TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Mait Sarv

Programmeerimine rakenduses Raintree ja keeles RSL

Bakalaureusetöö (6 EAP)

Juhendaja(d): Tõnu Tamme

Tartu 2016

Programmeerimine rakenduses Raintree ja keeles RSL

Lühikokkuvõte:

Töös antakse ülevaade programmeerimisest rakenduses Raintree RSL keele baasil. Töö on õppematerjaliks, millega on kaasas harjutusülesanded ja õppevideod.

Võtmesõnad:

RSL, Raintree, õppevahend

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria).

Programming in Raintree using RSL language

Abstract:

This paper gives an overview on how to program is Raintree using Raintree Scripting Language (RSL). The paper is a study material that includes programming exercises and videos.

Keywords:

RSL, Raintree, study material

CERCS: P170 Computer science, numerical analysis, systems, control.

Sisukord

1. Sissejuhatus

Raintree on haiglate juhtimissüsteem, mida arendatakse ettevõttes Raintree Systems. Ettevõte tegutseb vaid Põhja-Ameerika turul, kuid suur osa arendajatest asuvad Eestis. Raintree võimaldab hallata patsientide visiite, haiguslugusid ja arveid. Samuti on võimalik genereerida raporteid salvestatud andmetest. Arhitektuuriliselt on Raintree klient-server rakendus, kus kliendi poole funktsionaalsus on realiseeritud Raintree kliendis ja enamus serveri poolsest funktsionaalsusest Raintree agendis. Põhiliselt hoitakse andmeid serveri poolel MySQL andmebaasis. RSL ehk *Raintree Scripting Language* on agendi poolt interpreteeritav domeenispetsiifiline programmeerimiskeel, mille abil saab Raintree põhiprogrammi kergesti funktsionaalsusi lisada. RSLile pandi alus 1983. aastal ning keele arendus on toimunud läbi aastate vastavalt vajadusele. RSL koodile tehakse väga vähe süntaksi kontrolli, programmeerija saab takistusteta serverisse lisada süntaktiliselt ebakorrektset koodi. Seetõttu peab programmeerija aru saama instruktsioonide tähendusest. Samuti esineb probleemina RSL koodi täitmise kiirus, mis tihti tuleneb valest tehnikast või disainist, mida on kasutatud valminud koodis.

Töö eesmärgiks on alustada kursuse materjalidega, mis õpetaksid programmeerimist RSL keele põhjal. Kursuses tutvustatakse Raintree ja RSLi eripärasid. Lisamaterjalidena tööle valmivad õppevideod ja harjutusülesanded, mis omakorda jagunevad küsimusteks ja programmeerimisülesanneteks. Materjalid võiksid abiks olla Raintree Systems töötajate koolitamisel. Samuti võiksid materjale kasutada programmeerijad, kes on haiglate poolt eraldi palgatud. Kursuse läbijalt eeldatakse SQL(*Structured Query Language*) süntaksi tundmist ja kokkupuudet mõne SQLi kasutava süsteemiga. Lisaks tuleb kasuks mõne imperatiivse programmeerimiskeele tundmine. Kursuse läbijale võimaldatakse ka ligipääs Raintree keskkonda, kus on võimalik harjutusülesandeid sooritada.

Esialgne kursuse teemade plaanitav ülesehitus on järgnev.

- Raintree arhitektuur
    - Rakenduse ülesehitus
    - MySQL andmebaasi ülesehitus
        - Baastabelid
        - Kohandatavad tabelid
    - Töövahendid
- RSLi sissejuhatus
    - Domeeni spetsiifilisus
    - Kontekstipõhisus
    - Modulaarsus
- RSLi süntaksi tutvustus
    - Andmetüübid ja muutujad
    - If tingimuslause
    - Tsüklid
    - Sisse ehitatud funktsioonid, käsud muutujad
    - Konteksti seadmine käskudega
    - Sisendpunktid
- Raintreega seotud abistruktuurid ja definitsioonid
    - RTM

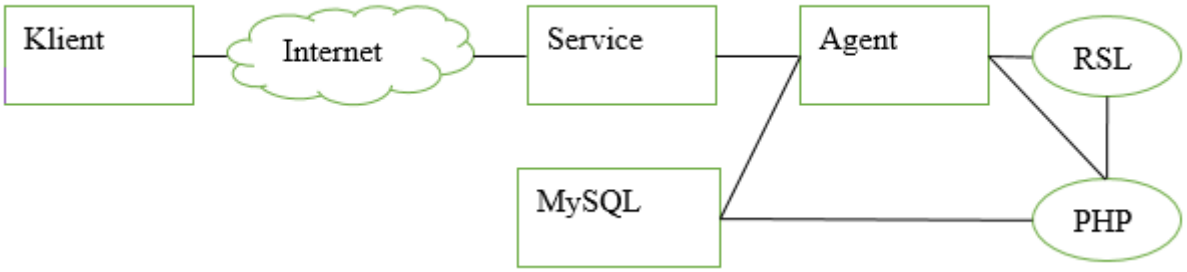
    - Menüüd
    - EMR *template*
    - EMR List
    - Narratiivid
    - RTO
    - Picklist
- Optimisatsioonid
    - loadRecordToVariables ja saveVariablesToRecord
    - Showquery
    - Native Array vs RSL Array
- Raintree alamdomeenid
    - Scheduler
    - EMR
    - Ledger
    - Raportid
    - Faili manipulatsioonid
- MySQLiga töötamine
    - Kuidas MySQL päringuid töötleb
    - Indeksi valimine
    - Suurte päringute jooksutamine indeksi peal
- Koodi kirjutamise head tavad/juhised

Lõputöö vastab kursuse struktuurile. Teemadest on täielkult kaetud kolm esimest: Raintree arhitektuur, RSLi sissejuhatus ja RSLi süntaksi tutvustus. Neljandast teemast on kirjeldatud RTMid, menüüd, EMR *template*'id, EMR listid ja narratiivid.

2. Raintree arhitektuur

2.1. Klient-server arhitektuur

Raintree kasutab klient-server arhitektuuri. Kliendi poole komponenti kutsutakse Raintree kliendiks. Serveri pool koosneb kolmest komponendist: *service*, agent ja MySQL. Raintree kliendi ülesandeks on saata agendile päringuid ning vastuseid kasutajaliidese kaudu kasutajale vahendada. Samuti on klienti sisse ehitatud Raintrees arendamiseks vajalikke tööriistu. Agendis on realiseeritud enamus Raintree funktsionaalsusest. Lisaks interpreteerib agent Raintree Skriptimiskeelt (ingl. k. Raintree Scripting Language, RSL) ja vajadusel kutsub välja PHP-d. RSL võimaldab agendi funktsionaalsust täiendada ja teataval määral ka agendi käitumist juhtida. MySQL-i kasutatakse andmete talletamiseks ning sellega suhtleb põhiliselt agent, kuid teatud määral ka PHP. Raintree *service* vahendab suhtlust kliendi ja agendi vahel. Kui klient ühendub Raintree serveriga, siis kõigepealt luuakse ühendus *service*'iga ning seejärel käivitab *service* iga kliendi sessiooni kohta serveris uue agendi. Joonisel 1 on välja toodud Raintree klient-server arhitektuuri mudel.



Joonis 1. *Raintree rakenduse lihtsustatud mudel.*

2.2. Failisüsteemi ülesehitus

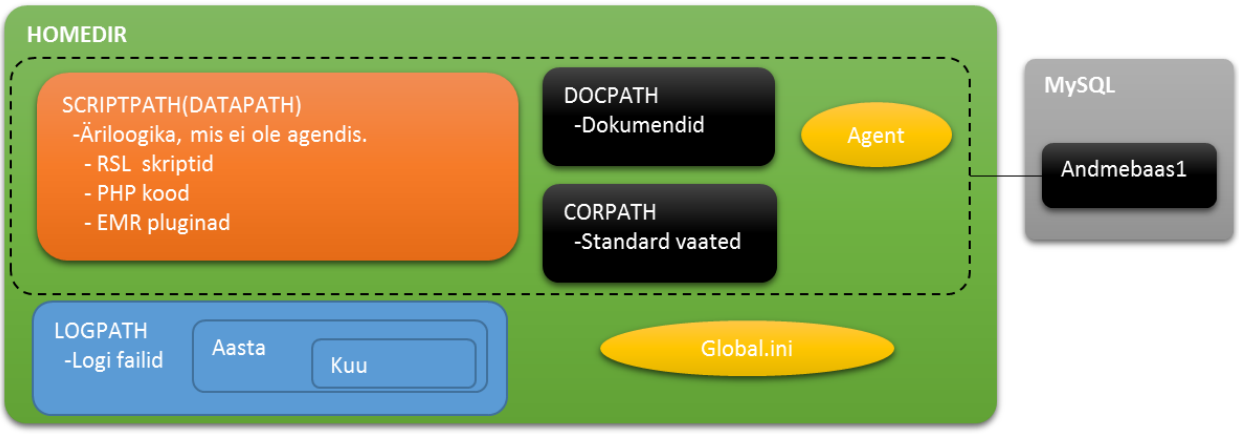
Raintree üles seadmiseks serverisse on vaja kõigepealt installeerida MySQL ja Raintree *service*. Seejärel saab installeerida Raintree eksemplari ehk kodukataloogi (ingl. k. *Homedir*). Otse kodukataloogis asub Raintree agent ja ka peamine konfiguratsioonifail global.ini, kus on näiteks defineeritud parameetrid ühendamiseks MySQL andmebaasiga. Alamkataloogina eksisteerib kindlasti logide kaust nimega log.

Kodukataloogis peab asetsema vähemalt üks andmebaasikataloog (ingl. k. *Scriptpath*), mis vaikimisi kannab nime dat. Selles kaustas asub andmebaasi jaoks vajalik äriloogika, mis ei ole realiseeritud agendis. Inglise keeles öeldakse kausta kohta ka *datapath*, mõiste pärineb MySQL eelsetest versioonidest, kui kaustas asusid ka ISAM tüüpi andmetabelite failid. Tähtsamad alamkaustad andmebaasikataloogis on:

corelib – Raintree PHP standardteegid.
emr – RSL-is realiseeritud moodleid ehk EMR pluginad.
menu – menüüde definitsoonid XML formaadis.
platform – PHP moodulid.
reports – RSL skriptid, mis väljastavad raporteid.
rules – RSL skriptid mida kutsutakse välja teatud kasutaja tegevuste peale(iganenud)

scripts – mitmesugused RSL skriptid.

Andmebaasiga seotud on ka *docpath*, kus asuvad prinditavate dokumentide kirjeldused, ja *corpath*, kus asuvad agendi jaoks vajalike vaadete definitsioonid. Need kaks kausta asuvad vaikimisi otse kodukataloogis. Kataloogid, kus asuvad andmebaasikaust, *docpath* kui ka *corpath*, on defineeritud MySQL andmebaasi tabelis. Andmebaasid võivad neid katalooge ka jagada. Joonisel 2 on välja toodud kodukataloogi tähtsamad komponendid.



Joonis 2. *Failisüsteemi lihtsustatud mudel.*

2.3. MySQL struktuur

Üks MySQL installatsioon võib sisaldada andmebaase erinevatest Raintree eksemplarist. Andmebaasi nimi koosneb alati kahest osast, eraldatuna alakriipsuga. Esimese osa väärtus tuleb global.ini faili paremeetritest, seega seob see andmebaasid antud kodukataloogiga. Teine osa eristab andmebaase Raintree siseselt. Näiteks raintree_dat ja raintree_test on korrektsed andmebaasi nimed.

Üldiselt jagunevad andmebaasi sees tabelid kaheks: baastabelid, mille struktuuri määrab agent, ning kohandatavad tabelid, mille struktuuri ehitab üles agent vastavate andmebaasi-kataloogis asuvate definitsioonide põhjal. Lisaks leidub veel tabelid, mis on loodud RSL skriptiga või PHPs.

2.3.1. Kohandatavad tabelid

Kohandatavate tabelite nimed sisaldavad keskel alati vähemalt ühte alakriipsu. Alakriipsule eelneva osa nimetuseks on *usefile* ja järgnev osa on *screen*. Tabelite struktuuride definitsioonid asuvad failides laiendiga rtm, ühtlasi defineerivad need ka vaate graafilise kasutajaliidese kaudu[Lisa]. *Screen* väärtuseks tabeli tekitamisel või uuendamisel saab faili nimi. *Usefile* on agendi muudetav kontekst, vaikimisi väärtusega ‘PMCAT’. Kui kasutajale näidata mingit vaadet ja tabelit selle kohta ei eksisteeri, siis agent teeb taustal tabeli valmis. Kuna usefile on muudetav, siis võib sama vaate põhjal tekitada mitu tabelit. Näiteks kuvades kasutajale vaadet failist aken.rtm ja selle hetkel on usefile muutmata, siis vaatele vastav tabeli nimeks saab pmcat_aken. Kohandavatele tabelitele lisatakse alati kaks veergu, mida ei tohi vaates defineerida: primaarvõti `_id` ja 11 märgiline unikaalne indeks `_pn`.

2.3.2. Baastabelid

Baastabelite struktuuri ei ole võimalik Raintree pakutavate tööriistade abil muuta. Struktuuri käsitsi muutmine võib lõppeda agendi jooksmisaegse veaga või põhjustada andmevigasid. Raintree struktuuri defineerivaid tabelid on kaks. Tabelis `_parameters` hoitakse andmebaasikausta, *docpath*-i ja *corpath*-i asukohta suhtes kodukataloogiga. Tabelis `blob_location` hoitakse asukohti kuhu on salvestatud failidena kirjed, mis on MySQL andmebaasis hoidmiseks liialt suured. Ülejäänud baastabelid on kasutajate sisestatud andmete talletamiseks. Näiteks tabel `doctor`, kus hoitakse raviarstide ja abipersonali kohta informatsiooni. Paljudel baastabelitel on ka omapärane struktuur, näiteks komposiit-indeksite jaoks esineb eraldi indekseeritud veerg, kus hoitaks soovitud väljade andmeid koos. Nende veergude nimed algavad sõnega "_index" ja väärtusi uuendavad trigerid, mis on defineeritud lisamise ja muutmise käskude jaoks. Antud eripärasus tuleneb sellest, kuidas võeti kasutusele MySQL. Vanadest andmebaasi mudelitest pärinevad indeksid ja andmed liideti kokku üheks tabeliks ja indeksi veerud tehti MySQL-is indeksiteks.

3. RSLi kirjeldus

RSL on Raintree agendi poolt interpreteeritav kõrgema taseme programmeerimiskeel, mis võimaldab täiendada ja realiseerida Raintree pakutavaid funktsionaalsusi. RSLil on kõik omadused olemaks domeenispetsiifiline programmeerimiskeel [1]. Lisaks toetab RSL veel imperatiivset, modulaarset ja sündmuspõhist paradigmat.

3.1. Sündmuspõhisus

Sündmuspõhise paradigma toetamine tähendab, et RSLil on palju võimalikke sisendpunkte. Agent võimaldab interaktiivses keskkonnas erinevate kasutaja tegevuste puhul välja kutsuda RSL koodi. Üldjuhul on sisendpunktiks RSLi funktsioon, kuid teatud juhtudel võib selleks olla ka ette antud faili esimene rida.

3.2. Modulaarsus

RSL-i modulaarsus on implementeeritud läbi EMR pluginate. Plugin grupeerib kokku RSL faile ja abidefinitsioone: vaated, listid, narratiivid. Igat faili saab adresseerida kasutades plugina koodi ja faili nime. Näiteks "TEST:hello.frm" viitab failile hello.frm, mis asub pluginas nimega TEST. Failisüsteemi poolel luuakse iga EMR plugina jaoks alamkaust emr kataloogi. Skripti faile on võimalik uses võtmesõna abil kokku liita. Kui failid asuvad samas kaustas, siis piisab kui uses käsule antakse parameetriks faili nimi. Erinevatest pluginatest failide liitmiseks tuleks ette anda ka plugina kood.

3.3. Kontekst

RSL-i on sisse ehitatud mitmeid globaalseid süsteemimuutujaid, -käske ja –funktsioone, kus igaühte neist võib klassifitseerida teatud aladomeeni. Süsteemimuutujate väärtuseid võib RSLis küsida, kuid nende väärtuseid otseselt muuta ei saa. Süsteemimuutujate väärtus sõltub programmi täitmisaegsest kontekstist antud domeenis. Kui teatud muutuja jaoks ei ole konteksti üles seatud, siis muutujal väärtus puudub(kuid see ei tähenda täitmisaegset viga). Käskudega on võimalik anda agendile täitmiseks ülesandeid, seal juures on võimalik muuta konteksti. Süsteemifunktsioonidel on kaks eesmärki. Esiteks need võimaldavad koodi lühemalt kirjutada, sest vajalikud instruktsioonid on juba implementeeritud agendis. Teiseks võimaldab see võita arvutusaegades, juhul kui RSLi interpreteerimise kiirus võib saada probleemiks.

Kui kasutaja navigeerib Raintrees interaktiivses keskkonnas, siis agent seab automaatselt üles kontekste, mis praktikas väljendub agendi poolt ette antud muutujate väärtustamises ja määramises kuidas mõned eeldefineeritud käsud ja funktsioonid töötavad. Korraga võibolla üles seatud mitu erinevat konteksti. Näiteks kui kasutajal on süsteemis avatud patsiendi kindlustuse kirje, siis ollakse korraga nii patsiendi kui ka kindlustuse kontekstis. Teatud kontekstide puhul agent eeldab, et skript lõpetab samas kontekstis, millega ta alustas. Näiteks kui kasutajal on avatud mõni kirje, kus teatud välja muutmise peale kutsutakse välja RSL funktsioon, siis funktsiooni lõppedes tuleb olla positsioneeritud samale kirjele, vastasel juhul võib tekkida andmevead. Teatud kontekstide hoidmiseks kasutatakse ka magasin andmetüüpe. Aktiivset konteksti on võimalik RSL käsuga panna magasini ja seda siis hiljem magasinist välja võtta.

4. RSLi süntaks

RSL skripti fail võib alata kohe RSL instruktsioonidega. Sellidel juhul hakkab agent instruktsioone faili algusest rea haaval täitma ja lõpetab kuni jõuab *end.* märksõnani. Pärast *end.* märksõna võib failis esineda kasutaja defineeritud funktsioone. Kui fail koosneb ainult kasutaja defineeritud funktsioonidest, siis võib *end.* käsu faili alguses ära jätta. Instruktsiooni lõpuks loetakse reavahetust või *;* märki. RSL ei ole tõstutundlik keel.

Kommentaare on RSLis kolme tüüpi. Sõnedega *rem* ja *//* algavad üherealised kommentaarid. Märksõna rem peab alati asetsema rea alguses, *//* võib asetseda ka rea lõpus. Mitme realise kommentaari alguseks on /* ja lõpuks */.

4.1. Muutujad

Muutujaid, mida on võimalik kasutada RSL koodis võib jaotada neljaks: Globaalsed süsteemimuutujad (kutsutakse ka agendimuutujateks), kohandavate tüüpkirjete muutujad, RSL muutujad ja varamumuutujad. Muutujate andmetüüpe defineerima ei pea ning need võivad olla järgnevatest tüüpidest:

- täisarvud,
- ujukomaarvud,
- sõned,
- tõeväärtus tüüp.

Agendimuutujad koosnevad ainult tähtedest ja numbritest. Mõned üksikud nendest on konstandid, näiteks muutuja *add* väärtuseks on alati 3. Suur osa agendimuutujatest on konteksti tundlikud, nende väärtus RSLi interpreteerimise käigus saab muutuda ainult siis, kui muutub kontekst. Mõnede muutujate väärtused püsivad agendi töötamise ajal alati samad, näiteks muutuja *homedir*, mille väärtuseks on kodukataloogi asukoht failisüsteemis, püsib muutumatuna Raintree installatsiooni jaoks. Paljud muutujad aga võivad omandad agendi tööaja ja RSLi interpreteerimise käigus erinevaid väärtusi. Näiteks muutuja *dfirst* väärtuseks on arsti, kellele ollakse hetkel positsioneeritud, eesnimi. Antud muutuja väärtust uuendatakse, kui positsioneeritakse uuele arstile või kui muudetakse arsti nime ja värskendatakse konteksti.

Kohandatavate tüüpkirjete muutujad on seotud kirjetega andmebaasis. Nende väärtus sõltub samuti kontekstist, kusjuures korraga saab aktiivne olla ainult üks kontekst. Väärtustamine toimub *set* võtmesõna abil. Muutujad algavad alati & märgiga ning sisaldavad keskel koolonit, mis eraldab kahte sõne:

```
//konteksti seadmine
usefile "ANDMEFAIL"
index "INDEKS1"

set &kirje1:muutuja1 = "SISU"
set &kirje2:muutuja1 = "SISU2"
```

Antud näites *usefile* käsu peale otsib agent üles kõik tabelid, mis algavad sõnega “ANDMEFAIL_” ja *index* käsu peale tõmbab mällu kõikidest leitud tabelitest kirjed mille võtmeks (_pn veeru väärtuseks MySQLis) on “INDEX1”. Lõpuks antakse “muutuja1”

veeru väärtusteks tabelites “ANDMEFAIL_KIRJE1” ja “ANDMEFAIL_KIRJE2” vastavalt “SISU” ja “SISU2”. Kui set käsu ajal vastavat tabelit ei eksisteeri, siis see luuakse. Kui veergu nimega “muutuja1” ei eksisteeri, siis lisatakse väärtus koos muutuja nimega veergu “_myvariabledata”.

Skriptimuutujad ehk RSL muutujad algavad alati sümboliga % ja võivad sisaldada tähti, numbreid ja alakriipsu. Muutujate väärtustamine toimub käsuga *set*. Muutujaid ei ole vaja eraldi deklareerida, mälu eraldus toimub kui muutujat väärtustatakse. RSL funktsioonide sees kasutusele võetud muutujad on lokaalsed, väljaspool funktsiooni sama muutuja väärtust ei oma. Kõik muutujad, mis on kasutusele võetud väljaspool funktsioone, on globaalsed. Lisaks on %. sümbolitega algavad muutujad globaalsed.

```
set %a = "Hello "
// %.b On alati globaalne muutuja
set %.b = "world!
set %a := %a . %.b
```

Näites väärtustatakse muutujad %a ja %.b ning lõpuks pannakse kaks sõne kokku ja %a väärtuseks saab “Hello world!”. Punkt võrdusmärgist paremal pool on sõnede kokku liimise operaator. Kui võrdusmärgist paremal pool asetseb avaldis, siis tuleb muutujat väärtustada := sümboleid kasutades, sama kehtib ka kohandatavate tüüpkirjete muutujate puhul.

Varamumuutujaid hoitakse mälus seni, kuni agent töötab või kuni need skriptis kustutatakse. Erinevus globaalsetest RSL muutujatest seisneb selles, et RSL muutujaid hoitakse vaid seni, kuni RSL jookseb, skripti lõppedes kõik muutujad kustutatakse. Agendi tööaja jooksul pannakse üldiselt tööle palju RSL skripte ning neil kõigil on oma muutujad.

```
addtorepository "tere" "hommikust"
set %a := GETFROMREPOSITORY("tere")
clearfromrepository "tere"
```

Näiteks lisatakse käsuga varamusse muutuja nimega “tere”. Seejärel võetakse väärtus “hommikust” varamust ja omistatakse RSL muutujale %a. Lõpuks kustutatakse muutuja varamust.

4.2. Funktsioonid

Kasutaja defineeritud funktsioonid RSLis sarnanevad pigem protseduuridele, nad ei tagasta mitte midagi, seega ei saa funktsiooni väärtust muutujale omistada. Funktsiooni alguseks loetakse võtmesõna *func* või *localfunc*, millele järgneb funktsiooni nimi ja sulgude vahel parameetrid, funktsiooni lõpuks loetakse võtmesõna *endfunc*. Funktsioone kutsutakse välja *call* võtesõnaga. Kuigi funktsioon ise midagi ei tagasta, on siiski võimalik parameetrina ette antud muutujate väärtusi tagastada, selleks tuleb funktsiooni definitsioonis parameetri ette kirjutada võtmesõna *var*. Funktsioonid, mis on defineeritud sõnega *localfunc* on nähtavad vaid failis, kus nad asuvad. Sõnega *func* defineeritud funktsioonid on nähtavad kõikides failides, selleks tuleb eelnevalt *uses* käsuga vajalik fail defineerida.

```
set %a := 0
set %b := 0
```

```
call tagastaSuvaline(%a,%b)
//%a väärtuseks jääb 0, %b väärtus on suvaline arv vahemikus 1-9.
end.

localfunc tagastaSuvaline(%esimene, var %teine)
 set %esimene := rand(1,9)
 set %teine := rand(1,9)
endfunc
```

Näites on *rand* eeldefineeritud funktsioon ehk agendi funktsioon, mille puhul ei kasutata *call* võtmesõna. Nende väärtusi saab ka otse muutujatele omistada.

4.3. Tingimuslaused

4.3.1. Kahene valik

Kahese valiku ehk if-lause jaoks on RSLis mitmeid vorme. Kõige lihtsam üherealine vorm on:

```
if <tingimus> then <tegevus>
```

Kui tegevus tõese if-lause korral on mitme realine tuleb kasutada sellist süntaksit:

```
if <tingimus> then
begin
 <tegevus>
end
```

Oluline on, et võtmesõnad begin ja end asuksid eraldi ridadel. Kui on vaja teha tegevusi ka väära tingimuse korral tuleks kasutada järgmiseid lauseid:

```
if <tingimus> then <tegevus>
else <tegevus>

if <tingimus> then <tegevus>
elsebegin
 <tegevus>
end
```

Tingimust on võimalik esitada kahel viisil. See võib asetseda sulgude vahel või sulud võivad puududa. Sulge tuleb kasutada, kui tingimuseks on mõni väärtust tagastav agendi funktsioon. Kui sulgude vahel tingimus on täisarvutüüpi, siis see loetakse tõeseks nullist suuremate väärtuste korral. Sõne tüübi korral loetakse vääraks kõik sõned, välja arvatud „true“. Kui tingimuse ümber ei ole sulge, siis loetakse tõeseks vaid tõeväärtustüüpi väärtusega *true*. Tingimuses saab kasutada ka võrdlustehted =,>=,<=,<>. Kui arv on esitatud tingimuses sõnena, siis see alati teisendatakse numbriks. Kui sulud tingimuse ümber puuduvad, siis erandjuhuna võrreldakse teatud kujuga sõnesid kui kuupäevi. Nii on võimalik võrrelda kuupäeva formaati, kus kuu on kõige esimene. Näiteks võrreldes sõnesid “01-20-16” ja “02-02-14” kuupäevana oleks esimene suurem kui teine, sõnena võrreldes oleks teine esimene kui suurem.

RSLis on ka olemas *if* funktsioon, millel on kolm parameetrit. Esimeseks parameetriks on tingimuslause, mida interpreteeritakse samasuguselt kui tavalise if-lause tingimust, mis asetseb sulgude vahel. Teiseks parameetriks on avaldis, mida täidetakse tõese väärtuse korral, kolmandaks parameetriks on avaldis, mida täidetakse muudel juhtudel:

```
//%a väärtuseks saab TRUE
set %a := if("1" = 1,TRUE,FALSE)
```

4.3.2. Mitmene valik

Kui tingimusel võib olla rohkem kui kaks väärtust on võimalik kasutada ka *switch* lauset:

```
switch (%a)
 case (1)  set %b = 1
 case (2)
    set %b = 1
    set %c = 2
 case(otherwise) set %b = 0
endswitch
```

Kui %a väärtus näites on 1, siis saab %b väärtuseks 1. Kui %a väärtus on 2, siis saab %b väärtuseks 1 ja %c väärtuseks 2. Kõikidel muudel juhtudel saab %b väärtuseks 0. Vaikimisi tegevust *otherwise* täidetakse vaid kui eelnevalt ei ole ühtegi vastet leitud ning selle võib ka ära jätta.

4.4. Tsüklid

Erinevaid tsükleid on Raintrees, on RSLis üsna palju, kuid enamus neist töötavad vaid kindlas alamdomeenis. Üldotstarbelisi tsükleid on kaks: *while* ja *repeat* tsüklid.

Repeat...until tsükli algusest loetakse märksõna *repeat* ja lõpuks sõna *until*, millele järgneb tingimus. Instruktsioone tsükli sees täidetakse seni kuni tingimus *until* saab tõeseks. Süntaksist järeldub ka, et *repeat* tsüklit läbitakse alati vähemalt üks kord. While tsükkel algab võtmesõnaga *while*, millele järgneb tingimus, mida kontrollitakse. Tsükli lõpuks loetakse *endwhile* sõna. Tsükklit täidetakse seni, kuni tingimuslause on tõene. Nii *repeat* kui *while* tsükli puhul toimub tingimuse tõeväärtustamine samade reeglite põhjal nagu if-lauses.

```
set %i = 1
repeat
 set %i := inc(%i)
until (%i >= 10)
// %i väärtus on 10
while (%i > 2)
 set %i := %i - 1
endwhile
// %i väärtus on 2
```

4.5. RSL Massiiv

RSL massiiv on oma olemuselt assotsiatiivne, see tähendab andmeid salvestatakse võtme ja väärtuse paaridena [2]. Massiivi loomiseks ja elementide lisamiseks on mitmeid viise:

```
//Luuakse tühi massiiv
set %arr := array()
//Massiivi võib ka tekitada kui lisada sellele element.
//Massiivi lisatakse element võtmega "0" ja väärtusega "TEST".
set %arr2{} := "TEST"
//Massiivi lisatakse element võtmega "V" ja väärtusega "TEST2".
set %arr2{"V"} := "TEST2"
//Massiivi lisatakse element võtmega "1" ja väärtusega "TEST3".
set %arr2{} := "TEST3"

foreach %elem with %key in %arr2
 set %msg := "Võti on: " . %key . "; Element on: " . %elem
 message "Notice" %msg
endForEach
```

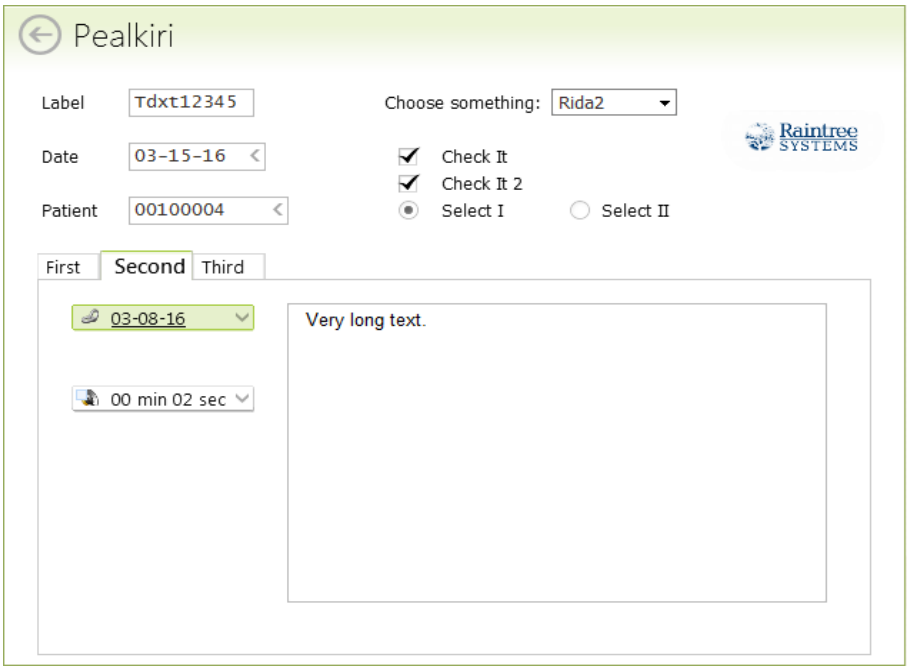
Massiivist väärtuste lugemiseks võib neid küsida võtme kaudu või läbida massiivi *foreach* tsükli abil [3]. *Foreach* on üks näide alamdomeeni spetsiifilisest tsüklist, sest see töötab ainult RSL massiivide peal. Elementi listis ei ole võimalik viida abil muuta, sest *foreach* tsükli puhul tehakse massiivi elementidest alati koopia.

5. Abistruktuurid

Raintrees kasutatakse mitmeid abistruktuure, mis võimaldavad kergesti äriloogikat lisada ja muuta. Enamasti on abistruktuurid kasutajaliidese elemenedid, mille käitumist saab RSLi abil kontrollida või mis lisavad RSLile funktsionaalsust.

5.1. RTM

RTM on failitüüp laiendiga rtm, mis defineerib vaate ehk akna väljanägemise Raintree kasutajaliideses ning samuti määrab ära, mis andmeid on võimalik näha ja sisestada (vt. Joonis 3). RTM koosneb nn vidinatest, mille hulka kuuluvad väljad ja sildid. Väljadesse saavad kasutajad andmeid sisestada, sildid on vaid informatsiooni kuvamiseks. Kasutajale kuvamisel seostatakse RTM automaatselt ühe kirjega MySQL andmebaasi tabelis. Tabeli nimi on kujul *usefile_screen*, kus *usefile* on määratud agendimuutjaga *usefile* ning *screen* on RTMi nimi ilma faililaiendita. Agendimuutuja *index* määrab kirje tabelis. Kui kirje on andmebaasis olemas, siis tõmmatakse akna kuvamisel kõik väärtused mällu ja vastavad väljad täidetakse. Andmebaasis võib olla väärtusi, mida RTMis ei ole defineeritud, sellisel juhul kasutaja neid ei näe, aga RSLis on võimalik kõiki väärtuseid kohandatavate tüüpkirjete muutujate kaudu kätte saada. Akna salvestamisel kirjutatakse kõik mälus olevad väärtused tagasi andmebaasi. Kui *index*-iga määratud kirjet andmebaasis ei eksisteeri, siis see luuakse.



Joonis 3. *RTM akna vaade väljade ja kujunuselementidega.*

5.1.1. Väljade tüübid

RTMi peale on võimalik lisada erinevat tüüpe välju. Kõige lihtsamaks tüübiks on *field*, mis on piiratud pikkusega tekstiväli. Kasutaja saab välja sisestada vaid niipalju tähemärke kui on välja definitsioonis lubatud. *Combobox* võimaldab teha valikuid vaid etteantud väärtuste hulgast. Lisaks on olemas *Check Box* väljad, millel eksisteerib kaks võimalikku väärtust: kas on valitud või mitte. Ette antud väärtuse hulgast valimiseks kasutatakse *Radio Button* vidinaid. Igale *Radio Button*'ile saab määrata väärtuse ning neid saab grupeerida. Grupist

saab olla korraga valitud vaid üks *Radio Button* ning grupp moodustab välja. Kõik mainitud väljade väärtused salvestatakse kuvatava aknaga seotud tabelisse, see on võimalik, kuna sisestatavate andmete maksimum pikkused on määratud.

Kui sisestavate andmete pikkust ja mahtu ei taheta piirata, siis on võimalik kasutata selleks ette nähtud välju. Antud väljade puhul salvestatakse aknaga seotud tabelisse vaid link päris andmetele. Sisestatud andmed asuvad tabelis, mille nimi koosneb muutuja *usefile* väärtusest ja sõnest "_blob", näiteks visit_blob. *RTF* on välja tüüp, kuhu saab sisestada lõpmatu pikkusega RTF formaadis teksti. Piltide lisamiseks on olemas mitu vidinat: *Image widget*, mis näitab kasutaja listatud pilti eelvaatena akna peal, ning *Image Link Button*, mis võimaldab lisatud pilti vaadata nupule vajutades. *Annotation Widget* võimaldab kasutajal taustapildi peale joonistada, seal juures taustapilt on salvestatud RTM faili. Heli failide jaoks on *Audio Widget*, mis võimaldab heli salvestada ja mängida. Lisaks on olemas kas *Attament Button*, kuhu on võimalik erinevaid faile salvestada.

5.1.2. Kujunduslikud vidinad

Kujunduslikud vidinad ei võimalda andmete sisestamist, nende eesmärk on määrata akende väljanägemine ja anda edasi informatsiooni.

- Label – Võimaldab ekraanil teksti kuvada.
- Picture Widget – Piltide näitamiseks. Pilti hoitakse RTM failis.
- Panel – Loob raamistiku vidinate jaoks. Vidinaid saab selle peale asetada.
- Tab Control – Loob mitmelehelise raamistiku vidinate jaoks, kus vidinad võivad asetseda erinevatel lehtedel. Korraga saab nähtav olla vaid üks leht.
- Shrinker – Võimaldab vidinaid peita koos akna kõrguse vähendamisega. Seda saab asetada vaid RTMi vasakusse äärde ning nähtavust kontrollitakse RSL skripti abil.
- Repeater – Võimaldab selle sisse asetatud vidinaid dünaamiliselt paljundada ehk korrata. Korduste arvu saab määrata RSLi abil, kõikide vidinate nimesse lisatakse number, mis määrab korduse arvu. Akna kõrgust suurendatakse *repeater*'i korduste võrra.

5.1.3. Funktsionaalsed vidinad

Vidinaid, mis ei ole ei väljad ega oma kujunduslikku tähendust, võib nimetada funktsionaalseteks. *EMR Link Button* võimaldab väljale linkida *emr* tabelis eksisteeriva kirje või lisada uut *emr* kirjet. Andmebaasi kirjutatakse vidina nimega väljale *guid* väärtus *emr* tabelist. RTMile on ka võimalik lisada nuppe, mille jaoks välja andmebaasis ei tehta, need on tüüpi *Image Button*. Nupule saab külge panna RSL funktsiooni, mida kutsutakse välja nupule vajutades. Lisaks saab RTMi peale lisada *List Box* vidinaid, mida kasutatakse nimekirjade kuvamiseks. *List Box* vidinaid võib realiseerida kolmel erineval viisil. Listi võib üles ehitada RSL skripti abil, *emlist* definitsiooni põhjal või *picklist* definitsiooni põhjal. Esimesel kahel juhul on list mõeldud andmebaasi kirjete kuvamiseks. Viimasel juhul on list mõeldud andmete talletamiseks, kus *picklist* määrab struktuuri andmete jaoks. MySQL tabelisse lisatakse *List Box* vidina puhul alati link, kuid ainult *picklisti* korral viitab link reaalsetele andmetele.

5.1.4. Vidinate ja akna omadused

Vidinad RTMi peal omavad teatud omadusi, mis vidina tüübi järgi võivad erineda. Iga vidin omab nime, mis on unikaalne, ja asukohta akna ülemise vasaku ääre suhtes. RTM aken on jagatud ruudustikuks, seega saab asukohta määrata positiivsete täisarvude abil. Samuti on olemas omadus nimega *Tab Control*, mis seob elemendi *Panel*, *Tab Control* või *Repeater* vidinaga.

Vaikimisi saab *field* väljadele sisestada kõiksuguseid tähemärke, kuid maski abil saab sisendit kitsendada. Näiteks määrates maskiks *nnn*, saab *field* tüüpi väljale sisestada vaid kuni kolme märgi pikkust numbrit. Määrates maskiks *nn-nn-nn*, tehakse field kuupäeva tüüpi, mis võimaldab kasutajal kalendrist kuupäev valida. *Field* välja on ka võimalik linkida mõnele standard tabelile, selleks tuleb kasutada & märki maski ees. Näiteks *&docto* mask seob välja *doctor* tabeliga, ehk kasutaja saab valida väärtusi ainult arsti tabelist, väljale kirjutatakse valitud arsti kood.

Kõikidel väljadel on omadused nimedega *No SQL Field* ja *Data Field Name*. Esimene neist määrab, kas välja kohta tuleks teha veerg MySQL tabelis ja teine määrab, mis nimega on väli andmebaasi tabelis. Kui *Data Field Name* on tühi, siis kasutatakse andmebaasis vidina enda nime. *Data Field Name* võimaldab RTM väljadel jagada väärtust.

Akna omaduste alla kuulub näiteks akna pealkiri, mõõtmed ja vaikimisi asukoht. Samuti saab määrata, kas aken on salvestatav ja akent lukku panna, ehk keelata kasutajal andmete muutmine. Üks oluline omadus on *Screen Type*, mis defineerib kuidas agent RTMi käsitleb.

5.1.5. Kuularid

Kuularid on RSL funktsioonid, mida kutsutakse välja teatud kasutaja tegevuste peale. Kuularid on seotud RTM akende definitsioonidega ning need võivad asetseda failides [RTMnimi]_handler.frm ja [RTMnimi]_events.frm. Näiteks akna test.rtm kuularid asetsevad test_handler.frm ja test_events.frm failides. Kuulari failid peavad olema RTMiga samas kasutas. *Handler* ja *events* failide vahe seisneb selles, et handler'eid arendatakse Raintree poolt, nendes on nn standard loogika, mida jagavad Raintree installatsioonid. Loogika *events* failides on Raintree implementatsiooni spetsiifiline ning neid võib vabalt muuta.

Kuularid võivad olla seotud kas RTM aknaga või vidinatega. Igal väljal on olemas *On Change* kuular, mida kutsutakse väärtuse muutumise korral. Näiteks välja *emrdoc* kuulariks oleks:

```
localfunc emrdoc_Change()
 //Siin on kood
endfunc
```

Aknaga seotud kuularid on näiteks WINDOW_BeforeShow() ja WINDOW_AfterClose(), kus esimene kutsutakse välja enne akna kuvamist ja teine pärast akna sulgemist. Võimalike kuularite nimekiri sõltub *Screen Type* omadusest.

Kuulareid kutsutase välja kindlas järjekorras [4]. Kõigepealt kutsutakse välja *events* faili funktsioon ning seejärel handler'i kuular. On ka võimalik defineerida funktsioon *events*

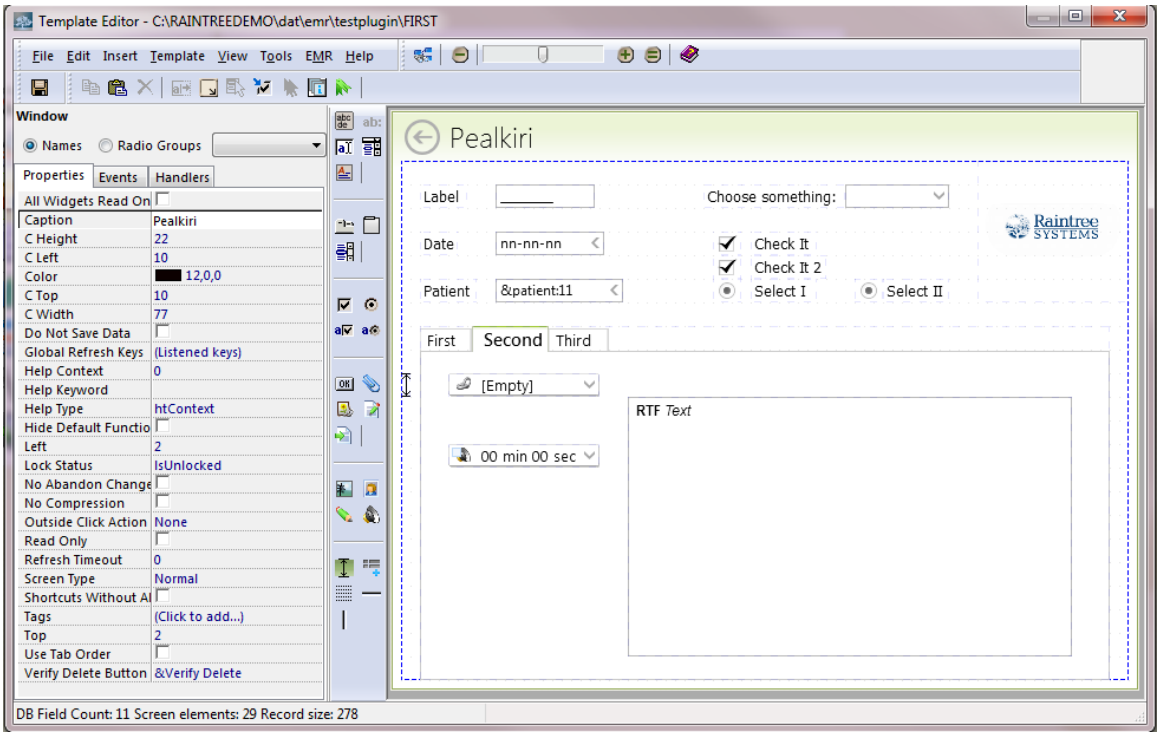
failis, mis kutsutakse välja kõige lõpus ning igas kuularis on võimalik lõpetada väljakutse ahel.

5.1.6. RTM seaded

Kui on vajadus tekitada andmebaasi veerg, mida kasutaja aknas ei näe, siis tuleks kasutada peidetud välju, mis tuleb seadete all üles loetleda. Selleks et RSL skriptis vidinaid koos adresseerida, tuleks need lisada ühte vidinagruppi. Näiteks on võimalik RSL skriptiga kõik gruppi kuuluvad vidinad ära peita, on vaja teada ainult grupi nime. Lisaks on mõned grupid agendi poolt kontrollitavad. Näiteks grupp *NoRollForward*, millesse kuuluvate väljade väärtusi ei kopeerita automaatselt järgnevatele kirjetele.

5.1.7. RTMi muutmine

RTMi loomiseks ja muutmiseks on Raintree klienti sisse ehitatud vahendid. Tööriist, millega saab RTM faile muuta, kannab nime Template Editor (vt. Joonis 4). Avades kasutajaliidese kaudu mõni aken, piisab Ctrl+k vajutamisest, et avada vastav RTM Template Edito-ris. Teine võimalus on vajutada Ctrl+r, mis avab arenduskeskkonna nimega RIDE, mille kaudu on võimalik RTM failisüsteemist üles otsida.



Joonis 4. *Template editor.*

Template Editor näitab vasakul akna välimust, ning paremal vidinate ja akna omadusi. Lisaks on võimalik hõlpsasti lisada kuulareid *handler* je *events* failidesse. Peidetud välju ja vidinagruppe saab üles seada valides tiitlimenüüst “Template”.

5.2. Menüüd

5.2.1. XML menüüd

Kõik menüü definitsioonid asuvad andmebaasikataloogis menu kaustas, kus iga menüü jaoks on alamkaust, mille nimeks on menüü kood. Menüü, mida näidatakse kasutajaliidese kaudu, pannakse kokku mitmest XML failist. Failides on defineeritud menüü valikud, kus igal elemendil peab olema id. Lisaks võib olla määratud pealkiri, link ja klaviatuuri nupp, millega saab elementi valida. Kui pealkirja ei ole defineeritud, siis menüü element peidetakse. Menüü elemendi lingiks võib olla teine menüü, RSL skript, RTM fail või PHP kood. XML failid võivad asetseda ka EMR pluginates menu kaustas. Reeglid menüü kokkupanekuks on järgmised.

- Kõigepealt otsitakse üles standard.xml fail [andmebaasikataloog]\menu\[kood]\ kaustast ning loetakse sisse menüü elemendid.
- Otsitakse kõik .xml laiendiga failid [andmebaasikataloog]\menu\[kood]\ kasutast, mille nimi pole custom.xml. Failid loetakse sisse tähestiku järjekorras. Failid võivad defineerida uusi menüü elemente, üle defineerida mõne elemendi parameetreid või muuta elementide järjekorda.
- Loetakse sisse .xml laiendiga failid [andmebaasikataloog]\emr\[pluginakood]\menu\[kood]\ kaustadest.
- Loetakse sisse seadistused [andmebaasikataloog]\menu\[kood]\custom.xml failist.
- Lõpuks kirjutatakse menüü [andmebaasikataloog]\menu\[kood]\custom_[andmebaas]\ menu.xml faili. Failis on menüü sellisel kujul nagu seda näidatakse. Menu.xml lõpus kirjas kõik failid, millest menüü kokku on pandud.

Menüüd saab muuta kahel viisil. Üks võimalus on avada XML fail tekstiredaktoris ning teha vastavad parandused. Muuta ei tohiks menu.xml faili, sest see pannakse kokku agendi poolt. Teine võimalus on kasutada Raintree Menu Editor tööriista, selleks tuleb Raintree kasutajaliidese kaudu avada menüü ja vajutada Ctrl+k. Raintree Menu Editor võimaldab muuta ainult custom.xml faili.

5.2.2. Dünaamilised menüüd

Menüüd saab kokku panna ja kuvada ka RSL skripti abil, selleks tuleb kasutada *dynmenu* käsku.

```
dynmenu clear
dynmenu caption "Menüü"
dynmenu add "Valik 1"
dynmenu add "Valik 2"
dynmenu show

if escaped <> "Y" and fldpos <> "" then
begin
// %pos väärtuseks saab kas 1 või 2
set %pos := trim(fldpos)
end
```

Näite koodis pannakse kokku kahest valikust koosnev menüü ning näidatakse seda kasutajale. Kui kasutaja on valiku teinud, siis saab *fldpos* agendimuutuja väärtuseks valitud elemendi number.

5.3. EMR pluginad

EMR pluginad võimaldavad modulaarselt arendada ja teostada äriloogikat Raintrees. Plugin määratakse failisüsteemi kausta, mis asub emr kataloogis, ja koodi kaudu. Kõik failid plugina kaustas kuuluvad ainult antud plugina alla. EMR pluginaid kasutatakse, et grupeerida äriloogikat funktsionaalsuse järgi. Plugina tekistamiseks on vaja luua plugina kaust ning selle sisse settings.ini fail, mis sisaldab seadeid, seal hulgas plugina koodi. Agent otsib seaded failisüsteemist üles ja kirjutab need andmebaasi *plugins* tabelisse, et vajalike failide üles leidmine toimuks kiiremini. Lisades uut pluginat kasutajaliidese läbi, tekitab agent vajalikud kaustad ja failid ise ning uuendab *plugins* tabelit automaatselt.

5.3.1. EMR *template* ja EMR kirje

EMR pluginate alla saab lisada EMR *template* definitsioone. *Template* on mall, mille järgi lisatakse kirjeid *emr* tabelisse. Failisüsteemis hoitakse EMR *template* definitsiooni failis, mille laiendiks on *emr* ja nimeks *template* kood. Sarnaselt pluginatega otsib agent failisüsteemist üles kõik EMR *template* definitsioonid ning kirjutab need andmebaasi *emrtemplates* tabelisse. *Emr* tabelis hoitakse kohandatavate tabelite kirjeid ühendavaid väärtusi. Üldiselt võib *emr* tabeli veerud omaduste poolest jagada kaheks. Osad veerud hoiavad nn seadeid (vt. Tabel 1), mis määravad kuidas EMR kirjet kasutajaliidese kaudu käsitletakse. Vaikimisi võetakse seaded EMR *template* definitsioonist, kuid neid on alati võimalik and-mebaasis üle kirjutada. Ülejäänud veerud on kirjete eristamiseks emr tabelis, näiteks eksisteerivad veerud patsiendi numbri, kuupäeva ja kirje omaniku salvestamiseks. Samuti on olemas unikaalne veerg *guid*, mis hoiab 11 märgi pikkust koodi. EMR avamisel seab agent *index* väärtuse *guid* veerust.

Tabel 1. Seadetega seotud veerud EMR tabelis.

Veeru nimi	Kirjeldus
templ	EMR *tempalte* kood.
usefile	Agent seab üles vastavad *usefile* väärtuse EMR kirje avamisel.
link	RSL skrip, mida kutsutakse välja, või RTM, mida näidatakse, EMR kirje avamisel. RTM nimeks saab olla vaid *template* kood.
cat	Kood, mille abil saab EMR kirjeid grupeerida.
soscript	RSL skript, mida mida kutsutakse välja EMR kirje allkirjastamisel.
package	Kausta, kus asub EMR *template*, nimi. EMRi avamisel otsitakse faile vaikimisi antud kaustast.
modeflags	Hoiab lippudena seadeid, mis üldiselt ei muutu EMR kirje elutsükkli jooksul. Näiteks, kas kirjega on seotud ka narratiiv.
noteflags	Lipud, mis määravad kirje oleku. Näiteks, kas kirjet on parandatud pärast allkirjastamist.

Usefile, *templ* ja *guid* väärtuste põhjal saab emr tabeli kirje otseselt siduda kirjega kohandatavas tabelis. Tabeli nimi pannakse kokku *usefile* ja *templ* väärtustest ning *_pn* veerg seotakse *guid* väärtusega. EMR kirje avamisel kasutajaliidesest toimub kohandatava tabeli kirjele positsioneerimine automaatselt.

5.3.2. EMR kirjed RSLis

EMRi kasutamine võimaldab hõlpsasti saada kätte väärtusi kohandatavatest tabelitest. Emr tabelis saab vajalikud kirjed välja filtreerida ning siis küsida väärtusi kohandatavast tabelist. EMR kirjete jaoks on agendis olemas eraldi kontekst, see tähendab et eksisteerivad kirjetele vastavad agendimuutujad. Lisaks on EMR konteksti mälus hoidiseks olemas ka magasin.

```
set %date := "02-29-16"
set %i := 0
emrpush
 pushindex
   emrindex "category"
   emrfilter "fromdate" %date
   emrfilter "toDate" %date
   emrfilter "category" "VISIT"
   loop emr
     usefile emrusefile
     index emrguid
     if &[emrtempl]:noVisit = "" then set %i := inc(%i)
   next emr
 popindex
emrpop
```

Näites otsitakse üles kõik “VISIT” kategooria kirjed kuupäeval 29.02.2016 ning loetakse kokku kui paljude puhul nendest ei oma kohandatavas tabelis *novisit* väli väärtust. *Emrpush* käsib meelde jätta kõik hetkel mälus olevad *emr* tabeliga seotud väärtused lükates need magasini. *Emrpop* käsu peale muudetakse jälle aktiivseks kõik *emrpush* käsuga magasini pandud muutujate väärtused, ehk taastatakse eelnevalt meele jäetud kontekst. *Pushindex* ja *popindex* töötavad sarnaselt, aga antud käsud mõjutavad konteksti, mida seatakse *usefile* ja *index* abil. *Emrindex* käsk teatab agendile, et järgnevalt tuleb *emr* tabelist kirjete otsimiseks kasutada indeksit *category*, mis emr tabelis on *_index8*. *Emrfilter* käsud annavad ette filtrid EMR kirjete otsimiseks. *Loop emr* ja *next emr* moodustavad tsükli, mida korratakse iga filtrite põhjal leitud vaste jaoks. Igal tsükli läbimisel ollakse positsioneerutud ühele emr kirjele. Kohandatava tabeli kirjele positsioneerimiseks kasutatakse *emrusefile* ja *emrguid* muutujaid. Kuna RSL on interpreteeritav keel, siis saab muutujate poole pöördumisel kasutada muutujaid, selleks tuleb muutuja asetada kantsulgude vahele. Näite koodi täitmisel asendatakse *[emrtempl]* väärtusega, mis on *emrtempl* muutujal. Erisüntaksina võiks ka kohandatava tüüpkirje muutujat kirjutada kujul *&self:novisit*, EMR kirjele positsioneerudes on *self* väärtuseks *template* kood. Tuleks mainida, et antud näide ei ole efektiivne moodus loendamiseks. Selleks et teada saada *novisit* välja väärtust tõmmatakse kontekstide seadmistega mällu palju ebavajalikku informatsiooni.

EMR kirje peale saab positsioneeruda ka *emrposguid* käsku kasutades, selleks tuleb parameetrina ette anda *guid* väärtus. Samuti on võimalik RSLis EMR kirjeid lisada, selleks tuleb kasutada *emradd* või *emraddrec* käske.

```
emrpush
 set %guid = guid
 emraddrec "templ" "DOC" "pat" "00000001" "guid" %guid "silent"
true
 emrposguid %guid
 if emrguid = %guid then set %added = true
emrpop
```

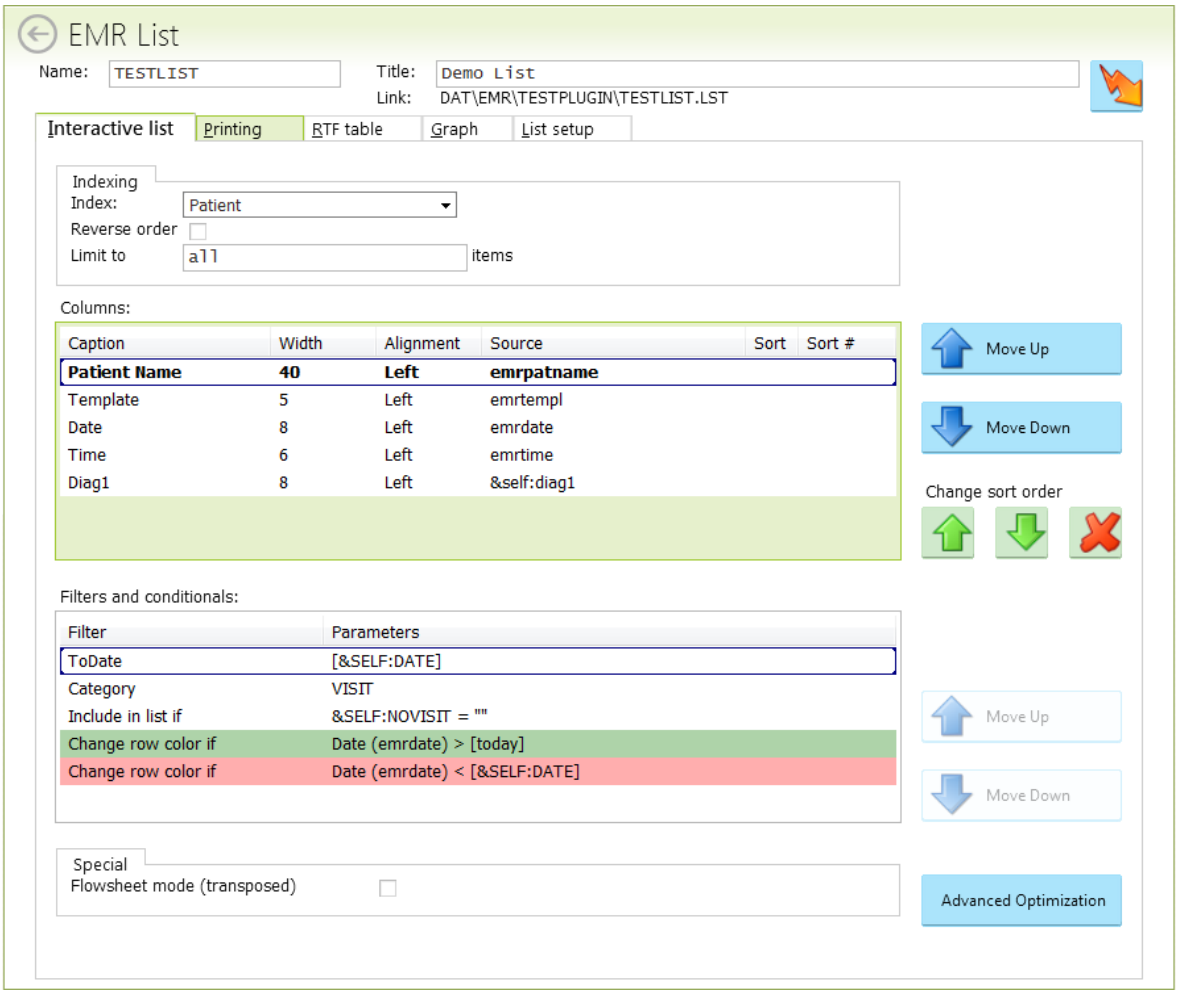
Näites saab %guid muutuja väärtuseks antud hetkel genereeritud 11 märgi pikkune sõne. EMR kirje loomiseks kasutatakse emraddrec käsku, millele antakse näites ette parameetrid *templ*, *pat*, *guid* ja *silent*. Parameetrite järjekord ei ole oluline ning ainukesena on kohustuslik *templ* parameeter. *Templ* määrab, et lisada tuleb "DOC" koodiga EMR kirje, *template* põhjal väärtustatakse ka muid parameetreid, näiteks *usefile* ja *link*. *Pat* parameeter määrab, et patsiendi numbriks saab "00000001". Kui numbrit ette ei anta, siis võetakse see hetkel kontekstis olevalt patsiendilt. EMRi lisamisel kutsutakse välja ka *link* välja viide. Andes ette parameetri *silent*, ei näidata kasutajale akent kui *link* viitab RTM failile. RTMiga seotud kuularid läbitakse endiselt, näiteks kutsutakse välja *Window_beforeAdd()* funktsioon. Pärast lisamist positsioneeritakse näites vastsele kirjele *emrposguid* käsuga. Kui kirje eksisteerib andmebaasis, siis saab *emrguid* väärtuseks parameetrina ette antud *guid*.

5.4. EMR listid

EMR listid võimaldavad kasutajatele kuvada EMR kirjeid. Listi definitsioone hoitakse *lst* laiendiga failides ning kõige sagedamini lisatakse neid *List Box* vidina külge RTMi peal. EMR listis tuleb alati defineerida filtrid ja väljundi veerud. Listide muutmiseks tuleks fokuseerida listi elemendi peal RTMil ja vajutada Ctrl+o, seejärel avaneb listi definitsioon vastavas tööriistas, mis on näha joonisel 5.

Filtrite põhjal otsitakse üles kirjed *emr* tabelist. Filtreerida saab nii *emr* tabeli kui ka kohandatavate tabelite väärtuse järgi. Kui filtrites antakse parameeter ette kantsulgude vahel, siis otsitakse väärtus agendi mälust, mis võimaldab filtritena kasutada väärtusi EMR listi kuvava RTMi pealt. Näiteks kui on parameetrina kasutatud sõne "[&self:date]", siis asendatakse see listi kuvamisel välja *date* väärtusega aknalt, mis listi kuvab. Kui aga kasutatakse sõne "&self:date", siis võetakse väärtus *emr* kirjega seotud kohandatava tabeli *date* veerust. Lisaks on võimalik määrata rea värvi filtreid. Rea värvi määramiseks vaadatakse tingimusi ülevalt alla, kirjele vastav rida saab värvi esimesest tingimusest, mis vastab tõele. Listi laadimiseks teeb agent reeglina palju väikeseid päringuid *emr* tabelisse. *Index* väärtus listi seadetes määrab, mis indeksit agent päringute tegemise ajal kasutab. Agent paneb paika indeks filtrite alguse ja lõpu väärtused ning jagab indeksi vahemiku väiksemateks osadeks. Seejärel tehakse iga jagatud vahemiku kohta päring *emr* tabelisse ning vajadusel tehakse lisa päringuid kohandatavasse tabelisse. Eeltoodust järeldub, et listi defineerimisel on tähtis valida õige indeks, mis filtrite põhjal tagastaks võimalikult vähe väärtusi, sest MySQL ei

saa indeksit valida. Valides *index* väärtuseks *patient*, ei ole vaja filtrites patsienti defineerida, agent võtab patsiendi numbri automaatselt hetkel kontekstis olevalt patsiendilt.



Joonis 5. *EMR listi muutmise tööriist.*

Kuvatavate veergude jaoks saab määrata pealkirja, vaikimisi pikkuse, joonduse, väärtuse allika ning sorteerimise järjekorra. Veeru väärtus võib samuti pärineda nii emr tabelist, kui ka kohandatavast tabelist. Väärtusi saab defineerida vastavate agendimuutujate või kohandavate tüüpkirjete muutujate kaudu. Lisaks võib veeru väärtus pärineda agendifunktsioonist.

5.4.1. Listi Seaded

EMR listid toetavad peale kirjete kuvamise ka muid operatsioone. Näiteks on võimalik listi kaudu uusi kirjeid lisada, olemasolevaid avada, muuta, kustutada ning listi printida. Kõiki lisaoperatsioone saab listi seadetes sätestada. Kirjete lisamise võimaldamiseks tuleb seadetes määrata, mis *template* koodiga EMRe lubatakse lisada. Kui kasutaja lisab uue kirje EMR listi kaudu, siis teatud väärtused määratakse listi filtritest. Näiteks kui EMR list näitab ainult “VISIT” *category* väärtusega kirjeid, siis lisatavatel kirjetel saab ka *category* väärtuseks “VISIT”, olenemata EMR *template* seadetest. Sarnaselt ka kui EMR listis on määratud kuupäeva filter, siis lisatavate kirjete kuupäev võetakse ka filtri väärtusest.

5.4.2. Listid RSLis

EMR listide abil on ka võimalik RSLis kirjeid üles otsida. Näiteks *emrfilter* ja *loop emr* käskude asemel võib kasutada *emrprocesslist* käsku.

```
emrprocesslist "mylist.lst" "functionToCall"
end.

localfunc functionToCall(var %abort)
 set %pn := emrpn . " " . emrpatname
 message "Notice" %pn
endfunc
```

Näites antakse *emrprocesslist* käsule ette kaks parameetrit. Esimene on listi fail ja teine on funktsiooni nimi. Agent kasutab listi definitsioonist filtreid, et üles leida emr kirjeid ning seejärel kutsub välja parameetrina ette antud funktsiooni iga leitud vaste jaoks. Funktsiooni välja kutsudes positsioneerib agent EMR kirje peale. Näites saab muutuja *%pn* väärtuseks patsiendi number ja nimi ning seejärel kuvatakse muutuja väärtus teates. Teadet näidatakse iga kirje kohta, mis vastab EMR listi filtritele.

5.5. Narratiivid

Narratiivid on definitsioonid, mille põhjal koostatakse tekstidokumente. Failisüsteemis esinevad narratiivid põhiliselt *nar* laiendiga, erandiks on *txt* laiendiga failid *docpath*'is. *Docpath* kataloogis asetsevad failid on seotud kirjetega *letter* tabelis, failide nimedeks on koodi väärtus *letter* tabelis. Narratiivid salvestatakse RTF formaadis, seega saab neid vormindada nagu tekstidokumente. Narratiivile on võimalik nurksulgude vahele lisada märgiseid, mille agent dokumendi koostamise ajal asendab kindla väärtusega. Samuti saab narratiividele lisada juhtimiskäsklusi, mis määravad dokumendi lõpliku väljanägemise.

5.5.1. Märgised

Kõige enam esinevad märgisena agendimuutujad või kohandavate tüüpkirjete muutujad, lõppdokumendil asendatakse need väärtustega agendi mälust. Näiteks märgis <*first*> asendatakse dokumendi koostamise ajal patsiendi eesnimega, juhul kui ollakse positsioneeritud patsiendile. Märgisesse saab lisada ka agendifunktsioone, sellel juhul dokumendile jõuab funktsiooni väärtus. Üldiselt võib märgis sisaldada erinevaid RSLis toetatud avaldisi, selleks tuleb kasutada fsexp *käsku*. Näiteks <*fsexp:"chr(97)"*> asendatakse tähega "a", täpselt sama tähe tagastaks avaldis *chr(97)* RSLis. Samuti on võimalik märgise abil välja printida EMR liste.

5.5.2. Juhtimiskäsklused

Juhtimislkäsklused võimaldavad juhtida agendi tööd dokumendi loomisel. Näiteks saab kasutada *if* tingimuslauset. *If* bloki sees olev tekst lisatakse dokumendile ainult siis, kui tingimus on tõene.

```
@if <emrpatname> <> "" then
Patsiendi nimi on <emrpatname>
@endif
```

Näites lisatakse rida dokumendile juhul kui *emrpatname* muutuja omab väärtust. Oluline on, et *@if* ja *@endif* oleks narratiivil eraldi ridadel. Samuti ei tasu narratiivile tekitada taandeid loetavuse paremaks muutmiseks, sest lõppdokumendil jääksid taanded näha. Näites oleva *if* tingimuse saaks asendada ka *@block* ja *@endblock* plokiga.

```
@block
Patsiendi nimi on <emrpatname>
@endblock
```

@block ja *@endblock* vahel olev tekst lisatakse dokumendile ainult juhul kui mõni märgis ploki sees omab väärtust. Tabelite jaoks on olemas ka *@tableblock* ja *@endtableblock*, mille puhul peidetakse tabeli rida juhul kui ükski märgis tabeli reas ei oma väärtust. Narratiividele on võimalik lisada ka teisi narratiive, selleks tuleb kasutada *include* käsku. Näiteks *<include:teine.nar>* asendatakse sisuga teine.nar failist. Samuti on võimalik narratiividel kasutada agendi käske. Üheks võimaluseks on kasutada *fsexp* võtmesõna, kuid teatud käsud eksisteerivad ka erisüntaksina. Näiteks *<usefile:"TEST">* määrab *usefile* väärtuseks "TEST", seega saab dokumendi koostamise ajal ka kontekste muuta. Dokumendi koostamise käigul seatud kontekstid taastatakse kui dokument on valmis.

5.5.3. Narratiivide kasutamine

Raintrees saab narratiive seostada EMR *template* definitsioonidega. Selleks peab narratiiv olema *template*'i koodiga sama nimega ning asetsema samas pluginas. Lisaks tuleb EMR *template* seadetes märkida, et narratiiv on kasutusel. EMR kirje salvestamisel Raintree kasutajaliideses luuakse narratiivi põhjal dokument, mis salvestatakse *emr_blob* tabelisse. Kasutaja saab ka käsitsi muudatusi teha EMR kirjega seotud dokumendile. Selleks, et järgmisel salvestamisel käsitsi tehtud muudatused kaotsi ei läheks, pannakse EMR kirje külge lipp ning hoiatatakse kasutajat muudatuste tegemise eest. Olemasolevat dokumenti on RSL-is võimalik kergesti välja printida *emrprint* käsuga, mille puhul saadetakse olemasolev dokument printimise puhvrisse.

RSL abil saab ka luua dokumente narratiivide põhjal vastavalt nõudmisele. Käsk *merge* võimaldab dokumenti kirjutada faili või printimise puhvrisse.

merge "testplgn:esimene.nar" "PRINT"

merge käsu esimeseks parameetriks on narratiivi fail. Antud näites asetseb fail nimega *esimene.nar testplgn* EMR plugina kaustas. Teine parameeter näites ütleb, et dokument tuleb saata printimise puhvrisse. Kui teiseks parameetriks oleks faili nimi, siis kirjutatakse valmis tehtud dokument faili.

5.5.4. Narratiivide loomine ja muutmine

Narratiivi avamiseks ja muutmiseks on mitu moodust. Esiteks saab narratiivi avada läbi RIDE rakenduse, kust on omakorda võimalik narratiivi avada vastavas sisse ehitatud redaktoris, mis suudab näidata RTF teksti. Kui narratiiv on seotud EMR *template* definitsiooniga, siis saab narratiivi avada *template* seadete alt. Narratiivi loomiseks tuleks luua tühi *nar* laiendiga fail või teha koopia mõnest olemasolevast narratiivist. Raintreesse sisse ehitatud tööriist ei ole suurte võimalustega, saab küll lisada teksti ja tabeleid, aga näiteks

tabelite veerude ja ridade suuruseid muuta ei saa. Piirangutest mööda pääsemiseks võib elemendid valmis teha mõnes teises RTF redaktoris ning seejärel kopeerida Raintree redaktorisse.

6. Õppevideode koostamine kursuse jaoks

Töö raames valminud õppevideod on mõeldud täiendama kirjalikku materjali. Videoid tehti paralleelselt kirjaliku osaga ning teemade osas need üldiselt kattuvad. Videotes tehakse läbi mõned programmeerimisülesanded või näidatakse visuaalselt Raintree omapärasid.

Enne videote salvestamist sai valmis tehtud abimaterjalid, milleks olid kas slaidid või koodi näited. Salvestamisel näidati ainult arvuti ekraani pilti, mis võimaldas videoid valmistada 30 sekundiliste kuni 5 minutiliste klippide kaupa, mida sai hiljem kokku monteerida. Enne videoklipi koostamist tuli valmis mõelda kui palju antud klipis näidata ning salvestamise ajal tuli hiir liigutada nõnda, et monteerimise järel jääks mulje ühest terviklikust videost. Teatud juhtudel tuli klipile ka hääl uuesti peale lugeda, näiteks kui selgus, et esmase salvestamise ajal esines liiga palju sõna kordusi.

Videote salvestamiseks kasutati programmi Ink2Go, mis võimaldab salvestada korraga heli, ekraani pilti ning vajadusel ka välise kaamera pilti. Heli salvestamiseks kasutati kõrvaklappide küljes asuvat mikrofoni. Sai katsetatud ka muude mikrofonidega, aga need salvestasid liiga palju müra väliskeskkonnast, eriti kui mikrofon asetses samal laual kus arvuti. Videoklippe monteeriti kokku Windows Movie Maker programmiga, mis võimaldab klippide algust ja lõppu ära lõigata ning heli lisada. Videote resolutsiooniks on 1024x576, kuna teatud juhtudel esineb videotes väiksema kirjaga teksti, mis ei oleks madalama resolutsiooni puhul arusaadav.

Kokkuvõte

Töö ise on põhiliselt RSL keele ja Raintree tutvustus. Esimeseks osas antakse ülevaade Raintree arhitektuurist. Tutvustatakse failisüsteemi ja MySQL andmebaasi tabelite struktuuri. Teises osas kirjeldatakse RSL keele omadusi, antakse läbilõige RSLi iseloomustavatest programmeerimisparadigmadest. Samuti kirjeldatakse RSLi konteksti tundlikust, millel on sarnasusi kontekst-orienteeritud programmeerimisparadigmaga [5], kuid suureks erinevuseks on, et programmeerija ei saa uusi kontekste defineerida. Kolmandas osas tutvustatakse RSLi süntaksit ning neljandas antakse ülevaade abistruktuuridest, mida saab Raintrees programmeerides kasutada.

Töö koostamisel tuli päris palju RSLi testida, et veenduda kirjapandu õigsuses, selle käigus tulid välja ka mõned vead RSLi implementatsioonis. Küllaltki ajakulukaks osutus ka õppevideote valmistamine, kuid kogemuse kasvades vähenes salvestamisele kuluv aeg ja teatud määral paranes ka videote kvaliteet. Harjutusülesannete koostamisel sai arvesse võetud sagedamini esinevaid vigu, mis RSLis programmeerides esinevad ning teatud juhtudel võeti näiteid reaalsetest probleemidest, mis on klientidel esinenud.

Käesoleva lõputöö käigus ei valminud kõik planeeritud kursuse materjalid. Täielikult valmisid materjalid Raintree arhitektuuri ja RSLi sissejuhatuse peatükkide kohta. RSLi süntaksit tutvustavatest materjalidest valmis kirjalik tekst ja harjutusülesanded. Videoist valmisid osad, mis tutvustavad RSLi muutujaid ning kirjetele positsioneerimist. Abistruktuuride tutvustuses valmis kirjalik osa koos harjutusülesannetega RTMide, Menüüde, EMR pluginate, EMR listide ja narratiivide kohta. Tegemata jäi RTOde ja *picklistide* tutvustus ning kõik järgnevad kursuse jaoks kavandatud peatükid. Tööst välja jäänud materjalid on kavas lõputöö väliselt valmis teha.

Viited

1. Kalmer Kurg. PHP ja Raintree skriptimiskeele ülevaade ja edasiarendus. Bakalaureusetöö. Tartu Ülikool, 2012.

2. Native RSL Array. Introduction.
https://wiki.rtedev.com/index.php/Native_RSL_Array#Introduction (2016-04-21)

3. Native RSL Array. ForEach.
https://wiki.rtedev.com/index.php/Native_RSL_Array#ForEach (2016-04-21)

4. RTM Events level order
https://wiki.rtedev.com/index.php/RTM_Events_level_order (2016-04-21)

5. Guido Salvaneschi, Carlo Ghezzi, Matteo Pradella, „Context-Oriented Programming: A Programming Paradigm for Autonomic Systems“, arXiv:1105.0069, 2011.
https://arxiv.org/ftp/arxiv/papers/1105/1105.0069.pdf (2016-04-23)

Lisad

I. Lühendite ja mõistete sõnastik

Agendifunktsioon	Eeldefineeritud funktsioon RSL keeles.
Agendikäsk	Eeldefineeritud käsk RSL keeles.
Agendimuutuja	Eeldefineeritud muutuja RSL keeles, mille väärtust sõltub kontekstist.
Agent	Raintree serveri poolne rakendus.
Baastabel	MySQL andmebaasi tabel, mille struktuuri määrab agent.
EMR	*Electronic Medical Record*. Kirje MySQL tabelis nimega *emr*.
EMR list	Definitsioon, mis sisaldav filtreid *emr* tabelist kirjete pärimiseks ning seadedlisti kuvamiseks.
EMR *template*	Mall, mille järgi luuakse kirjeid *emr* tabelisse.
Homedir	Kataloog kuhu on serveris installitud Raintree eksemplar.
Index	Muutuja, mis määrab kirje kohandatavates tabelites.
Klient	Raintree klindi poolne rakendus.
Kohandatav tabel	MySQL andmebaasi tabel, mille struktuuri saab määrata RTMi abil.
Kontekst	Agendi teatud olek. RSLis väljendub see agendimuutujate, -käskude ja –funktsioonide väljundi määramisega.
Kuular	RSL funktsioonid, mida kutsutakse väljakasutaja kindlate tegevuste peale. On soetud RTM definitsiooniga.
Magasin	Andmestruktuur, milles sinna viimasena kantud andmed võetakse esimesena välja.
MySQL	Vabavaraline relatsioonandmebaasi haldur.
Narratiiv	Failiformaat, mille põhjal luuakse Raintrees dokumente.
PHP	*PHP: Hypertext Preprocessor*. Programmeerimiskeel,mida kasutatakse Raintrees serveri pool.
RSL	*Raintree Scripting Language*. Programmeerimiskeel, mida interpreteerib Raintree agent.
RTF	Platvormist sõltumatu dokumendi failivorming.
RTM	Fail, mis defineerib akna väljanägemise Raintrees. Akna avamisel seotakse see alati kohandatava tabeliga.
Screen	MySQL andebaasi kohandatava tabeli nime teine pool.
Scriptpath	Kataloog, mis sisaldab Raintree andebaasi jaoks vajalikke faile.
Service	Serveri poolne rakendus,mis vahendab suhtlust kliendi ja agendi vahel.
SQL	*Structured Query Language*. Päringukeel relatsiooniliste andebaaside andmete haldamiseks.
Süsteemifunktsioon	vt agendifunktsioon.

Süsteemikäsk	vt agendikäsk
Süsteemimuutja	vt agendimuutuja.
Template Editor	Tööriist, mis võimaldab muuta RTM definitsioone.
Usefile	MySQL andmebaasi kohandatava tabeli nime esimese pool. On ka agendimuutuja.
Varamumuutuja	Muutuja, mis võimaldab väärtusi edasi anda RSL skriptide vahel. Muutuja eksisteerib, kuni agent töötab.
XML	*Extensible Markup Language*, standardne ja üldotstarbeline märgistuskeel.

II. Harjutusülesanded

Harjutusülesanded on lisana kaasas harjutused.zip failis.

Failis 1_Architecture.pdf on valikvastustega küsimused Raintree arhitektuuri kohta.

Fail 2_RSL_intro.pdf sisaldab valikvastustega küsimusi RSLi kohta.

Failis 3_RSL_syntax.pdf on küsimused RSLi süntaksi kohta ja programmeerimisülesanded.

Failis 4_RTM.pdf on harjutusülesande juhed RTMi tegemiseks, sellele kuularite lisamiseks ja menüüsse kirje lisamiseks.

Faili 5_Plugins.pdf sisaldab harjutusülesannet, milles on juhend kuidas teha EMR pluginat ja sellesse skripte lisada.

III. Õppevideod

Valminud õppevideod asuvad järgneval aadressil.

https://drive.google.com/folderview?id=0B1Ez-YX58iGueTVIcVRHRmtpX2c&usp=sharing

IV. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Mait Sarv** (sünnikuupäev: 1988-10-28)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose **Programmeerimine rakenduses Raintree ja keeles RSL**,

   mille juhendaja on Tõnu Tamme,

   1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **10.05.2016**