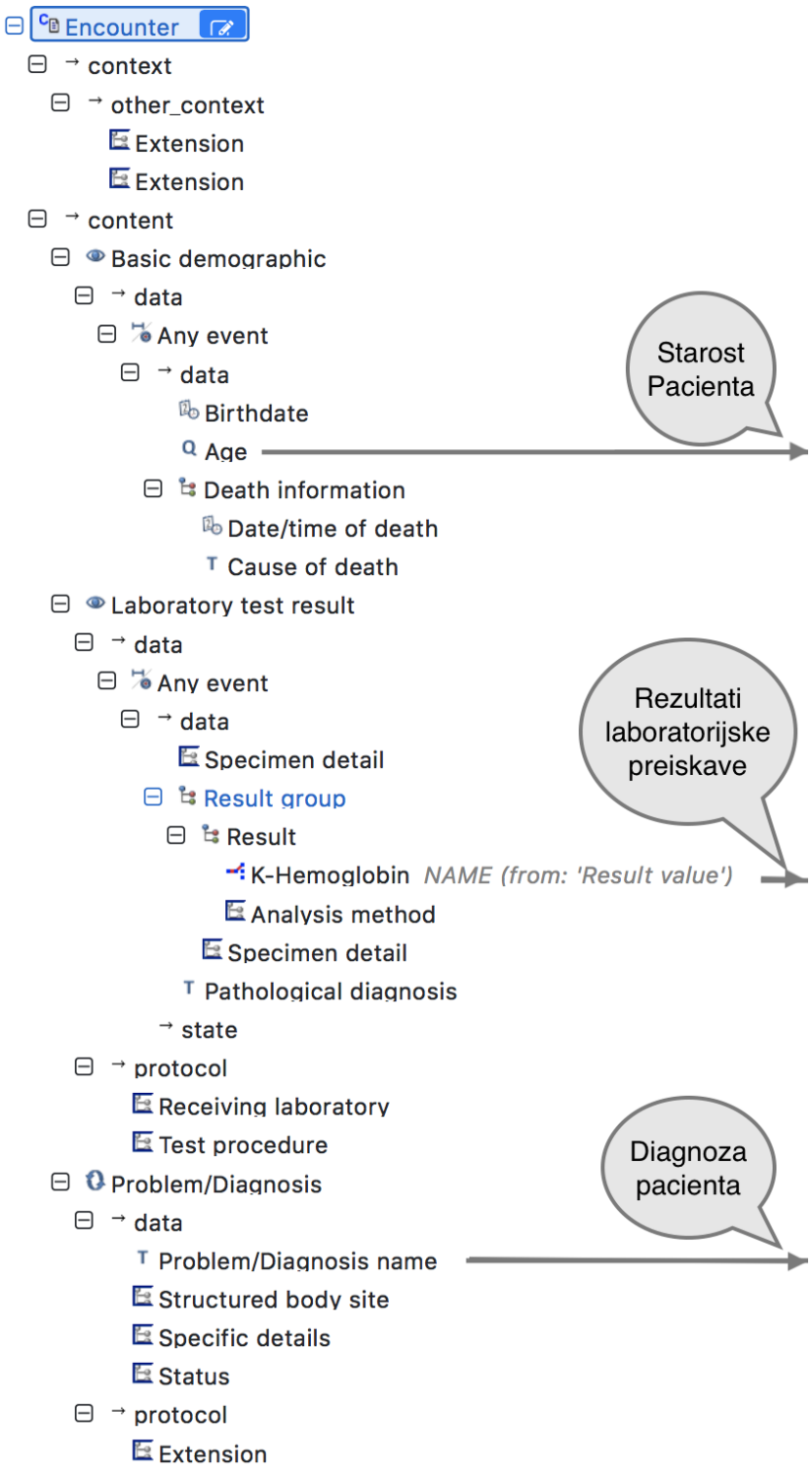
Na primeru anemije v diplomskem delu v tem poglavju predstavim potek izdelovanja navodila in preizkus njegovega delovanja z uporabo programa GDL editor, ki ga v svojih standardih omenja specifikacija openEHR. Primer temelji na ugotavljanju anemije pri bolnikih s kronično ledvično boleznijo. Postopki za njeno ugotavljanje so opisani v poglavju 4.1.2. Delovanje urejevalnika je neodvisno od platforme, na kateri se izvaja. Za njegovo izvajanje je potrebno imeti na sistemu nameščen JRE (angl. Java Runtime Environment), verzije 1.6 ali več. Je odprtokoden program. Na voljo je že nekaj kreiranih terminologij, predlog, arhetipov in navodil, ki jih je za demonstracijske namene razvilo podjetje Cambio Healthcare Systems. Po odprtju programa je potrebno nastaviti pot do arhetipov, predlog in drugih pomembnih datotek. To storimo tako, da v vrstici menija izberemo Konfiguriranje (angl. Configuration) in ustrezno nastavimo poti.

Naprej sem s pomočjo programa ADL urejevalnik (angl. ADL designer) kreiral predlogi za vhodne in izhodne podatke. ADL designer je orodje za grajenje predlog in arhetipov [1]. Za vhodne podatke služi predloga AnemiaGDL. Ta vsebuje tri arhetipe: laboratorijske rezultate, demografske podatke pacienta in podatke o diagnozi. Vsak od njih vsebuje del podatkov, ki so potrebni za določitev anemičnosti pacienta. Arhetipi Basic Demographic, Laboratory test result in Problem/Diagnosis so bili pridobljeni iz upravljavca kliničnega znanja CKM.

Za izhodne podatke služi predloga AnemiaGDLResults. Ta vsebuje podatke o starosti pacienta (1 – starejši od dveh let in 2 – mlajši od dveh let), logični podatek o pacientovi diagnozi za kronično ledvično bolezen in logično podatek o anemičnosti pacienta. Arhetip Anemia summary je bil zgrajen samostojno.



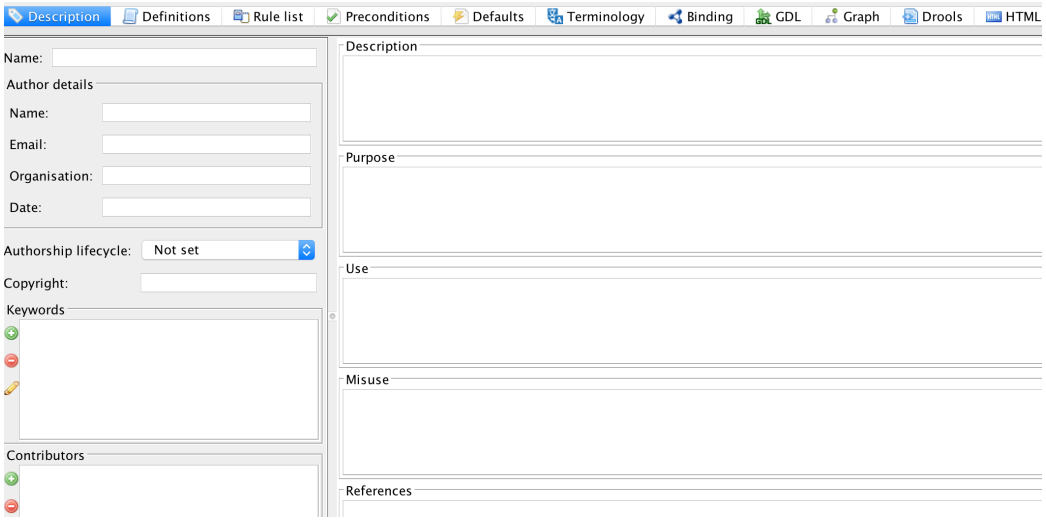
Slika 4.1: Izhodna predloga, zgrajena s pomočjo programa ADL designer



Slika 4.2: Vhodna predloga, zgrajena s pomočjo programa ADL designer

Arhetipe in predloge izvozimo iz programa ADL designer in jih uvozimo v program GDL editor. S tem korakom lahko začnemo kreirati novo navodilo.

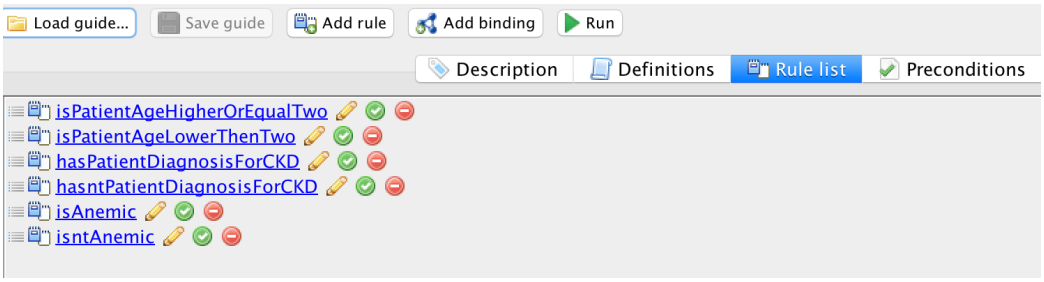
Kreiranje navodil se izvede v večih korakih. Najprej v okno Opis (angl. Description) vnesemo osnovne podatke. Okno nam omogoča vnos informacij o metapodatkih pravila, kot so: ime avtorja in pravila, pravice za kopiranje, ključne besede, imena soavtorjev, opis pravila, namen, uporabo, povezave in informacije o tem, kdaj tega pravila ne smemo uporabljati. Po prvem koraku je potrebno shraniti novo navodilo.



Slika 4.3: Vpis meta podatkov novega pravila

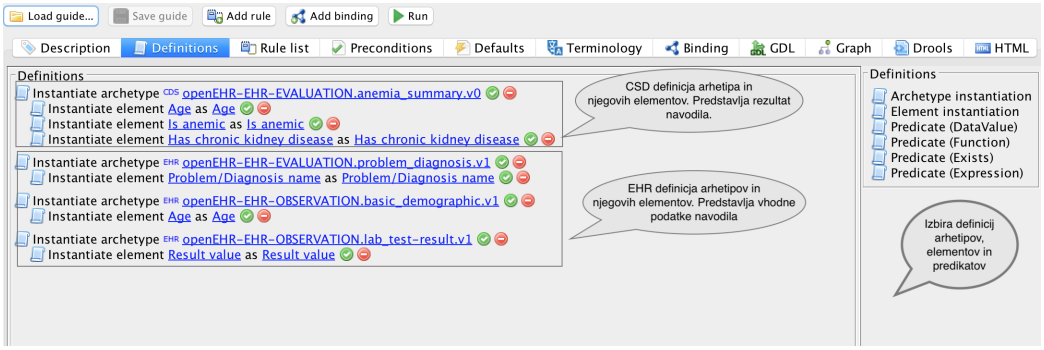
Po vpisanih metapodatkih je potrebno dodati definicije. To lahko storimo s pomočjo okna Definicija (angl. Definition), v katerem lahko dodamo poljubno število EHR in CSD elementov. Na desni strani lahko s pomočjo principa povleči in spusti (angl. drag and drop) iz podokna izberemo arhetipe, elemente in predikate. S klikom na ime arhetipa in element lahko s pomočjo izbirnega okna izberemo ustrezne vrednosti. Za vhodne podatke izberemo vrednost EHR, za izhodne pa CSD. Za definicijo EHR arhetipov izberemo arhetipe za demografijo pacienta, njegove diagnoze in rezultate laboratorijskih preiskav, za CSD arhetip pa arhetip s povzetkom anemije. Definicijam arhetipov nato dodamo elemente. Za element izberemo atribut arhetipa,

kateremu pripada. Služijo kot spremenljivke v pravilu.



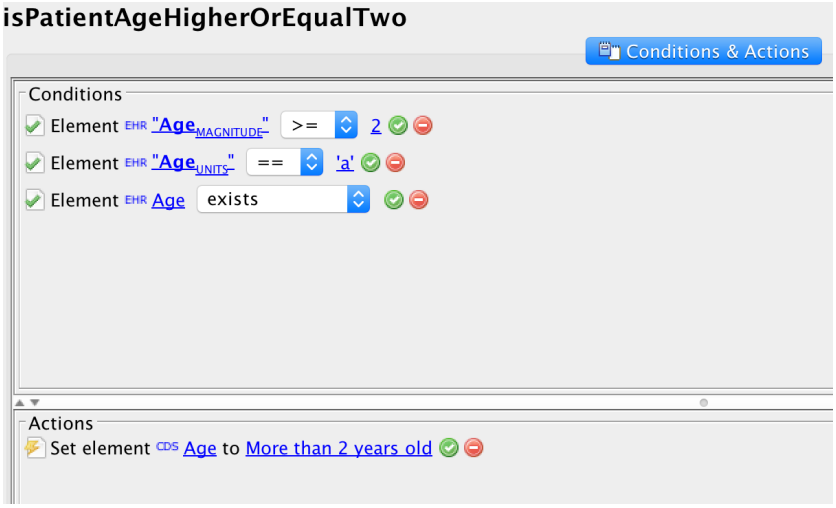
Slika 4.4: Definicija arhetipov in njihovih elementov

V naslednjem koraku moramo definirati pravila, ki jih zgradimo s pomočjo definicij pravil. Novo pravilo kreiramo s pomočjo gumba Dodaj pravilo (angl. Add rule). Najprej definiramo seznam pravil, nato vsebino vsakega pravila. Vsako pravilo vsebuje pogoje in rezultat. Možne pogoje izberemo na desni strani zgoraj, možne pridobitve rezultata pa desno spodaj.



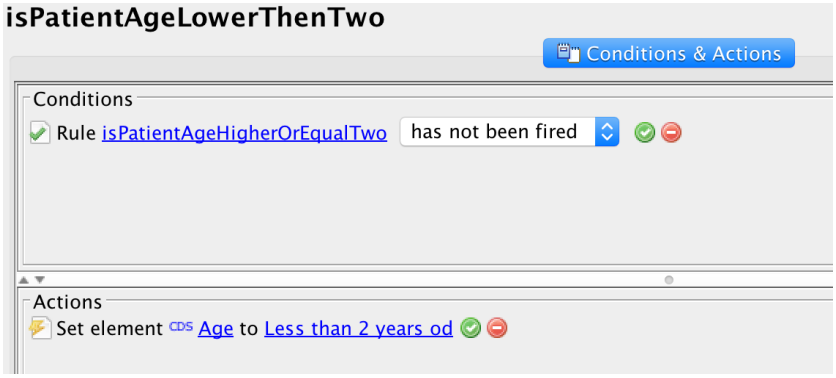
Slika 4.5: Seznam pravil za ugotavljanje anemičnosti pacienta

1. pravilo: "Ali je pacientova starost večja ali enaka dva?" Pravilo pravi: "Če obstaja element Age arhetipa demografskih podatkov tipa EHR, je njegov atribut magnitude večji ali enak dva in je njegov atribut units enak a (leto), je pogoj izpolnjen. Element Age arhetipa povzetka anemije tipa CSD se nastavi vrednost 1 - starejši od dveh let."



Slika 4.6: Ali je pacient starejši od dveh let?

2. pravilo: "Ali je pacientova starost manjša od dveh let?" Pravilo pravi: "Če se prvo pravilo ne zgodi, potem je izpolnjen pogoj. Elementu Age arhetipa povzetka anemije tipa CSD se nastavi vrednost 2 - mlajši od dveh let."



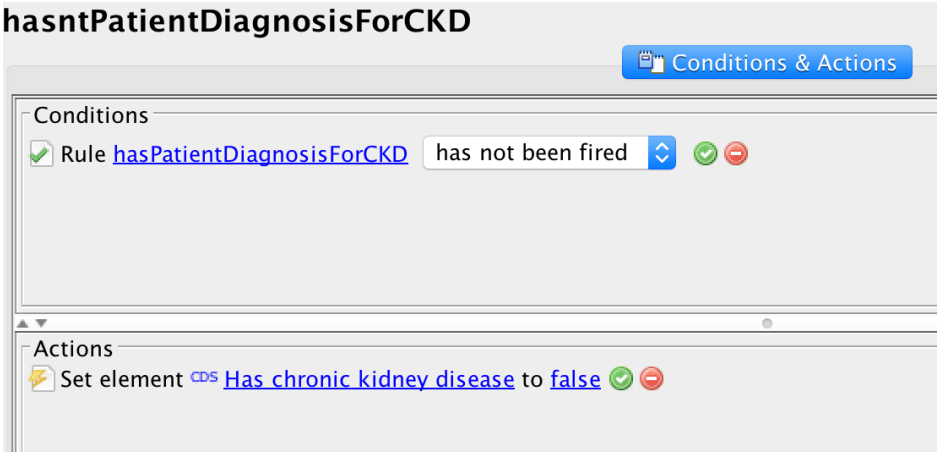
Slika 4.7: Ali je pacient starejši od dveh let?

3. pravilo: "Ali ima pacient diagnozo kronične ledvične bolezni?" Pravilo pravi: "Če je element Problem/Diagnosis name enak kateri od MKB-10 kod diagnoze kronične ledvične bolezni in element obstaja, potem je

pogoj izpolnjen. Elementu Ima kronično ledvično bolezen (angl. Has chronic kidney disease) arhetipa povzetka anemije tipa CSD se nastavi logična vrednost true."

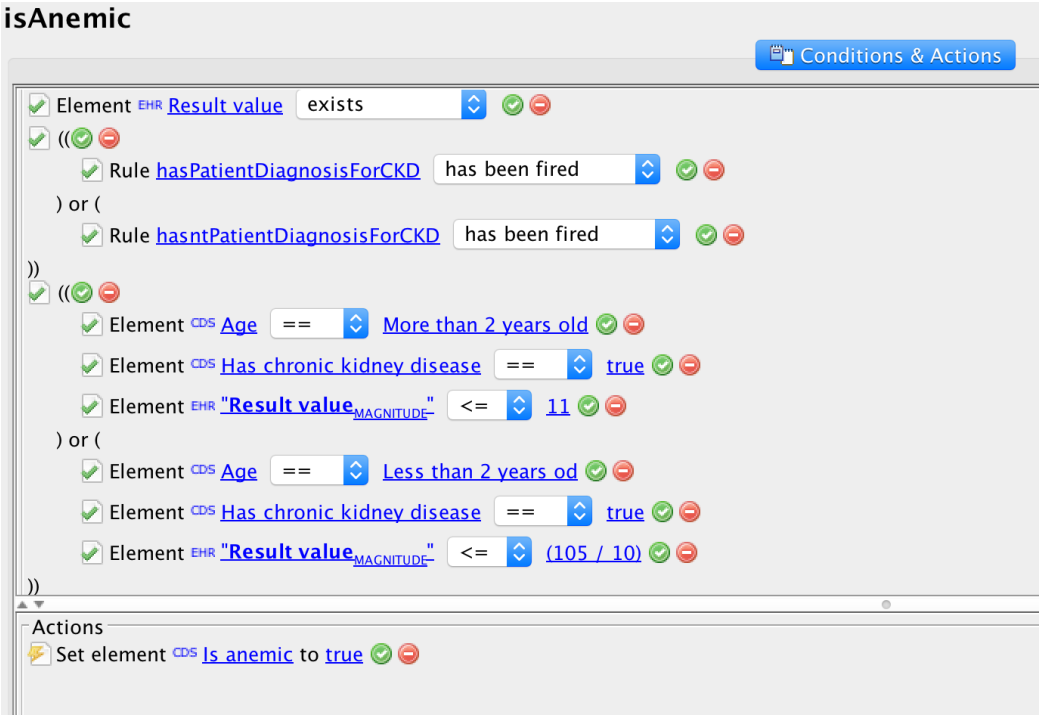
Slika 4.8: Ali ima pacient diagnozo kronične ledvične bolezni?

4. pravilo: "Ali pacient nima diagnoze kronične ledvične bolezni?" Pravilo pravi: "Če se tretje pravilo ne zgodi, potem je pogoj izpolnjen. Elementu Ima kronično ledvično bolezen (angl. Has chronic kidney disease) arhetipa povzetka anemije tipa CSD se nastavi logična vrednost false."



Slika 4.9: Ali pacient nima diagnoze kronične ledvične bolezni?

5. pravilo: "Ali ima pacient anemijo?" Pravilo pravi: "Če se je zgodilo tretje ali četrto pravilo, element Vrednost rezultata (angl. Result value) arhetipa laboratorijskih rezultatov tipa EHR obstaja, ima njegov atribut units vrednost g/dl in je vrednost njegovega atributa magnitude manjša ali enaka 11, element Age arhetipa povzetka anemije tipa CSD enak 1 – starejši od dveh let in elementu Ima kronično ledvično bolezen (angl. Has chronic kidney disease) arhetipa povzetka anemije tipa CSD enaka true ali pa je vrednost atributa magnitude manjša ali enaka 10.5, element Age arhetipa povzetka anemije tipa CSD enak 2 – mlajši od dveh let in elementu Ima kronično ledvično bolezen (angl. Has chronic kidney disease) arhetipa povzetka anemije tipa CSD enak true, potem je pogoj izpolnjen. Elementu Je anemičen (angl. Is anemic) arhetipa povzetka anemije tipa CSD se nastavi logična vrednost true."

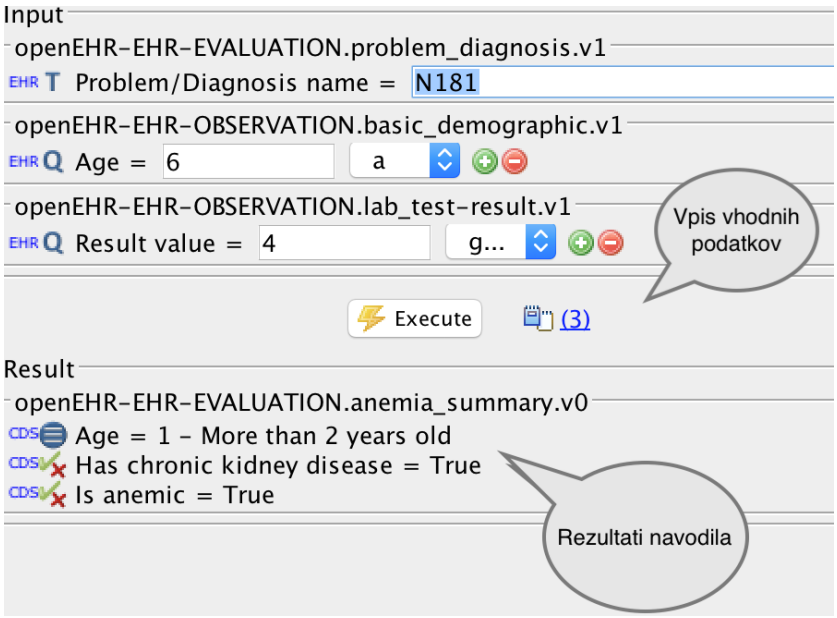


Slika 4.10: Ali je pacient anemičen?

6. pravilo: "Ali pacient nima anemije?" Pravilo pravi: "Če se ni zgodilo peto pravilo, potem je pogoj izpolnjen. Elementu Je anemičen (angl. Is anemic) arhetipa povzetka anemije tipa CSD se nastavi logična vrednost false."

Slika 4.11: Ali pacient ni anemičen?

Po definiciji pravil in metapodatkov nam okno Terminologija (angl. Terminology) prikaže vse kode in njihov opis. S pomočjo gumba Zaženi (angl. Run) lahko navodilo testiramo. S klikom nanj se odpre dialog za vpis testnih vrednosti. Te vpišemo in pritisnemo gumb Začni izvajanje (angl. Execute). Dialog prikaže rezultat navodila.



Slika 4.12: Vpis vhodnih vrednosti in rezultat pravila

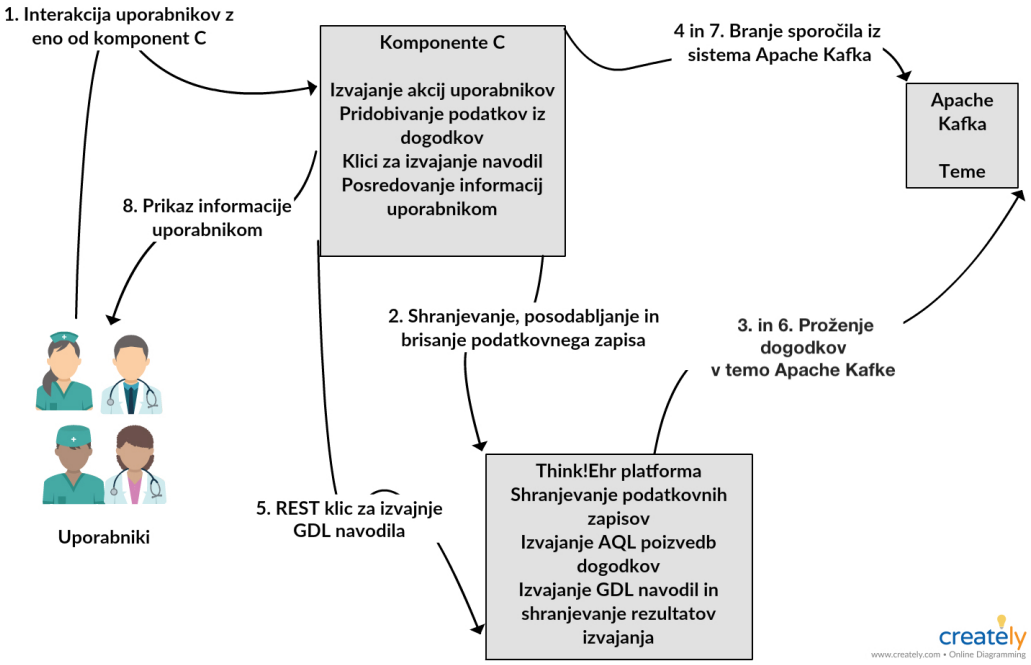
Rezultate lahko preverimo tudi v oknu HTML, ki poda rezultate v HTML obliki. V oknu Graf (angl. Graph) lahko pogledamo diagram poteka navodila in posameznega pravila. V oknu GDL dobimo izvorno kodo GDL pravila, v oknu Drools pa prevedeno kodo GDL navodila v jezik DRL (angl. Drools rule language). Izvorna koda GDL navodila se nahaja v predlogi.

Poglavje 5

Predstavitev prototipa

5.1 Opis

Kot je razvidno iz slike 5.1, je prototip sestavljen iz več komponent: Apache Kafka, Think!EHR Platform™ platforme in prejemnikov rezultatov pravil in obenem sprožilci AQL poizvedb (C). Think!EHR Platform™, Apache Kafka in klinični informacijski sistem Think!Clinical so bile predstavljene v poglavjih 3.3, 3.6 in 3.7. Vsako novo navodilo mora razvijalec dodati na Think!EHR Platform™ platformo in za njegovo implementacijo dodati ustrezno logiko.



Slika 5.1: Prikaz sodelovanja komponent v časovnem sosledju

Ker je v sistemu prisotnih več komponent, se pojavi potreba po definiranju komunikacije med komponentami sistema. Zaradi arhitekture platforme Think!EHR Platform™, kjer je potrebno zagotoviti dostavo podatkov več aplikacijam in potrebe po zagotavljanju hitrosti, zmogljivosti sistema ter zanesljivosti in konsistentnosti podatkov, je najprimernejša arhitektura za komunikacijo med posameznimi komponentami dogodkovna arhitektura in protokol REST.

- Vsak dogodek, ki se kreira v platformi Think!EHR Platform™, mora vračati EHR identifikator pacienta, saj lahko le na podlagi tega ugotovimo, za katerega pacienta se je dogodek prožil.
- Vsak podatkovni zapis vsebuje identifikator ene izmed komponent C, ki je dogodek kreirala.
- Komponente C imajo definirane tipe navodil in z njimi povezano programsko logiko.

- Sporočilo v dogodkih, ki jih kreira in proži Think!EHR Platform™ platforma, so v obliki JSON.
- V primeru neuspešne komunikacije med komponentami do točke predaje dogodka v teme navodil ali napake pri pridobivanju dodatnih podatkov se prepreči shranjevanje podatkovnega zapisa v Think!EHR Platform™ platformo.
- Podatki, ki niso shranjeni v platformi, se pridobijo iz tiste komponente C, ki je dogodek prožila.
- Vsi dogodki, ki niso rezultat navodil in so kreirani s strani platforme Think!EHR Platform™, se pošiljajo v isto imenovano temo Event.
- Na temo Event so prijavljene vse komponente C. Dogodek sprejme le tista, ki ima enak identifikator komponente v EHR identifikatorju pacienta.
- Komponenta C, ki prejme dogodek iz teme Event, nato pridobi potrebne podatke (če so le-ti potrebni) in proži REST klic na Think!EHR Platform™ platformo za izvedbo pravila.
- Vsi dogodki, ki so kreirani s strani platforme Think!EHR Platform™, so tipa Queue.
- Glede na tip faze dogodka tipa Queue se uporablja faza PRE_COMMIT, ki zagotavlja uspešno pošiljanje rezultatov dogodka vsem prejemnikom.
- Vsako definirano navodilo ima svojo temo, na katero se prijavijo komponente, ki želijo prejeti informacijo o rezultatu pravila nad podanimi podatki dogodka.
- Vsako navodilo se po izvedbi shrani v Think!EHR Platform™ platformo.
- Za vsak rezultat navodila je definiran svoj dogodek.
- V dogodku se vedno pošlje celoten rezultat navodila.

- Zadosten pogoj, da se komponenta doda v sistem je, da deluje nad arhitekturo platforme Think!EHR Platform™, kar zagotavlja konsistentnost in prenosljivost podatkov.

- V dogodkih, ki nastanejo kot posledica rezultata navodil, je na voljo EHR identifikator pacienta in cel podatkovni zapis, ki vsebuje tudi identifikator komponente. Na podlagi tega se lahko komponente C odločijo, ali bodo dogodek sprejele ali zavrgle.

- Komponente C lahko sprejmejo tudi dogodke rezultatov navodil, ki niso nastale kot posledica njihove akcije (shranjevanje, urejanje, brisanje). Zavedati se morajo, da nekateri podatki zaradi tega morebiti niso konsistentni.

5.2 Opis delovanja

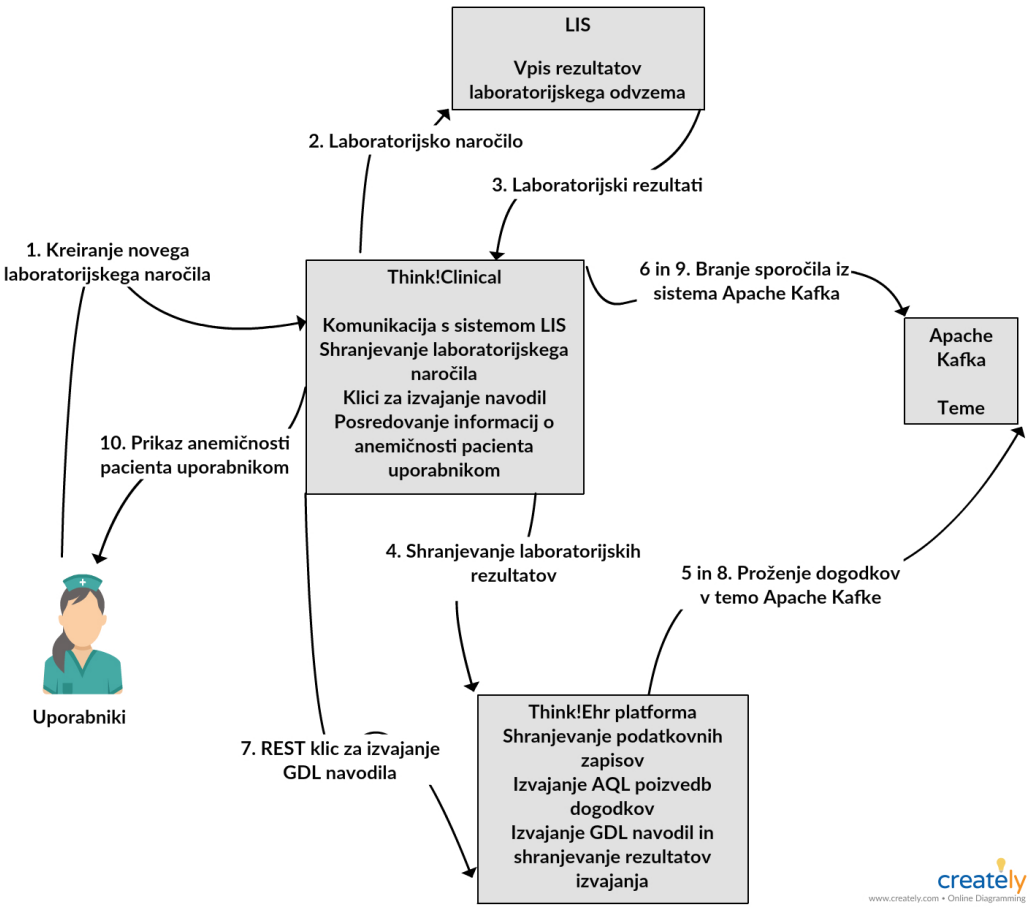
Zaradi lažjega razumevanja bom delovanje opisal v več korakih. Vsak korak izvaja ena od komponent. Koraki so predstavljeni v kronološkem vrstnem redu. Predpostavljeno je, da je v sistemu več komponent, ki delujejo na skupni arhitekturi Think!EHR Platform™ in zadostujejo zgoraj definiranim pogojem. Tipi dogodkov in z njimi povezane poizvedbe ter predloge so že definirane v Think!ERH platformi. Prav tako Think!EHR Platform™ že vsebuje GDL pravila, komponente pa logiko za procesiranje dogodkov in pridobivanje ustreznih podatkov

Postopek se začne, ko ena od komponent C proži AQL poizvedbo, ki dodaja, spreminja ali briše podatkovni zapis. Pri tem se na Think!EHR Platform™ platformi prožijo AQL poizvedbe definiranih dogodkov. V primeru, da poizvedba vrne rezultat, Think!EHR Platform™ proži dogodek Queue na temo Event. Na podlagi podatkovnega zapisa (ta vsebuje atribut identifikatorja komponente) dogodek sprejme tista od komponent C, ki ima enak identifikator. Na podlagi tipa se iz dogodka pridobijo vrednosti, ki so potrebne za GDL navodilo. Pridobijo se za tistega pacienta, ki ima enak EHR

identifikator. Če GDL potrebuje še kakšen drug podatek, ga priskrbi komponenta, ki je prebrala dogodek. Po pridobitvi vseh podatkov se glede na tip dogodka kreira ustrezen objekt tipa JSON, ki vsebuje seznam poti in vrednosti, ki se priredijo posamezni poti. Po kreaciji objekta se izvede REST klic metode POST na platformo Think!EHR Platform™ (vir /guide/guideId/execute/ehrIds/composition/templateId) z logično vrednostjo true atributa persist. Ta izvede zahtevano navodilo nad podatki, ki so bili poslani ter vrne rezultat navodila. Rezultat navodila se shrani tudi v Think!EHR Platform™ platformo. Ker je nad shranitvijo rezultata definirana ustrezna AQL poizvedba, ki dogodek proži, se ta izvede, kreira dogodek in ga pošlje v temo ustreznega navodila. Komponente, prijavljene na posamezno temo, prejmejo dogodek ter na podlagi pridobljenih rezultatov ustrezno posodobijo stanje.

5.3 Princip delovanja na primeru ugotavljanja anemije

Prototip v tem primeru sestoji iz Think!EHR Platform™ platforme, sistema Think!Clinical in sistema Apache!Kafka. Za njega veljajo enake predpostavke, kot so zapisane v predstavitvi prototipa.



Slika 5.2: Primer za ugotavljanje anemičnosti pacienta s sistemom Think!Clinical

Na podlagi zgoraj predstavljenega prototipa, njegovih komponent, definiciji kronične ledvične bolezni in pravil za njeno ugotavljanje je naveden primer, primeren za predstavitev poteka delovanja sistema. Delovanje je predstavljeno po korakih in urejeno v kronološkem vrstnem redu. Dogodek se sproži vsakič ob posodabljanju, brisanju ali shranjevanju podatkovnega zapisa, ki vsebuje hemoglobin. Njegova AQL poizvedba vrne neprazen rezultat. Predpostavljeno je, da je pacient že sprejet v kliničnem informacijskem sistemu Think!Clinical, njegovi laboratorijski izvidi hemoglobina pa imajo normalne vrednosti. Pri odvzemu je raven hemoglobina previsoka. V

platformi Think!EHR Platform™ obstaja AQL poizvedba, ki preverja vsebovanost magnitude hemoglobina. Vsebuje tudi predlogi AnemiaGDL in AnemiaGDLResult ter GDL navodilo za odkrivanje anemije pri bolnikih s kronično ledvično boleznijo. AQL poizvedba se proži vsakič ob shranjevanju novega rezultata navodila. Komunikacija s sistemom LIS (angl. Laboratory Information System) in sistemom Think!Clinical poteka s pomočjo sporočil HL7 verzije 2.

1. Uporabnik oz. uporabnica sistema Think!Clinical vnese novo laboratorijsko naročilo za odvzem hemoglobina izbranega pacienta.

2. Medicinska sestra v laboratoriju naredi odvzem ter v laboratorijski informacijski sistem LIS vnese rezultate.

3. Sistem Think!Clinical prejme rezultate in jih shrani v platformo Think!EHR Platform™.

4. V platformi Think!EHR Platform™ se ob shranitvi, posodobitvi ali brisanju izvede poizvedba AQL, ki preverja vsebovanost magnitude hemoglobina v podatkovnem zapisu.

5. Glede na zgornje predpostavke se pri tem kreira dogodek tipa Queue in se pošlje na temo Event.

6. Ker identifikator komponente, ki je shranjen v podatkovnem zapisu, ustreza identifikatorju sistema Think!Clinical, ta dogodek sprejme.

7. Think!Clinical pridobi podatke o starosti pacienta in kodo diagnoze po standardu MKB-10. Pacienta določi iz EHR identifikatorja dogodka.

8. Think!Clinical kreira seznam poti predloge AnemiaGDL in ji priredi ustrezne vrednosti.

9. Think!Clinical izvede REST klic metode POST na platformo Think!EHR Platform™ z ustreznimi parametri za izvedbo GDL navodila anemičnosti pacienta.

```
select distinct obs, c
from EHR[ehr_id/value=ehrId]
contains Observation c[openEHR-EHR-OBSERVATION.lab_report.v1]
contains Observation obs[openEHR-EHR-OBSERVATION.lab_test.v1]
where
    c/name/value='Laboratory report' AND
    obs/data[at0001]/events[at0002]/data[at0003]/items[at0095]/
        items[at0096]/name/value matches {'K-Hb', 'K-Hemoglobin'}
and
    obs/data[at0001]/events[at0002]/data[at0003]/items[at0095]/
        items[at0096]/items[at0078]/value/units = 'g/L'
```

Slika 5.3: AQL poizvedba, ki preverja raven hemoglobina

10. Think!EHR Platform™ platforma izvede navodilo in vrne rezultate.
11. Ob tem se v platformo shrani rezultat navodila.
12. Izvede se AQL, ki preverja, ali je bil shranjen rezultat navodila.

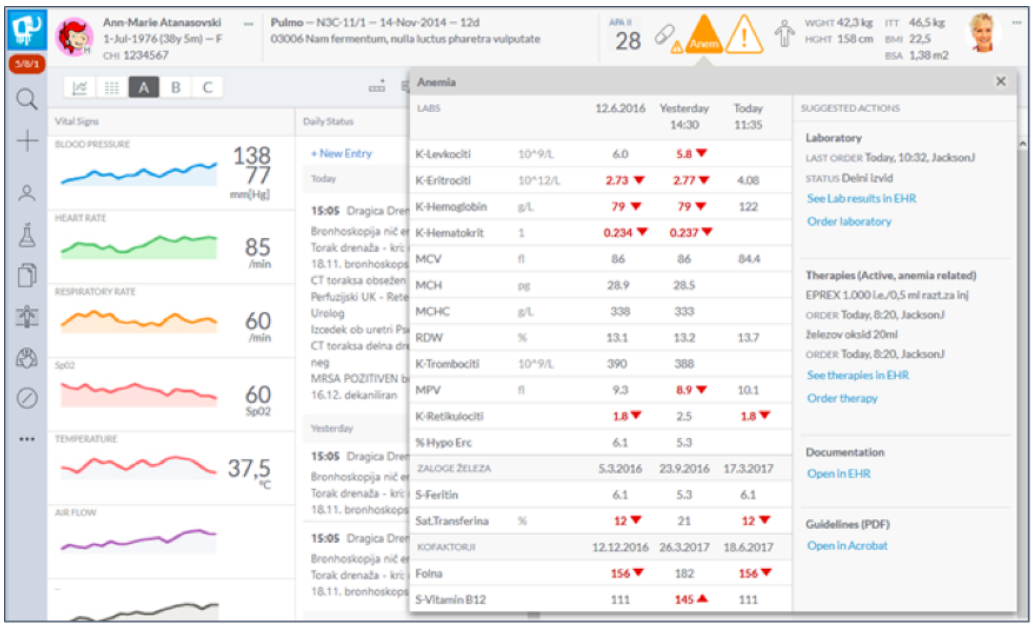
```
select a
from EHR[ehr_id/value=ehrId]
contains COMPOSITION a[openEHR-EHR-EVALUATION.anemia_summary.v0]
where a/name/value='Anemia summary'
```

Slika 5.4: AQL poizvedba, ki preverja, ali je podatkovni zapis rezultat navodila

13. AQL poizvedba vrne neprazen rezultat. Ob tem se kreira ustrezen dogodek in se pošlje na temo navodila.
14. Think!Clinical, ki je kot prejemnik prijavljen na temo navodila, prejme dogodek, z njegovimi podatki pa ustrezno posodobi stanje podatkov v

svojem sistemu.

15. Prejemnik pokaže rezultate spremembe stanja podatkov. Spodaj je podan primer prikaza anemičnosti pacienta v sistemu Think!Clinical.



Slika 5.5: Primer predstavitve anemičnosti pacienta v sistemu Think!Clinical

5.4 Prednosti in slabosti uporabe prototipa

Uporaba zgoraj predstavljenega prototipa in orodij prinaša veliko prednosti in tudi nekaj slabosti. Spremenijo se vloge v procesu razvoja. GDL navodila lahko kreira in posodablja analitik. Programer je v procesu razvoja še vedno nujen, saj je treba implementirati dodatno programsko logiko. S tega vidika se vloge razporedijo, čas razvoja se pospeši, vsebino pa pozna več ljudi, kar je zagotovo prednost. Z uporabo navodil je koda bolj centralizirana in pregledna ter manj kompleksna. Vsa logika izvajanja pravil in pridobivanja kliničnih podatkov je premaknjena na platformo. V kodi se tako ohrani

logika pridobivanja dodatnih podatkov, pridobivanja podatkov iz rezultata in izvajanja REST klicev za izvajanje GDL navodila. Zaradi odvisnosti je takšna koda bolj centralizirana. Ker je vsa kompleksnost izvajanja navodil razvijalcu skrita, je koda sistema bolj intuitivna in pregledna. Kode je tudi manj (pravila in koda za izvajanje pravil se ne nahajata v sistemu). Na primeru anemije je vidno, da so nekatere vrednosti konstantne. Takšnih vrednosti ni treba nastavljati v kodi, saj se nahajajo v izvorni kodi pravila. Razvijalcu ni treba razumeti, kako deluje sistem za izvajanje GDL navodil na platformi, kar je lahko tako prednost kot slabost. Razvijalec lahko razvija sistem dosti hitreje. Po drugi strani pa se zaradi nerazumevanja zanaša na delovanje platforme. Navodila se naložijo na platformo in so dostopna vsem sistemom, ki jo uporabljajo. Če navodilo kreira in razvije en sistem, ga lahko uporabljajo tudi drugi. Vsa navodila se izvajajo na platformi. To pomeni, da ima takšen sistem pri izvajanju navodil enotno točko odpovedi. Če se pojavi napaka, jo je potrebno odpraviti na eni točki. Napako lahko odpravijo samo razvijalci platforme, kar pomeni odvisnost od njih. Razvijalci sistema morajo tako čakati na popravke. Delovanje prototipa je počasnejše, kot če bi se vse izvajalo znotraj enega sistema. Komunikacija med različnimi sistemi, pretvarjanje objektov v obliko JSON in shranjevanje dodatnih podatkovnih zapisov zagotovo zahteva več časovnih virov in virov v obliki procesorske moči in delovnega pomnilnika. Zaradi teh razlogov je delovanje počasnejše. Z vidika komunikacije in več sistemov je možnost za pojavljanje napak dosti večja. Napake se lahko zgodijo med komunikacijo med sistemi ali v primeru nedostopnosti katerega od njih. Apache Kafka in platforma omogočata, da se podatki v primeru napake ponovno pošljejo na prednastavljen časovni interval in njihovo začasno shranjevanje, dokler niso prejeti s strani prejemnika. Glede na zgoraj navedene prednosti in slabosti ugotavljam, da uporaba prototipa za podporo pri odločanju nad standardom openEHR prinaša več prednosti kot slabosti. Čeprav se napake lahko pojavijo pogosteje, posamezne komponente ponujajo ustrezne načine za njihovo odpravljanje. Kljub temu, da je delovanje sistema nekoliko počasnejše, razlike niso tako velike,

da bi prototip deloval počasneje.

Poglavje 6

Zaključek

Hitrejši razvoj tehnologije močno vpliva na vse dele našega življenja, tudi na hitrejšo informatizacijo zdravstva ter razvoj vedno boljših in naprednejših zdravstvenih sistemov. Podatki postajo bolj strukturirani, njihova souporaba in izmenjava pa lažja. Pomemben delež k temu so prispevali številni standardi, kot sta HL7 in openEHR. Težavo pri razvoju predstavljajo vse večji in kompleksnejši informacijski sistemi in večje število poslovnih pravil. Sistemi so implementirani s pomočjo programske kode. Razumljivi so le njihovim razvijalcem. Še večji problem predstavljajo podatki, ki so specifični in razumljivi samo za aplikacijo, ki je ustvarila podatke. Pojavi se vse večja potreba po podpori pri odločanju. Pomembno vlogo pri reševanju teh problemov imajo sistemi za upravljanje poslovnih pravil, ki ponujajo enostaven centraliziran pregled. Omogočajo enostavno vzdrževanje in definiranje novih poslovnih pravil. Standard openEHR je za podporo pri odločanju razvil jezik GDL, ki za izvajanje uporablja sitem za upravljanje poslovnih pravil. V diplomsko delu sem na začetku podal teoretično podlago predstavljenemu prototipu z opisom standardov openEHR in HL7, jezika GDL in predstavil sisteme za upravljanje poslovnih pravil. Nato sem opisal posamezne komponente prototipa ter komunikacijo med njimi. Predstavil sem postopek kreiranja novega navodila in način njegove uporabe v platformi Think!EHR Platform™. Zaradi potrebe prototipa sem predstavil proženje dogodkov znotraj

platforme. V diplomskem delu sem podal primer delovanja na podlagi ugotavljanja anemičnosti pacienta s kronično ledvično boleznijo. Nadaljnje delo vidim predvsem v definiciji novih navodil in odstranjevanju z njimi povezane odvečne programske kode. Največje možnosti se pojavljajo pri implementaciji specifikacije planiranja nalog, kjer je potrebno klinično delo. Z implementacijo modula v platformo Think!EHR Platform™ bo možno shranjevati še večje količine različnih podatkov. Nad njimi bo možno definirati različna nova navodila. S tem se bo kompleksnost sistemov še dodatno zmanjšala, vse večja pa bo tudi podpora pri odločanju.

Poglavje 7

Priloge

7.1 Izvorna koda navodila GDL za določanje anemije pacienta

```
(GUIDE) <
      gdl_version = <"0.1">
      id = <"anemiaGuide">
      concept = <"gt0001">
      language = (LANGUAGE) <
      original_language = <[ISO_639-1::en]>
>
    description = (RESOURCE_DESCRIPTION) <
        details = <
  ["en"] = (RESOURCE_DESCRIPTION_ITEM) <
      keywords = <"anemia",...>
    misuse = <"Only use with KDIGO guidliens">
      purpose = <"Detect anemia for patients with chronic kidney
          diagnosis">
      use = <"Provide patient age, MKB-10 code for diagnosis and
          level of hemoglobin in g/dl.">
  >
 >
    lifecycle_state = <"Initial">
      original_author = <
```

```
    ["date"] = <"15.1.2018">
    ["email"] = <"primoz.delopst@marand.si">
    ["name"] = <"Primoz Delopst">
    ["organisation"] = <"Marand d.o.o">
  >
    other_details = <
    ["references"] = <"http://www.kdigo.org/
        clinical_practice_guidelines/pdf/KDIGO-Anemia%20GL.pdf">
  >
>
    definition = (GUIDE_DEFINITION) <
        archetype_bindings = <
    ["gt0002"] = (ARCHETYPE_BINDING) <
      archetype_id = <"openEHR-EHR-EVALUATION.anemia_summary.v0
          ">
      domain = <"CDS">
      elements = <
      ["gt0010"] = (ELEMENT_BINDING) <
      path = <"/data[at0001]/items[at0003]">
      >
      ["gt0011"] = (ELEMENT_BINDING) <
      path = <"/data[at0001]/items[at0002]">
      >
      ["gt0012"] = (ELEMENT_BINDING) <
      path = <"/data[at0001]/items[at0006]">
      >
    >
  >
    ["gt0004"] = (ARCHETYPE_BINDING) <
      archetype_id = <"openEHR-EHR-EVALUATION.problem_diagnosis.
          v1">
      domain = <"EHR">
      elements = <
      ["gt0009"] = (ELEMENT_BINDING) <
      path = <"/data[at0001]/items[at0002]">
      >
    >
  >
```

```
  ["gt0005"] = (ARCHETYPE_BINDING) <
      archetype_id = <"openEHR-EHR-OBSERVATION.basic_demographic
          .v1">
      domain = <"EHR">
      elements = <
     ["gt0008"] = (ELEMENT_BINDING) <
      path = <"/data[at0001]/events[at0002]/data[at0003]/items[
          at0013]">
     >
    >
   >
  ["gt0006"] = (ARCHETYPE_BINDING) <
      archetype_id = <"openEHR-EHR-OBSERVATION.lab_test-result.
          v1">
      domain = <"EHR">
      elements = <
     ["gt0007"] = (ELEMENT_BINDING) <
      path = <"/data[at0001]/events[at0002]/data[at0003]/items[
          at0095]/items[at0096]/items[at0112]">
     >
    >
   >
 >
    rules = <
  ["gt0013"] = (RULE) <
      when = <"$gt0008.magnitude>=2", "$gt0008.units=='a'", "
          $gt0008!=null">
      then = <"$gt0010=1|local::at0004|More than 2 years old
          |",...>
    priority = <6>
   >
  ["gt0014"] = (RULE) <
      when = <"!fired($gt0013)",...>
    then = <"$gt0010=2|local::at0005|Less than 2 years old
        |",...>
    priority = <5>
   >
  ["gt0015"] = (RULE) <
```

```
      when = <"$gt0009!=null", "(((($gt0009=='N181')||($gt0009=='
          N182'))||(($gt0009=='N183')||($gt0009=='N184')))||(
          $gt0009=='N185')">
      then = <"$gt0012=true",...>
    priority = <4>
  >
  ["gt0016"] = (RULE) <
      when = <"!fired($gt0015)",...>
    then = <"$gt0012=false",...>
    priority = <3>
  >
  ["gt0017"] = (RULE) <
      when = <"$gt0007!=null", "$gt0007.units=='g/dl'", "(fired(
          $gt0015))||(fired($gt0016))", "(($gt0010==1|local::
          at0004|More than 2 years old|)&&(($gt0012==true)&&(
          $gt0007.magnitude<=11)))||(($gt0010==2|local::at0005|
          Less than 2 years od|)&&(($gt0012==true)&&($gt0007.
          magnitude<=(105/10))))">
      then = <"$gt0011=true",...>
    priority = <2>
  >
  ["gt0018"] = (RULE) <
      when = <"!fired($gt0017)",...>
    then = <"$gt0011=false",...>
    priority = <1>
  >
 >
>
    ontology = (GUIDE_ONTOLOGY) <
        term_definitions = <
  ["en"] = (TERM_DEFINITION) <
      terms = <
    ["gt0001"] = (TERM) <
      text = <"Primoz Delopst">
      description = <"Gudiline that calculate if patient has
          anemia for patients with chronic kidney diagnosis">
    >
    ["gt0007"] = (TERM) <
```

```
 text = <"Result value">
 description = <"*">
>
["gt0008"] = (TERM) <
 text = <"Age">
 description = <"Age in years, and for babies: months,
    weeks or days">
>
["gt0009"] = (TERM) <
 text = <"Problem/Diagnosis name">
 description = <"Identification of the problem or diagnosis
    , by name.">
>
["gt0010"] = (TERM) <
 text = <"Age">
 description = <"*">
>
["gt0011"] = (TERM) <
 text = <"Is anemic">
 description = <"*">
>
["gt0012"] = (TERM) <
 text = <"Has chronic kidney disease">
 description = <"*">
>
["gt0013"] = (TERM) <
 text = <"isPatientAgeHigherOrEqualTwo">
>
["gt0014"] = (TERM) <
 text = <"isPatientAgeLowerThenTwo">
>
["gt0015"] = (TERM) <
 text = <"hasPatientDiagnosisForCKD">
>
["gt0016"] = (TERM) <
 text = <"hasntPatientDiagnosisForCKD">
>
["gt0017"] = (TERM) <
```

```
        text = <"isAnemic">
      >
      ["gt0018"] = (TERM) <
        text = <"isntAnemic">
      >
    >
  >
 >
>
```

Literatura

[1] Adl designer. Dosegljivo: https://ehrscape.marand.si/designerv2/. [Dostopano: 15. 11. 2017].

[2] Apache kafka. Dosegljivo: https://hackernoon.com/thorough-introduction-to-apache-kafka-6fbf2989bbc1. [Dostopano: 19. 2. 2018].

[3] Api. Dosegljivo: https://medium.freecodecamp.org/what-is-an-api-in-english-please-b880a3214a82. [Dostopano: 1. 2. 2018].

[4] Api explorer. Dosegljivo: https://www.ehrscape.com/api-explorer.html. [Dostopano: 4. 11. 2017].

[5] Aql. Dosegljivo: https://www.openehr.org/releases/QUERY/latest/docs/AQL/AQL.htmlm. [Dostopano: 2. 11. 2017].

[6] Archetypes. Dosegljivo: https://openehr.atlassian.net/wiki/spaces/resources/pages/4554753/Archetype+FAQs. [Dostopano: 10. 12. 2017].

[7] Brms. Dosegljivo: https://www.progress.com/faqs/corticon-faqs/what-is-a-business-rules-management-system. [Dostopano: 19. 2. 2018].

[8] Brms. Dosegljivo: https://medium.com/@ryanjollyyoung/what-is-a-business-rule-management-system-39ebcb7900c7. [Dostopano: 19. 2. 2018].

[9] Cen. Dosegljivo: https://www.cen.eu/Pages/default.aspx. [Dostopano: 14. 12. 2017].

[10] Clinical knowledge manager. Dosegljivo: https://openehr.atlassian.net/wiki/spaces/healthmod/pages/2949126/Clinical+Knowledge+Manager. [Dostopano: 10. 12. 2017].

[11] Dogodkovno usmerjena arhitektura. Dosegljivo: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven. [Dostopano: 15. 12. 2017].

[12] Dogodkovno usmerjena arhitektura. Dosegljivo: https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch02.html. [Dostopano: 19. 2. 2017].

[13] Drools. Dosegljivo: https://www.drools.org. [Dostopano: 12. 2. 2018].

[14] Drools. Dosegljivo: https://docs.jboss.org/drools/release/6.5.0.Final/drools-docs/html_single/index.html. [Dostopano: 19. 2. 2018].

[15] Ehr in emr. Dosegljivo: https://www.practicefusion.com/blog/ehr-vs-emr/. [Dostopano: 21. 12. 2017].

[16] Emr in ehr. Dosegljivo: https://www.softwareadvice.com/resources/ehr-vs-emr-whats-difference/. [Dostopano: 21. 12. 2017].

[17] Gdl. Dosegljivo: http://www.openehr.org/releases/CDS/latest/docs/GDL/GDL.html. [Dostopano: 15. 12. 2017].

[18] Gdl editor. Dosegljivo: https://sourceforge.net/projects/gdl-editor/. [Dostopano: 1. 12. 2017].

[19] Hl7. Dosegljivo: http://www.ifasystems.de/index.php?view=article&catid=1%3Awhy-ifa&id=79%3Ahl7-healthcare-standards&format=pdf&option=com_content&Itemid=75. [Dostopano: 19. 2. 2018].

[20] Hl7. Dosegljivo: https://www.uef.fi/documents/677096/736588/JQ-2011-05-23_HL7_Finland__HL7_Who_What_and_What_s_New.pdf/c4ebb4c8-ac51-4fda-ab80-0dbd30b2f048. [Dostopano: 19. 2. 2018].

[21] Hl7 standards. Dosegljivo: http://www.hl7.org/implement/standards/index.cfm?ref=common. [Dostopano: 4. 1. 2018].

[22] Introduction to apache kafka. Dosegljivo: https://kafka.apache.org/intro. [Dostopano: 6. 1. 2018].

[23] Introduction to archetypes. Dosegljivo: https://openehr.atlassian.net/wiki/spaces/healthmod/pages/2949191/Introduction+to+Archetypes+and+Archetype+classes. [Dostopano: 14. 11. 2017].

[24] Kdigo. Dosegljivo: http://www.kdigo.org/clinical_practice_guidelines/pdf/KDIGO-Anemia%20GL.pdf. [Dostopano: 12. 2. 2018].

[25] Klb zgibanka. Dosegljivo: https://www.sb-celje.si/media/files/dokumenti/Nasveti_-_ledvicni/KLB_zgibanka-koncna_pdf.pdf. [Dostopano: 20. 12. 2017].

[26] Kronična ledvična bolezen - ledvična anemija. Dosegljivo: http://www.viva.si/Bolezni-secil-in-ledvic-Urologija/2299/. [Dostopano: 20. 12. 2017].

[27] Openehr. Dosegljivo: https://code4health.org/platform/content/openehr. [Dostopano: 15. 11. 2017].

[28] Openehr architecture overview. Dosegljivo: http://www.openehr.org/releases/BASE/latest/docs/architecture_overview/architecture_overview.html. [Dostopano: 15. 11. 2017].

[29] Openehr origins. Dosegljivo: http://www.openehr.org/about/origins. [Dostopano: 15. 11. 2017].

[30] Rate algorithem. Dosegljivo: https://www.sparklinglogic.com/rete-algorithm-demystified-part-2/. [Dostopano: 19. 2. 2018].

[31] Rateoo algorithem. Dosegljivo: https://salaboy.com/2011/06/06/drools-reteoo-for-dummies-1-intro/. [Dostopano: 19. 2. 2018].

[32] Representational state transfer. Dosegljivo: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm. [Dostopano: 17. 1. 2018].

[33] Rest. Dosegljivo: http://searchmicroservices.techtarget.com/definition/REST-representational-state-transferx. [Dostopano: 19. 2. 2018].

[34] Think!ehr platform™. Dosegljivo: http://www.marand-think.com. [Dostopano: 13. 11. 2017].

[35] Web services. Dosegljivo: https://www.tutorialspoint.com/webservices/what_are_web_services.htm. [Dostopano: 1. 2. 2018].

[36] What is an electronic health record. Dosegljivo: https://www.healthit.gov/providers-professionals/faqs/what-electronic-health-record-ehr. [Dostopano: 3. 11. 2017].

[37] Working with templates. Dosegljivo: http://www.openehr.org/downloads/ADLworkbench/working_with_templatesl. [Dostopano: 15. 11. 2017].

[38] K-P Adlassnig, Bernd Blobel, and John Mantas. *Medical Informatics in a United and Healthy Europe: Proceedings of MIE 2009*, volume 150. IOS Press, 2009.

[39] Nadim Anani, Rong Chen, Tiago Prazeres Moreira, and Sabine Koch. Retrospective checking of compliance with practice guidelines for acute stroke care: a novel experiment using openehr's guideline definition language. *BMC Medical Informatics and Decision Making*, 14(1):39, May 2014.

[40] Nishant Garg. *Apache Kafka*. Packt Publishing Ltd, 2013.

[41] Dipak Kalra, Thomas Beale, and Sam Heard. The openehr foundation. *Studies in health technology and informatics*, 115:153–173, 2005.

[42] Wooshik Kim, Suyoung Lim, Jinsoo Ahn, Jiyoung Nah, and Namhyun Kim. Integration of ieee 1451 and hl7 exchanging information for patients' sensor data. *Journal of Medical Systems*, 34(6):1033–1041, Dec 2010.

[43] Peter Schloeffel, Thomas Beale, George Hayworth, Sam Heard, and Heather Leslie. The relationship between cen 13606, hl7, and openehr. 01 2006.

[44] Pradeep K Sinha, Gaur Sunder, Prashant Bendale, Manisha Mantri, and Atreya Dande. *Electronic health record: standards, coding systems, frameworks, and infrastructures*. John Wiley & Sons, 2012.