T A R T U Ü L I K O O L

MATEMAATIKA-INFORMAATIKATEADUSKOND

Arvutiteaduse instituut

Infotehnoloogia eriala

Kalmer Kurg

PHP ja *Raintree* skriptimiskeele ülevaade ja edasiarendus

Bakalaureusetöö (6 EAP)

Juhendajad: Madis Abel

Ain Isotamm

Vambola Leping

Autor: …… “…..“ mai 2012

Juhendaja: …… “…..“ mai 2012

Juhendaja: …… “…..“ mai 2012

Lubatud kaitsmisele:

Professor: …… “…..“ mai 2012

TARTU 2012

Sisukord

Sissejuhatus

Skriptimiskeeled on tänapäeval leidnud kindla koha teiste programmeerimiskeelte kõrval. Nad on kergekaalulised ja lihtsasti muudetavad võrreldes kompileeritavate kõrgema taseme programmeerimiskeeltega. Skriptimiskeeli kasutatakse käsurea operatsioonide kontrollimiseks, graafilistes kasutajaliidestes, veebibrauserites ja tekstitöötluses. Neid leidub nii universaalsete programmeerimiskeelte, manuskeelte kui ka rakendus-spetsiifiliste keelte seas.

Töö eesmärgiks on uurida asutuses *Raintree Estonia OÜ* ärilahenduste loomiseks kasutatavaid keeli nagu *PHP* ja *Raintree* skriptimiskeel, võrrelda neid ning seejärel pakkuda välja strateegia sobiva keele edasiarendamiseks, kus oleksid ühendatud mõlema keele implementatsiooni tugevad küljed vaadelduna domeenispetsiifilisuse aspektist.

Lugejalt eeldatakse orienteerumist erinevates üldtuntud programmeerimiskeeltes, objektorienteeritud programmeerimise ja tarkvara disaini põhitõdede tundmist. Kasuks tuleb ka kokkupuude programmeerimiskeeltega nagu *BASIC* ja *Pascal,* ning relatsioonilise andmebaasisüsteemiga *MySQL*.

Käesolev töö on jaotatud viieks peatükiks.

Esimeses peatükis on antud lühike ülevaade domeenispetsiifilistest programmeerimiskeeltest, luues sobiva aluse, mille põhjal *PHP*-d ja *Raintree* skriptimiskeelt võrrelda.

Teise peatükis antakse ülevaade *Raintree* põhirakenduse domeenist ja arhitektuurist, kirjeldatakse *Raintree* skriptimiskeele elemente ning süntaksit.

Kolmandas peatükis kirjeldatakse *PHP* integreerimist *Raintree* põhirakendusega, *PHP* tarbeks loodud *Core Library* raamistiku põhiklasse ning selle kasutamise eesmärke.

Neljandas peatükis võrreldakse *PHP* ja *Raintree* skriptimiskeelt süntaksi, domeenispetsiifilisuse, laiendatavuse ja keelte elementide aspektist.

Viiendas ja viimases osas püütakse eelnevate osade põhjal selgitada *Raintree* skriptimiskeele ja *PHP* kasutamist *Raintree Estonia OÜ*-s, nende eelised ja puudusi ning välja pakkuda lahendust olemasolevate keelte parendamiseks.

1. Sissejuhatus domeenispetsiifilistesse keeltesse

Domeenispetsiifiline keel (ingl. k *domain specific language*, *DSL*) on programmeerimiskeel, mis on kohandatud rakenduse domeenile. Selmet olla üldotstarbeline keel, kirjeldab ta täpselt domeeni semantikat. Domeenispetsiifilise keele põhine arendus adresseerib vajadust kasvava domeenispetsiifilisuse järgi tarkvaraarenduse vallas. Mõned näited domeenispetsiifilistest keeltest: *lex* ja *yacc*, mida kasutatakse programmi leksiliseks uurimiseks ja sõelumiseks (ingl. k. *parsing*); *HTML*, mida kasutatakse dokumentide kirjeldamiseks; ja *VHDL*, mida kasutatakse elektroonilise riistvara kirjeldamiseks. Domeenispetsiifilised keeled võimaldavad tihendada rakenduse loogika kirjeldust, redutseerides probleemi ja programmi semantilist kaugust, ehk teisisõnu domeenispetsiifilises keeles kirjutatu väljendab lühemalt ja täpsemalt domeeni olemust ja probleemi lahendust. [1]

DSL-id püüdlevad universaalsuse asemel suurema väljendusrikkuse poole omal kitsal alal. Kuna domeenispetsiifilised keeled pakuvad esitusi ja konstruktsioone, mis on kohandatud kindla rakenduse domeeni (ingl. k. *application domain*) suhtes, siis nad pakuvad olulist keele väljendusrikkuse kasvu ja kasutuslihtsust võrreldes universaalsete keeltega, kui küsimus on kindlas domeenis. Samas kaasneb *DSL*-idega ka produktiivsus ja vähenevad hoolduskulud. Kuna väheneb ka vajadus domeeni ning programmeerimisalaste teadmiste järele, siis domeenispetsiifilised keeled avavad oma rakenduse domeeni suuremale hulgale tarkvaraarendajatele võrreldes universaalsete keeltega. [9]

Siiski ei tule eelnevalt mainitud *DSL*-ide head küljed tasuta. Nimelt on *DSL*-ide arendamine raske, nõudes teavet nii domeenist kui ka keelte arendamisest. Teiseks on *DSL*-ide arendamise tehnika varieeruvam kui universaalsete keelte arendamise puhul, nõudes seotud tegurite hoolikat arvessevõtmist. Kolmandaks, sõltuvalt kasutajaskonna suurusest võib dokumentatsiooni koostamine, keele toetamine, standardiseerimine ja hooldamine muutuda tõsisteks ning aeganõudvateks probleemideks. [9]

2. *Raintree* skriptimiskeel

2.1 *Raintree* domeen

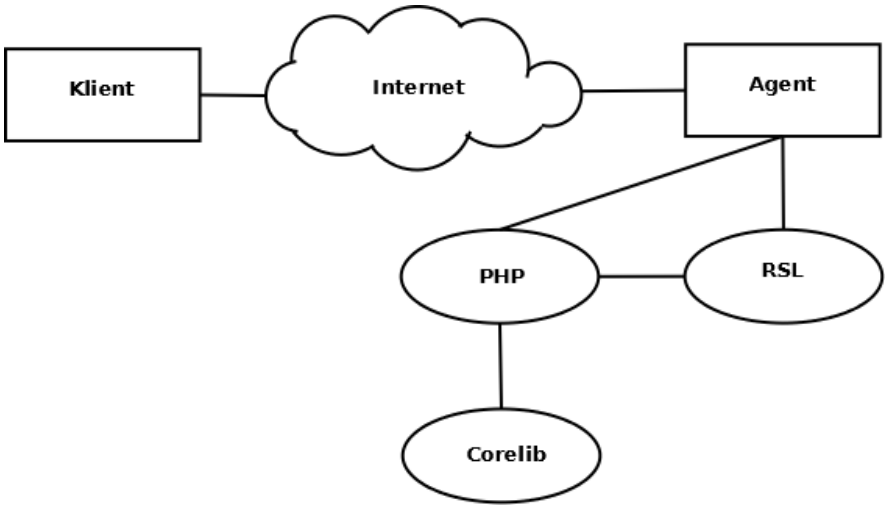
Raintree domeen hõlmab endas erinevate meditsiiniga seotud objektide andmete sisestamist ja käitlemist nagu patsiendid, kindlustusfirmad, diagnoosid, haiguslood, osutatud teenused, arstid, retseptid jne. Sellele lisaks tuleb tegeleda ka aruannete, arvete, tšekkide genereerimise ja trükkimisega, ning liidestuda kolmandate osapooltega (meditsiiniga seotud valitsuse või valdkonnaspetsiifiliste infosüsteemidega). Vastavalt eelmainitud aladele saab *Raintree* domeeni jagada väiksemateks alamdomeenideks, näiteks patsiendiga seotud demograafilised andmed, patsiendi kindlustus, aruannete koostamine, haiguslugude väljatrükk jne.

Keskmine programmeerimisülesanne *Raintree*-s seisneb mingi mooduli loomises platvormile või kliendile. Selleks tuleb realiseerida kasutajavaated, andmetega seotud äriloogika ja mõnikord ka dokumendimallid.

2.2 *Raintree* arhitektuur

Raintree meditsiinitarkvara on klient-server rakendus (vt Lisa 1), kus enamus funktsionaalsust on realiseeritud serveripoolses rakenduses (*Raintree* agendis) ja ülejäänud kliendis (*Raintree* klient), mille abil saadakse ligi agendi poolt pakutavatele teenustele. *Raintree* agendis on realiseeritud suurem osa *Raintree* äriloogikast, mida saab vastavalt *Raintree* skriptimiskeelele (ingl. k. *Raintree Scripting Language, RSL*) või *PHP* ja sellega koos kasutatavast *Core Library* [3] raamistikule kohandada ja täiendada.

Vertikaalselt saab *Raintree* domeeni vaadelda kui kolmekihilist süsteemi, milledest esimese moodustab agendis realiseeritud äriloogika, mis püsib muutumatuna kõikide klientide jaoks, teise *RSL*-is realiseeritud moodulid või funktsionaalsed osad *Raintree*-st ja kõige lõpuks kliendispetsiifiline äriloogika. Analoogselt *RSL*-ile on ka *PHP*-s realiseeritud funktsionaalsus jagatud vastavalt põhimooduli ja kliendispetsiifilisse kihti.



Joonis 1. *Raintree* rakenduse lihtsustatud mudel.

2.3 *RSL*-i kirjeldus

Raintree skriptimiskeel on loodud meditsiinitarkvara tootva firma *Raintree Systems, Inc*. poolt 1983. aastal. Tegemist on domeenispetsiifilise keelega, mille ülesandeks on *Raintree* põhiprogrammi võimaluste realiseerimine ja kohandamine – valdkonnaspetsiifiliste kirjete haldamine ja äriloogika kirjeldamine. *RSL*-i on arendatud vastavalt hetkevajadusele – ärimaailm esitab väljakutse, mida peab *RSL* realiseerima. Senine arendus ei ole allutatud ühtsele strateegiale, s.t. *RSL*-i funktsionaalsust on täiendatud programmeerijate parima äranägemise järgi – mis küll töötab ja lahendab püstitatud probleemi, kuid tihtilugu põhjustab vajakajäämisi keele disainis ja seetõttu seab takistusi keele edasiseks arenguks.

*Fowler*i [2] järgi on *RSL*-il kõik karakteristikud, et olla domeenispetsiifiline programmeerimiskeel: see on inimeste poolt kergesti arusaadav, *RSL*-i ladusus väljendub viisis, kuidas lauseid on kombineeritud; tema väljendusvõime on piiratud (*Raintree* domeeni piires), tema fookus on suunatud väikesele domeenile (suur domeen on jagatud paljudeks alamosadeks). Teda saab iseloomustada kui välist domeenispetsiifilist keelt, sest ta erineb keelest, milles ta on implementeeritud (*Delphi*) ja tema süntaks on kohandatud.

Lisaks saab *RSL*-i iseloomustada kui multi-paradigmaatilist keelt, sest *RSL* on ühtlasi veel imperatiivne struktureeritud modulaarne keel.

2.4 *RSL*-i alamdomeenid

Tinglikult saab *RSL*-i jagada alamdomeenideks, kus tema süntaksis leiduvad käsud ja funktsioonid seonduvad kindlate valdkondadega. Järgnevalt on ära toodud peamised valdkonnad:

Showquery	*MySQL* päringute koostamine, tulemuste töötlemine ja vormindamine
Report	Aruannete koostamine ja vormindamine
Overlay	Mallipõhise dokumendi koostamine ja vormindamine
UI	Kasutajaliidesega seotud funktsioonid
Scheduler	Kalender
COR records	*Raintree*-sse sisseehitatud baaskirjetüübid
Custom records	Kohandatavad kirjetüübid
EMR	*Electronic Medical Records* – kombinatsioon baaskirjetüübist ja kohandatavast kirjetüübist
Ledger	Raamatupidamisega seotud funktsioonid
Printer	Printimisega seotud funktsioonid
Utilities	Erinevad utiliidid
File	Failioperatsioonid

Tabel 1. *RSL*-i peamised alamdomeenid.

Nendest ülalmainitud domeenidest on baaskirjetüübid ja kohandatavad kirjetüübid abstraktsed domeenid, milledest esimesed jagunevad konkreetseteks sisseehitatud, muutumatu struktuuriga baaskirjetüüpideks ja teised vabalt valitava struktuuriga kohandatavateks kirjetüüpideks.

2.5 *Pascal*-i ja *BASIC*-u mõjutused

Kuna esialgselt oli *Raintree* põhirakendus kirjutatud programmeerimiskeeles *Pascal*, siis sellest keelest on tulnud mõned *RSL*-i võtmesõnad.

Eraldajad (*ingl. k. special symbols*): `+`, `-`, `*`, `/`, `=`, `<`, `>`, `.`, `(`, `)`, `[`, `]`, `<>`, `<=`, `>=`, `:=`, `|`, `^`, `:`, `;`.

Võtmesõnad (ingl. k. *word symbols*): `and`, `begin`, `case`, `else`, `end`, `goto`, `if`, `mod`, `or`, `repeat`, `set`, `then`, `until`, `var`.

Aritmeetilised funktsioonid: `abs`, `round`, `sqr`, `sqrt`, `trunc`.

Sõnefunktsioon: `chr`.

Sisend-väljundfunktsioonid: `input`, `write`. [4]

Laenud *BASIC*-keelest:

Võtmesõnad: `call`, `loop`, `next`, `rem`, `reset`, `return`, `tron`, `troff`.

Sõnefunktsioon: `pos`.

Sisend-väljundfunktsioonid: `close`, `cls`, `line`, `log`, `print`, `run`, `sound`, `type`, `view`, `wait`, `window`. [5]

2.6 *RSL*-i süntaks

2.6.1 Modulaarsus

RSL-i skripte saab jagada failideks, mida saab `uses`-võtmesõna abil kokku liita ja seejärel `call`-võtmesõnaga teistest failidest kasutada. Teine võimalus on kasutada käsku `callform`, mis käivitab skripti seest teise skripti. Skriptifaili-siseselt saab funktsionaalsust jagada globaalseteks ja lokaalseteks funktsioonideks. Teine, kuid iganenum võimalus on kasutada `goto`- ja `return`-paari.

2.6.2 Andmetüübid

Raintree skriptimiskeeles on kasutusel järgmised andmetüübid:

1) Täisarvud

2) Ujukoma-arvud

3) Sõned

4) Boolean-tüüpi muutujad

2.6.3 Muutujad ja nende väärtustamine

Muutujaid on *RSL*-is mitut liiki. Laias laastus võib jagada neid kolmeks: nn. skriptimuutujad, kohandatavate tüüpkirjete muutujad, alamdomeenidega (baaskirjetüüpidega) ja seotud muutujad ja globaalsed süsteemimuutujad. Kahte viimast võib nimetada ka agendimuutujateks. Muutujaid saab *RSL*-is väärtustada `set`-võtmesõna abil.

Skriptimuutujaid saab väärtustada lihtsa omistamise teel:

```
// väärtustame muutuja
set %muutuja = "minu"
```

```
// väärtustame uue muutuja sõnede kokkuliitmise teel
set %uus_muutuja := %muutuja . "sõne"
```

Kui muutuja omistamisega kaasneb mingi avaldis (nagu ülaltoodud näites sõnede kokkuliitmine), siis tuleb omistamine teha kasutades operaatorit „`:=`". Teine võimalus on ümbritseda kogu avaldis sulgudega:

```
// väärtustame uue muutuja sõnede kokkuliitmise teel
set %uus_muutuja = (%muutuja . "sõne")
```

Kohandatavate kirjetüüpide- ja alamdomeenimuutujate (alg)väärtustamine on märksa keerukam. Korraga saab olla aktiivne ainult üks kirje konkreetsest (baas)kirjetüübist, mistõttu tuleb kirje muutujate kättesaamiseks eelnevalt ette anda kirje tüüp ja konkreetse kirje võti:

```
usefile "ANDMEFAIL"
index "INDEKS1"

set &kirje:muutuja1 = "Muutuja1 sisu"
```

Kui seda kirjet veel andmebaasis ei ole, loob *Raintree* põhirakendus selle kirje *MySQL* tabelisse nimega „ANDMEFAIL_KIRJE", võtmega „INDEKS1" ja väljale „muutuja1". Hiljem saab seda väärtust omistada näiteks skriptimuutujale (eeldusel, et kasutatakse sama kirjetüüpi ja võtit):

```
set %kirje_muutuja1 = &kirje:muutuja1
```

Nüüd on `%kirje_muutuja1` väärtus „Muutuja1 sisu". Kohandatava kirjetüübi struktuur on vabalt valitav. Siiski võib juhtuda, kirjetüübi väli võib kokku langeda *MySQL*-i võtmesõnaga (ingl. k *reserved word*) – sel juhul ümbritseb *Raintree* agent need päringu koostamisel vajalike kursiivülakomadega.

Alamdomeenide muutujate väärtustamine on analoogne kohandatavate kirjetüüpidega seotud muutujate väärtustamisele, kuid sel juhul on teatud võtmega kirje aktiivseks muutmiseks, lugemiseks ja kirjutamiseks käsustik, mis on iga *Raintree* baaskirjetüübi jaoks erinev.

Oletame, et meil on *PATIENT*-nimelises tabelis kirje võtmega „0000001“ ja me tahame teada patsiendiga seotud välju, siis kasutame:

```
setpat "0000001"
set %eesnimi = pfirst
set %perekonnanimi = plast
```

Antud juhul on tegemist kitsa domeeniga: `setpat` käsk positsioneerub kindla *PATIENT* tabeli kirjele, `pfirst`, `plast`, `p*` muutujate arv on lõplik ja need väljendavad kindla kirjetüübi (antud juhul *PATIENT*) välju. Ka nende kirjete muutmise jaoks on *Raintree*-s kindlad käsud:

```
writepfirst "Jüri"
writeplast "Kask"
```

PATIENT-tabeli väljade muutmise käsud on analoogsed „`p*`“ muutujatega, kuid nende käskude prefiksiks on „`writep`“.

2.6.4 Funktsioonid ja käsud

Raintree skriptimiskeeles jaotuvad instruktsioonid käskudeks ja funktsioonideks. Nende põhiline erinevus seisneb kirjapildis – viisis, kuidas parameetreid neile ette antakse.

Funktsiooni süntaks:

```
set %muutuja := funktsioon(argument1, argument2, argumentN)
```

Käsu süntaks *RSL*-is:

```
käsk argument1 argument2 argumentN
```

Erinevalt funktsioonidest ei tagasta käsud tavaliselt midagi, kuid *Raintree*-s teatud kindlate käskude korral seda tehakse.

Käsk mis ei tagasta väärtust – sõnumi dialoogi näitamine:

```
message "Pealkiri" "Esimene rida." "Teine rida."
```

Käsk, mis tagastab väärtuse – faili olemasolu kontrollimine:

```
set %fail_olemas := FILEEXIST "c:\fail.txt"
```

Muutuja `%fail_olemas` sisaldab kas väärtust „Y“ või „N“, vastavalt sellele kas argumendiga antud fail eksisteerib või ei.

Funktsioonid *Raintree*-s näevad välja nagu käsud, kuid selle vahega, et nende argumentide loetelu ümbritsevad sulud ja nende tulemuse peab kindlasti salvestama muutujasse, sest muidu käsk ei tööta:

```
set %juhuarv := RAND(1,5)
```

Muutuja `%juhuarv` sisaldab juhuarvu väärtusega 1, 2, ..., 5.

Kolmas meetod, kuidas mõningate *RSL*-i alamdomeenide käske kirjutatakse, on järgnev:

```
showquery.setandopen = "SELECT * FROM patient"
set %kirjete_arv = showquery.count
```

`Showquery` käsud on viis *RSL*-is *MySQL*-i päringute ja mõningate lisavõimaluste kasutamiseks. Eelnevas näites on päringutega seotud käsud koondatud `showquery` domeeni, mis hõlmab endas nii käske kui ka muutujaid.

2.6.5 Massiivid

RSL-is massiivi kui andmetüüpi ei eksisteeri, kuid selle asendamiseks on kasutusel pseudomassiivid, mida saab teatud viisil muutujatele nimesid andes väärtustada ja kasutada.

Näide kahemõõtmelisest massiivist:

```
set %i = 1
set %j = 1
set %massiiv_[%i]_[%j] = "esimene element"
```

```
set %j = 2
set %massiiv_[%i]_[%j] = "teine element"
```

Tegelikkuses on väärtustatud muutujad nimedega `%massiiv_1_1` ja `%massiiv_1_2`, kuid nendes muutujanimedes esineb seaduspära, mis võimaldab neid kasutada kui massiivi. Kuna massiiv sisaldab alati teatud hulk muutujaid, siis tuleb arvet pidada ka massiivi elementide arvu üle.

Näide kolmeelemendilise massiivi läbimisest:

```
set %massiivi_element1 = "esimene"
set %massiivi_element2 = "teine"
set %massiivi_element3 = "kolmas"
set %massiivi_elemendid = 3

set %i = 1
repeat
  set %element = %massiivi_element[%i]
  set %i := inc(%i)
until %i > %massiivi_elemendid
```

3. *PHP* ja *Core Library* (raamistik) *Raintree* rakenduses

3.1 *PHP* integreerimine *Raintree* agendiga

Raintree skriptimiskeel oli esialgselt mõeldud *Raintree* põhirakenduse standardse äriloogika muutmiseks. Tihtilugu on klientide vajadused aga erinevad ja nii järjest enam programmeeriti *RSL*-is koodi, mis kattis üle põhirakenduse koodi. See aga viis punktini, kus *RSL*-i vahendid ja jõudlus ei suutnud enam rahuldada süsteemi kasvavaid vajadusi. Teiseks muutus *RSL* järjest keerukamaks, mis muutis selle ülalpidamise kulukamaks. [6]

Tuli võtta kasutusele uus keel, mis vastaks järgmistele kriteeriumitele:

- Keel peab olema üldsusele tuntud;
- Keel peab võimaldama suhtlemist põhirakendusega (agendiga);
- Keelel peab olema objekt-orienteeritud programmeerimise tugi, mis võimaldab ehitada keerukaid ja laiendatavaid platvorme;
- Keelel peab olema minimaalne mõju serveri ressurssidele (protsessori tööajale ja mälu kasutamisele) ja kõrge täitmiskiirus;

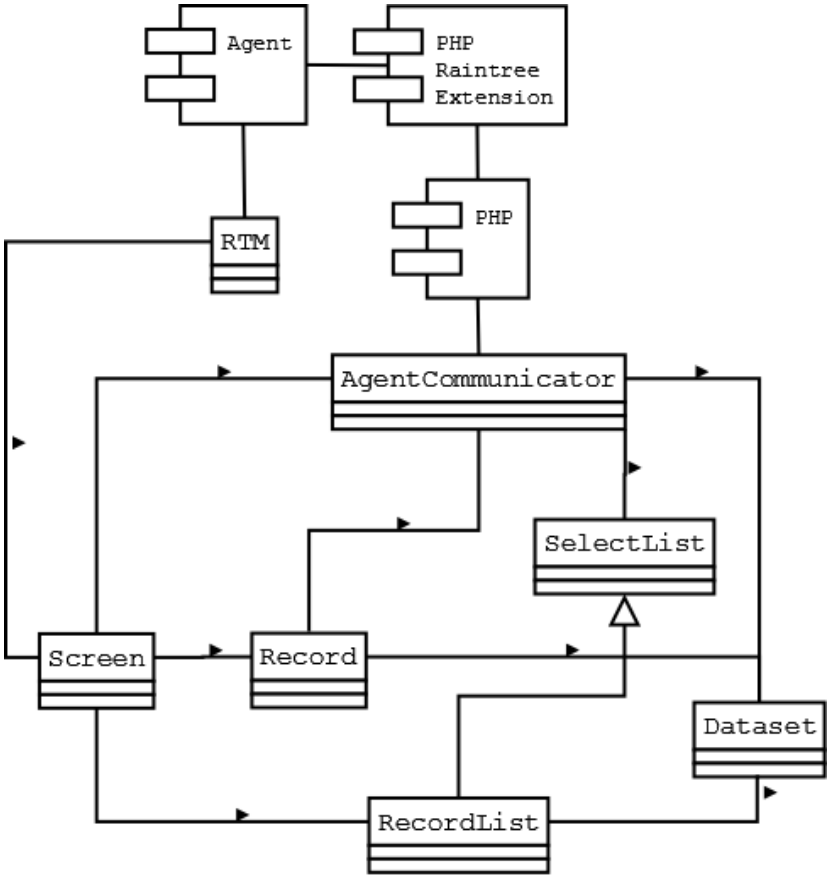
Üks tähtsamaid punkte siinkohal on suhtlus agendiga, kusjuures uus keel peab suutma suhelda *RSL*-iga, sest uut keelt hakatakse kasutama juba eksisteerivates süsteemides ja funktsionaalsuse ümberkirjutamine ei ole jõukohane. Seetõttu asendaks uues keeles kirjutatud funktsionaalsus neid ajakriitilisi kohti, kus *RSL*-i süntaksianalüsaator osutub liiga aeglaseks või kohti, kus lahenduse keerukus on piisavalt suur, et vajada kõrgema taseme keele võimalusi. [6]

Eelnevatest probleemidest ajendatuna valiti selleks uueks keeleks 2006. aastal *PHP*.

Põhilisteks ajenditeks olid:

- Laialt tuntud keel suure kasutajaskonnaga;
- Igati laiendatav mootor vajaminevate lisavõimaluste jaoks;
- *PHP*-l on palju avatud lähtekoodiga silumismootoreid, mida saab integreerida infosüsteemi ja koodiredaktoriga; [6]

Lahendusena programmeeriti *PHP5*-le laiendus, mis võimaldas *PHP* skriptidel suhelda agendiga.



Joonis 2. Seosed *Raintree* põhirakenduse, *PHP* ja *Core Library* vahel.

3.2 *PHP* ja agendi integratsiooni tööpõhimõte

Kuna *Raintree* agent on programmeeritud *Delphi*-s ja *PHP C*-s, siis nende kahe keele ühendamiseks peab olema mingi sild, mida mõlemad oskavad kasutada teisele teadete saatmiseks ning tulemuse saamiseks. *PHP* poole pealt on sellise silla realiseerimiseks otstarbekas kasutada *PHP* laienduste tuge, mis võimaldab programmeerimiskeeles *C* defineerida uusi kasutatavaid funktsioone keelele *PHP*. Selleks on *Raintree* tarbeks loodud *PHP* laienduse teegis `fs.dll` funktsioon `fs_command($params)`. Läbi selle funktsiooni saadetakse serialiseeritud kujul kokkulepitud ühtses formaadis teateid põhirakendusele. Teate saamisel sooritab põhirakendus vastavad operatsioonid ning tagastab *PHP*-le serialiseeritud kujul vastuse, mida *PHP* tõlgendab vastavalt saadetud teatele ning tegutseb sellest lähtuvalt edasi. Kahe keele ühendamiseks kasutatava silla olemus seisneb mingitele kokkulepitud mälupesa aadressidele informatsiooni kirjutamises ja lugemises mõlema osapoole poolt. [3]

Põhirakenduse ja *PHP* skriptimootori vahelist suhtlust koordineeritakse lõimede abil, milledest ühte võib nimetada põhirakenduse lõimeks ja teist *PHP*-lõimeks. Põhirakenduse lõim eksisteerib kogu selle jooksmise aja, kuid *PHP*-lõim luuakse siis, kui põhirakenduses tekib vajadus *PHP* skriptide jooksutamise järele. *PHP* skripti jooksmise ajal läheb lõime täitmise kord üle *PHP*-lõimele, mis siis vastavalt vajadusele pöördub põhilõime poole, kui tekib vajadus utiliseerida *PHP* laienduses defineeritud funktsioone. Kui põhilõim on *PHP* suhtes oma töö täitnud, läheb lõime täitmise järg tagasi *PHP*-lõime kätte kuni skriptide jooksmise lõpuni.

3.3 *Core Library* raamistik

Raintree Core Library [7] raamistiku põhiidee on teatud tunnuste alusel jagada kasutajavaated, mudelid ning kontrollerid moodulitesse. Raamistiku suhtlemine põhirakendusega käib läbi keskse klassi (mis utiliseerib *PHP* laienduses defineeritud funktsioone), mida ülejäänud komponendid saavad kasutada. Raamistik toetab *MVC* (ingl. k. *model-view-controller*) arhitektuuri, mille kihid, välja arvatud vaade, on realiseeritud raamistiku erinevate osade poolt. Vaated on defineeritud põhirakenduse poolt ning eeldatakse, et need on salvestatud vastavatesse failidesse, mida tähistatakse lühendiga *RTM (Raintree cusToM screen)*. [3]

Core Library baasklassid on loodud eesmärgiga olla kasutusvalmid ilma laiendamiseta, juhul kui ei ole tarvis äriloogikat muuta ega juurde lisada. Kuid kui seda peaks tarvis minema, siis neid klasse on võimalik omakorda spetsiifilise äriloogika lisamiseks laiendada. [3]

3.4 *Core Library* baasklassid

3.4.1 `AgentCommunicator` – suhtlus põhirakendusega

Läbi `AgentCommunicator` klassi kulgeb kogu *Core Library* raamistiku ja *Raintree* põhirakenduse vaheline suhtlus, seega selle klassiga on otseselt või kaudselt seotud kõik ülejäänud raamistiku baasklassid. Selles klassis leiduvad funktsioonid on pakendiks (ingl. k. *wrapper*) *Raintree*-spetsiifilise *PHP* poolt pakutavatele funktsioonidele. Vastavalt agendi ja *PHP* vahelise integratsioonile kutsutakse välja serialiseeritud parameetritega *PHP* funktsioone põhirakenduse poolt pakutava funktsionaalsuse kasutamiseks ja seejärel edastatakse raamistiku kõrgematesse tasemetesse välja kutsutud funktsioonide väärtused. [3]

Näide *RSL*-i käsu andmiseks läbi *PHP*:

```
$agent = AgentCommunicator::getInstance();
$agent->fs('setpat "0000001"');
```

3.4.2 `Record` ja `Dataset` – kirjete haldamine

`Record` klassi eesmärk on pakkuda üksust, mille läbi saab teha kõiki vajalikke toiminguid ühe kirjega andmebaasist. Nendeks toiminguteks on uue kirje lisamine, olemasoleva laadimine andmetabelist, väljadel oleva info muutmine ning salvestamine või kustutamine. Andmebaasis oleva kirje laadimiseks, salvestamiseks ja kustutamiseks kutsutakse välja sobivale kujule viidud kirje parameetritega klassis `AgentCommunicator` defineeritud meetodeid. [3]

Näide kohandatava kirje muutmisest:

```
$rec = Record::getInstance('ANDMEFAIL', 'INDEKS1', 'KIRJE');
$rec->setField('muutuja1', 'Muutuja 1 sisu');
$rec->save();
```

`Dataset` kujutab endast kirjete konteinerit, mis sisaldab endas teatud tingimuste vastavaid kirjeid ehk teisisõnu teatud hulka `Record` klassi eksemplaridest (ingl. k. *instance*). Eelnevast tulenevalt on vaadeldavas klassis tarvis meetodeid järgnevate operatsioonide jaoks: kirje tüübi defineerimine otse konstruktoris, filtrite lisamine ja eemaldamine. Lisaks on vaja meetodeid kirjete otsimise päringu tegemiseks, ühe kirje objekti kättesaamiseks ning tingimustele vastavate kirjete unikaalsete võtmete tagastamiseks. *MVC* mustri järgi paigutuvad nimetatud komponendid *Model* ehk mudeli kihti. [3]

3.4.3 `SelectList` ja `RecordList` – dünaamilised loendid

Raamistikus on `SelectList` ja `RecordList` klassid dünaamiliste loendite defineerimiseks ja täitmiseks. `SelectList` pakub põhifunktsionaalsust loendite loomiseks, sealhulgas veergude defineerimist, ridade lisamist ja eemaldamist, sorteerimist ning sündmuste haldurite lisamist. `RecordList` on aga `SelectList` klassi laiendus, mis on juba konkreetselt realiseeritud andmebaasi kirjete näitamiseks `Dataset` klassi abil. `SelectList`-i sisu saab näidata eraldiseisva graafilise loendina (ingl. k. *list box*) või *RTM*-vaate ele-

mendina. *MVC* mustri järgi paigutuvad nimetatud komponendid vaate ja ka kontrolleri kihti. [3]

3.4.4 `Screen` – ekraanide haldus

Raamistikus on kasutajaekraanide näitamiseks kasutusel klass `Screen`, mis pakub selleks vajalikku funktsionaalsust. *MVC* arhitektuurimustri järgi võib vaadeldava klassi paigutada *controller*-kihti. Põhiülesanneteks on käsitletava raamistiku komponendil kasutajaekraanide näitamine koos sisaldavate andmetega, kasutaja tegevustele reageerimine ning loendikastides kirjete listide haldamine. Põhiidee `Screen`-klassi puhul on automaatselt üks-ühele-seosega siduda ekraan ja kirje. [3]

Näide kasutajaekraani kuvamisest:

```
$scr = Screen::getInstance('KIRJE');
$rec = Record::getInstance('ANDMEFAIL', 'INDEKS1', 'KIRJE');
$scr->setData($rec);
$scr->show();
```

Ülaltoodud näites loodi `Screen`-klassi eksemplar „KIRJE“-nimelise vaate näitamiseks, kusjuures see vaade seoti konkreetse kohandatava kirjega, mille andmefail on „ANDMEFAIL“ ja võti “INDEKS1“.

3.4.5 `Module` – moodulite loomiseks

Suuremates ärilahendustes võib olla palju erinevaid kasutajavaateid ning menüüsid, millega omakorda on seotud mingid dünaamilised kirjete loendid. Otstarbekas oleks teatud sarnaste tunnuste alusel erinevate kasutajavaadete, kirjete, ekraanide ja loendite haldamisega tegelevad klassid koondada erinevatesse moodulitesse. Mooduliks loetakse sellist üksust, mis sisaldab endas erinevaid kasutajavaateid ning kirje, ekraanide, andmehulkade ning loendite klasse. [3]

Core Library-s näeb mooduli failistruktuur välja järgmine:

```
ModuleName/
     Screens/
     Records/
```

```
RTM/
```

Iga mooduli jaoks on moodulite kataloogis olemas sellele vastav kaust `ModuleName`. Selles kaustas on alamkaustad `Screens`, `Records` ja `RTM`, milles vastavalt asuvad ekraanide klassid, kirjete klassid ning vaated. Olgu kokku lepitud, et mooduli deklareerimiseks peab mooduli kaustas eksisteerima fail `Module.php`, milles asub klass `[ModuleName]`, mis laiendab allpool kirjeldatud üldist mooduli haldamise klassi `Module`. Mooduli poolt kasutatavad `Dataset` ja `SelectList` alamklassid paigutatakse samuti otse mooduli juurkausta. [3]

3.4.6 *Core Library* kasulikkus

Alampeatükis 3.4.5 kirjeldatud mooduli struktuur võimaldab organiseerida ja hallata mooduliteks jaotatud *Raintree* rakenduse funktsionaalsust. Kuna üheks keele valiku kriteeriumiks oli skriptimiskeele objekt-orienteerituse tugi, siis seda ongi kliendispetsiifiliste lahenduste loomiseks raamistiku disainimisel ära kasutatud.

Näide kliendispetsiifilisest mooduli struktuurist:

```
ModuleName/
     Screens/
     Records/
     RTM/
     ClientExtension/
          Screens/
          Records/
          RTM/
```

`ModuleName`-nimelises kaustas asuva mooduli baasfunktsionaalsus on laiendatud kliendispetsiifilises laienduses nimega `ClientExtension`, mis on samuti mooduli struktuuriga, kuid sisuliselt on tegemist baasmooduli laiendusmooduliga. Laiendusmoodulite loomise peamine eesmärk on platvormi (sõltub kindlast meditsiinivaldkonnast, nagu kiropraktika, radioloogia, taastusravi, bariaatria jne) või kliendispetsiifilise lahenduse realiseerimine. Tehniliselt seisneb see baasfunktsionaalsuse (klasside ja funktsioonide) ülekatmises või laiendamises.

Teisalt omab *Core Library* raamistik kohandatavust ka *Raintree* agendi ja andmebaasimootori suhtes. Näiteks varasemad *Raintree* agendid kasutasid andmete hoidmiseks B-puu (ingl. k. *B-tree*) struktuuri, uuemad aga *MySQL* andmebaasimootorit.

4. *RSL*-i ja *PHP* võrdlus ehk domeenispetsiifilise keele võrdlus universaalse keelega

4.1 Süntaks ja keele elemendid

RSL-i ja *PHP* süntaks on sarnane, kuna mõlemad on interpreteeritavad skriptimiskeeled. Nende keelte alged on aga erinevad (*RSL* pärineb *Pascal*-ist ja *PHP C*-st [8]), ja sellest tulenevalt esineb nendes keeltes erinevusi. *RSL*-i ja *PHP* (koos *Core Library*-ga) suurim erinevus on domeenispetsiifiliste elementide olemasolu *RSL*-is. *PHP*-s on seevastu arendatud universaalsele keelele omaseid võimalusi nagu massiivid ja objekt-orienteeritus.

Järgnevalt on esitatud *RSL*-i ja *PHP* keele elementide võrdlus:

Keele element	***RSL***	***PHP***
Andmetüübid	Täisarv, ujukoma-arv, sõne, *boolean*	Täisarv, ujukoma-arv, sõne, *boolean*, massiiv, objekt, ressurss, NULL
Muutujad	%muutja, &muutuja, agendimuutuja	$muutuja, $GLOBALS..., $_POST...
Konstandid	Puudub	Olemas
Avaldised	Olemas, kuid piiratud võimalustega	Rikkalik avaldiste tugi
Operaatorid	Aritmeetilised-, omistamis-, võrdlus-, loogilised- ja sõneoperaatorid	Aritmeetilised-, omistamis-, võrdlus-, loogilised- sõne-, bitipõhised-,veakontrolli-, inkremendi-/dekremendi-, massiivi ja sõneoperaatorid
Massiivid	Pseudomassiiv	Suur hulk funktsioone massiivide manipuleerimiseks
Tingimuslaused	`if-then-else, switch`	`if-then-else, switch`
Silmused	Universaalne: `repeat/until` Spetsiifilised: `loop emr/ next emr`, `loopfile/ next file`, `loop ledger/next ledger` ...	`do/while, while, for, foreach`

Muud kontrollstruktuurid	Puuduvad	`break`, `continue`, `return`
Kasutaja funktsioonid	Sarnane, kuid võimaldab mitut samaaegset väljundparameetri tagastamist, argumentide arv täpne	Sarnane, võimaldab argumentide puudumisel väärtustada neid vaikeväärtustega
Objekt-orienteeritus	Puudub	Rikkalik tugi
Erindite töötlus	Teatud muutujas näeb viimase käsu/funktsiooni täitmisel tekkinud vea teadet	Tugi olemas
Domeenispetsiifilised keelekonstruktsioonid	Rikkalik tugi	Minimaalsed *Raintree*-spetsiifilised funktsoonid

Tabel 2. *RSL*-i ja *PHP* keele elementide süntaksi võrdlus [10].

Laias laastus võib erinevused jagada kolme suurde kategooriasse:

1) Erinevused mis on tingitud keele mõjutustest just süntaksi osas – nimelt faktist, et *RSL* pärineb *Delphi*-st ja *PHP C*-st;

2) Erinevused, kus keele elemendid on mõlemas võrreldavas keeles olemas, kuid implementeeritud erinevalt;

Näiteks omistamine, kus *PHP*-s üks ja ainus operaator selleks on „=“, kuid *RSL*-is peab eristama avaldist tavalisest omistamisest, kasutades esimesel juhul operaatoreid „**:**=“ ja „**()**“, teisel aga „=“. Samuti ka silmused, kasutajafunktsioonid ja erindite töötlus.

3) Erinevused, mis on tingitud keele disainist ja eesmärgist.

Näiteks *PHP* on disainitud kui universaalne programmeerimiskeel, omades mitmekesiseid keele elemente ja võimalusi. *RSL*-s esineb universaalsele programmeerimiskeelele omaseid elemente minimaalselt, kuid seevastu on sellel keele domeenispetsiifiliste konstruktsioonide tugi.

4.2 Domeenispetsiifilisus

Kombineerituna rakenduse teegiga (ingl. k. *application library*), võib mistahes universaalne programmeerimiskeel käituda domeenispetsiifilise keelena. Teegi rakendusliides (ingl. k. *application programming interface*, *API*) moodustab domeenispetsiifilise sõnastiku klassidest, meetoditest ja funktsioonidest, mis on kasutatavad mistahes universaalse prog-

rammeerimiskeelega. [9] Selles suhtes võib *Raintree* rakenduses *PHP*-d ja *Core Library*-t näha domeenispetsiifilise keelena.

Järgnevalt analüüsime *RSL*-i ja *Core Library*-t domeenispetsiifilisuse aspektist:

PHP skript, mis väärtustab teatud patisendi ees- ja perekonnanime muutujatesse:

```
$patsient = new CoreRecord(CoreRecord::T_PATIENT, '0000005');
$eesnimi       = $patsient->getField('first');
$perekonnanimi = $patsient->getField('last');
```

Sama funktsionaalsus kirjutatuna *RSL*-is:

```
setpat "0000005"
set %eesnimi       = pfirst
set %perekonnanimi = plast
```

Ülaltoodud näites on selgelt näha vahe üldkasutatava keele ja domeenispetsiifilise keele vahel:

1) *PHP*-s sama funktsionaalsuse kirjutamiseks kulub rohkem tähemärke kui *RSL*-is;
2) Kuna universaalne keel toetab erinevaid abstraktsioonistruktuure, siis seda on kasutatud kõikide baaskirjetüüpidele ligipääsemiseks klassi `CoreRecord` abil, mis aga muudab selle õppimise ja kasutamise raskemaks. Muidugi võiks igale `CoreRecord`-klassi kirjetüübile laiendada spetsiifilise klassi – näiteks spetsiaalne klass patsiendi baaskirjetüübile. Seevastu `setpat` käsk ja `pfirst`, `plast` muutujad on piiratud väljendusvõimega – nad on mõeldud just patsiendi kirje käitlemiseks;
3) Abstraktsioonide puudumine *RSL*-is võimaldab keele elementidel olla fokusseeritud kitsale domeenile, milleks antud näite puhul oli patsiendi kirje;

4.3 Laiendatavus

Core Library on disainitud lihtsasti laiendatavaks, mida täpsemalt kirjeldab alampeatükk 3.4.6. Me saame piisavalt suure täpsusega laiendada või üle defineerida klasse ja meetodeid, mida võimaldab *PHP* objekt-orienteerituse tugi. *RSL*-is sarnast laiendatavust võimaldab saavutada skriptifaili paigutamine samal tasemel olevasse *custom*-nimelisse kausta.

Siiski on selline laiendamisevõimalus ühetasandiline – pärast *custom*-kausta enam edasi laiendada ei saa.

Näide:

```
minuskript.frm
     custom\minuskript.frm
```

Oletame, et failis `minuskript.frm` sisaldub mingi äriloogika või funktsionaalsus, mida on vastavalt kliendi spetsiifikale tarvis kohandada. Selle tarbeks tehakse esialgsest skriptist koopia, paigutatakse `custom`-kausta ja muudetakse vastavalt vajadustele. Kui *Raintree* põhirakendus kutsub välja teatud kindlast asukohast faili `minuskript.frm`, siis käivitatakse muudetud skriptifail. Sellise lähenemise negatiivne külg on see, et baasmoodulis tehtud hilisemad muudatused ei kajastu enam kliendispetsiifilises laiendatud skriptis, ning ühilduvus teiste moodulitega pärast uuendusi võib olla probleemne.

5. *Raintree-s* kasutatavate skriptimiskeelte edasiarendus

5.1 *RSL*-i ja *PHP* kasutamine *Raintree*-s

Raintree-s on kasutusel skriptimiskeeled, millel on omad eelised ning puudused. Tänaseks selgeks saanud põhjused, millal kasutada üht või teist. Nimelt kui on tegemist lihtsa skriptiga või mooduliga (näiteks mingi aruanne, teatud andmete küsimine) siis kasutatakse pigem *RSL*-i, sest seda koodi on lihtsam kirjutada ja seetõttu valmib lahendus kiiremini. Teisalt on *RSL*-i lihtsuse tõttu ka sellel suurem organisatsioonisisene kasutajatebaas. *PHP*-d on kasutatud ja kasutatakse keerulisemate moodulite arendamisel, mis saavad kõige rohkem kasu *Core Library* moodulite laiendamise võimalustest. Võrreldes *RSL*-iga nõuab *PHP* koos *Core Library*-ga oma keerukuse võimaluste tõttu suuremat programmeerimisoskust, siis paljud endised *PHP*-s arendatud moodulid on kirjutatud *RSL*-i peale ümber – isegi kui need vajavad laiendatavuse paindlikkust, mida pakub *Core Library*. Teine põhjus, miks eelistatakse *RSL*-i *PHP*-le, on kiirus. Kiiruse vahe tuleb viisist, kuidas ühes või teises keeles realiseeritud skripte käivitatakse. Sellal, kui agent saab asuda *RSL*-is kirjutatud käsu täitmisele (sest *RSL*-i interpretaator asub agendis), siis *PHP* puhul tuleb kõigepealt lähtestada *PHP* skriptimootor ja *Core Library*, ning seejärel saab asuda skriptis olevate funktsioonide täitmise juurde.

Siiski on säilinud *Raintree*-s *PHP* kindel kasutusvaldkond, milleks on erinevad liidesed, mille abil *Raintree* rakendus suudab liidestuda teiste infosüsteemidega. Siiski kaugenetakse sellest põhilisest eemärgist, miks *PHP Raintree*-s kasutusele võeti – objekt-orienteeritus laiaulatuslike platvormide ehitamiseks.

5.2 *RSL*-i ja *PHP* eelised ning puudused

Järgnevas tabelis on eelneva põhjal kokkuvõtlikult ära toodud *RSL*-i ja *PHP* eelised ning puudused:

Skriptimiskeel	**Eelised**	**Puudused**
RSL	1) Väike semantiline kaugus domeenist 2) Üldiselt kulub funktsionaalsuse kirjeldamiseks vähem tähemärke 3) Suurem firmasisene kasutajaskond	1) Ebaühtlane süntaks 2) Nõrk massiivide tugi 3) Olematu objekt-orienteerituse tugi 4) Kohmakas kasutajafunktsioonide kasutamine
PHP + Core Library	1) Objekt-orienteerituse tugi 2) Massiivide tugi 3) Erindite tugi 4) Palju laiendusi 5) Lai firmaväline kasutajaskond	1) Suur semantiline kaugus domeenist 2) Üldiselt kulub funktsionaalsuse kirjapanekuks rohkem tähemärke 3) Teatud juhtudel aeglasem võrreldes *RSL*-iga

Tabel 3. *RSL*-i ja *PHP* eelised ning puudused.

Ülaltoodud tabelis ei võrrelda nende skriptimiskeelte ühiseid eeliseid, sest ühe või teise keele parendamise seisukohast on need ebaolulised. Pigem keskendume just erinevustele – eeliste ületoomisele ja puuduste kõrvaldamisele, millede realiseerimine sõltub kõige rohkem viisist, kuidas *RSL* ja *PHP* on *Raintree*-s implementeeritud.

RSL on skriptimiskeel, mis on arenenud nullist. *PHP* puhul on selle kasutuselevõtmine *Raintree*-s saanud võimalikuks keele laiendamise läbi. *Mernik, Heering* ja *Sloane*'i [9] järgi saame võrrelda nende implementatsioonide puudusi ning eeliseid:

Implementatsioonimeetod	**Eelised**	**Puudused**
DSL-i interpretaator	1) *DSL*-i süntaks saab olla lähedane esitusviisile, mida kasutavad domeenieksperdid 2) Võimalik on hea veateadete esitamine 3) Võimalik on domeenispetsiifiline analüüs, verifikatsioon,	1) Arenduse jõupingutus on suur, sest tuleb implementeerida keerukeeleprotsessor 2) *DSL* tuleb arvatavasti disainida nullist, mis viib sageli seosetute kavanditeni võrreldes olemasolevate keelte kasutamisega

	optimisatsioon, parallelisatsioon ja transformatsioon	3) Keele laiendamist on raske realiseerida, sest paljud keeled ei ole disainitud laiendamist silmas pidades
Universaalse keele kasutamine	1) Arendamise jõupingutus on tagasihoidlik, sest saab taaskasutada olemasolevaid implementatsioone 2) Tihti saavutatakse võimsam keel, sest paljud võimalused tulevad „tasuta“ 3) Peremeeskeele infrastruktuuri saab taaskasutada (arenduse ja silumise keskkonnad) 4) Kasutajate koolituskulud võivad olla madalamad, sest paljud kasutajad juba teavad peremeeskeelt	1) Süntaks ei ole optimaalne, sest paljud keeled ei luba meelevaldset süntaksi laiendamist 2) Olemasolevate operaatorite üledefineerimine võib olla segadust tekitav, kui uus semantika ei oma samu omadusi kui vana 3) Halb veateadete esitamine, sest teated kasutavad peremeeskeele terminoloogiat *DSL*-i oma asemel 4) Domeenispetsiifilisi optimisatsioone ja transformatsioone on raske saavutada, mistõttu see mõjutab efektiivsust eriti funktsionaalsete keelte kasutuselevõtmise korral

Tabel 4. Kahe domeenispetsiifilise keele implementatsioonimeetodi võrdlus [9].

5.3 Lahendus

Arvestades *RSL*-i ja *PHP* implementatsiooni, omadusi ning kasutamist *Raintree*-s, siis kahe senise paralleelse lahenduse asemel oleks tarvis ühte, kus oleksid ühendatud *PHP* objekt-orienteeritus ja *RSL*-i lihtsus. Üks võimalus oleks implementeerida *RSL*-is objekt-orienteeritus, kuid kuna seda keelt ei ole arendatud eriti laiendusi silmas pidades (*RSL*-i on arendatud hetkevajadusi arvestades, teiseks on *RSL* kontekstipõhine), siis see võib päädida terve senise *RSL*-i interpretaatori ümberkirjutamisega. Teiseks jäävad sellise lahenduse puhul kättesaamatuks teised *PHP*-s pakutavad võimalused, näiteks massiivid ja liidestamisel kasutatavad *PHP* laiendused nagu *SOAP* ja *HTTP*. Teine võimalus on edasi arendada *PHP* puhul võetud suunda – keele laiendamist, kuid sellega kaasnevad omad ohud, kus võib tekkida sama olukord mis eelneva *PHP* kasutuselevõtuga – võrreldes *RSL*-iga oli

PHP ja *Core Library* abil kirjutatud kood abstraktsem, keerukam ja domeenist kaugemal, mistõttu kannatas keele kasutuslihtsus. Kasutuslihtsuse puudumise tõttu ei ole suudetud *PHP*-d *Raintree*-s firma-üleselt juurutada, mis on omakorda tinginud olemasolevate moodulite ümberkirjutamist *RSL*-i, sest ei leidu piisavalt arendajaid, kes suudaksid olemasolevaid *PHP*-s kirjutatud mooduleid hallata.

Pidades silmas keele kasutuslihtsust, siis antud olukorra lahendaks *PHP* laienduse edasi arendamine viisil, kus *PHP*-sse tuuakse sisse *RSL*-s kasutatavaid käske. Selleks tuleb:

1) Jaotada kõik *RSL*-i võtmesõnad kasutusvaldkonna või alamdomeeni järgi kategooriatesse.

Näiteks aritmeetilised operaatorid, sõnefunktsioonid, baaskirjetüüpidega seotud võtmesõnad jne.

2) Käia läbi kõik saadud kategooriad ja jätta välja need võtmesõnad, millele leidub *PHP*-s võrdlemisi täpne vaste. Siinkohal ei tohiks kärpida sarnaste, kuid domeenispetsiifilisuse seisukohast vajalikke käske.

Näiteks *RSL*-is on defineeritud funktsiooni vastava sümboli *ASCII* koodi leidmiseks:

```
set %kood := ASCII("A")
```

Selle funktsiooni ekvivalent *PHP*-s on funktsioon nimega `ord()`.

Teine näide on *RSL*-is `showquery` alamdomeeniga seotud käsud. Kuid enamus selle alamdomeeni funktsionaalsust on võimalik asendada *PHP mysql*-i laienduse funktsioonidega. Siiski tuleks laiendada ka *mysql*-i laiendust, sest `showquery` alamdomeen sisaldab ka domeenispetsiifilist funktsionaalsust nagu `showquery.print` või `showquery.show`, mis on vastavalt päringu tulemuse printimiseks või interaktiivse loendi koostamiseks.

3) Tuua *PHP*-sse laiendamiseks üle kõik järele jäänud funktsioonid, käsud ja agendimuutujad, kuid selle erinevusega, et *RSL*-i käsud ja agendimuutujad saavad funktsioonide kuju.

Näide *RSL*-i käsust `setpat` ja laiendatud PHP funktsioonist `getPatient()`:

```
setpat %patsiendi_number
```

```
$patsient = getPatient($patsiendi_number);
```

Kuna *RSL*-is kontrollitakse pärast `setpat` käsu andmist patsiendi olemasolu `SS`-muutja abil (kui `SS = "NOMATCH"`, siis patsienti kirjet ei leitud), siis *PHP* `getPatient()` võiks tagastada patsiendi kirje objekti või `NULL`, vastavalt sellele, kas patsiendi kirje leiti või mitte, mis võimaldab kokkuvõttes mugavamalt kontrollida patsiendi kirje leidumise fakti. Teisalt saame lahti kontekstipõhisest loogikast, kasutades selle asemel objekt-orienteeritud funktsionaalsust. Analoogselt patsiendi baaskirjetüübi näitega tuleks *PHP*-s realiseerida ka teiste baaskirjetüüpide käitlemine.

Agendimuutujad saavad funktsiooni kuju ja neid saab otse küsida *PHP*-s, selle asemel, et kasutada `AgentCommunicator` klassi nende muutujate küsimiseks:

```
$täna = AgentCommunicator::getInstance()->get('today');
```

```
$täna = today();
```

Muutuja nimega `$täna` sisaldab nüüd hetkekuupäeva kujul „kk-pp-aa“, mis on *Raintree*-s vaikimisi kasutatav kuupäevaformaat. Konkreetse näite puhul saab ka standardse *PHP* vahendite abil kirja panna samaväärse funktsiooni kujul `date('m-d-y')`, kuid see on domeenispetsiifilisuse seisukohast üldisem ja tähemärkide arvu poolest pikem, kui `today()`.

Antud lahendus võimaldaks saavutada soovitud tulemuse: domeenispetsiifilisusest tuleva keele lihtsuse ja üldkasutatava keelega kaasnevad võimalused.

PHP laiendamisel ei tule tegeleda probleemidega, mis esinevad iseseisva keele arendamisel, nagu näiteks stabiilsus, töökindlus, jõudlus ja keele disaini ja süntaksi ühtsena hoidmine. Lisaks sellele on *RSL*-i arendamiseks vaja rohkem *Delphi* programmeerijaid, mis aga ei pruugi olla väikesel firmal mõttekas.

Siiski on vaja veel lahendada *RSL*-is kirjutatud moodulite ja skriptide probleem. Kuna *RSL*-is on programmeeritud suur osa *Raintree* funktsionaalsusest, siis tagasiühilduvuse huvides tuleb olemasolev *RSL*-i funktsionaalsus alles jätta, kuid samas tuleb alustada olemasolevate skriptide uue keele peale viimist moodulite kaupa, nagu hetkel tehakse seda *PHP*-s kirjutatud moodulite puhul, mida kliendid *Raintree* rakenduse versiooni uuendamisel saavad kasutama hakata.

Kokkuvõte

Käesoleva töö eesmärk oli välja pakkuda lahendusi ühe või teise *Raintree Estonia OÜ*-s kasutatava skriptimiskeele edasiarendamiseks. Selle eesmärgi saavutamiseks kirjeldati töö esimeses osas domeenispetsiifiliste keelte olemust ja mõningaid karakteristikuid. Teises peatükis vaadeldi *Raintree* põhirakenduse arhitektuuri ja *Raintree* skriptimiskeele osa selles. Samas vaadeldi ka *Raintree* skriptimiskeele elemente ning süntaksit. Kolmandas peatükis keskenduti *Raintree* ja *PHP* põhirakenduse integratsiooni selgitamisele ja *PHP* keelele loodud *Core Library* raamistiku põhiklasside kirjeldamist ning raamistiku kasutamise eesmärki. Neljandas peatükis võrreldi *Raintree* skriptimiskeelt ja *PHP*-d nende domeenispetsiifilisuse, laiendatavuse, süntaksi ja keele elementide aspektist. Viiendas ja viimases peatükis võeti kokku eelnevate osade põhjal *Raintree* skriptimiskeele ja *PHP* eelised, puudused ja kasutamise *Raintree*-s ning jõuti järeldusele, et mõlema keele eeliseid saab ühendada laiendades *PHP* keelt nii, et ta oleks *Raintree* skriptimiskeelega sarnaselt domeenispetsiifiline.

Keelte edasiarendamise eesmärk on tingitud faktist, et olemasolevaid keeli ei kasutata eesmärgipäraselt – kuigi *PHP* võeti kasutusele just objekt-orienteeritud programmeerimise võimaluste poolest lihtsasti laiendatavate ärirakenduste moodulite ehitamiseks, kasutatakse selle asemel Raintree skriptimiskeelt. Põhjus peitus nende kahe keele implementatsioonis, kus *Raintree* skriptimiskeel oma domeenispetsiifilisuses väljendab palju täpsemalt *Raintree* põhirakenduse olemust, võrreldes *PHP*-ga. Kuna tuli ühendada nii objekt-orienteeritus kui ka domeenispetsiifilisus, siis seda on lihtsam saavutada *PHP* keele domeenispetsiifilisuse edasiarendamise teel, sest *PHP* kasutamisega kaasnevad ka teised selle keele võimalused, näiteks massiivid ja liidestamisel kasutatavad *PHP* laiendused nagu *SOAP* ja *HTTP*.

Antud töös kirjeldatud probleemidest vääriks edasi uurimist jõudluseprobleemid *Raintree*-spetsiifilises *PHP*-keele laienduses. Teiseks, järgides antud töö lahenduses välja pakutud *PHP* parendamise plaani, saaks koostada konkreetse disainidokumendi, mille abil saaks *PHP*-d laiendada *RSL*-i eeskujul domeenispetsiifiliseks keeleks.

Review and Enhancement of PHP and Raintree Scripting Language

Bachelor Thesis

Kalmer Kurg

Summary

The main goal of the thesis was to find a solution for enhancing one or another scripting language that is used by *Raintree Systems, Inc.* Language enhancement was elicited by the fact that the current languages are not used purposefully, although *PHP* was put to use because of its object-oriented programming features in order to create easily extendable business application modules, but *Raintree* scripting language is used instead. The reason of this phenomenon was in the way how these two languages were implemented – compared to *PHP*, *Raintree* scripting language in its domain specificness is expressing more accurately the essence of *Raintree* application. Since the solution to the problem was incorporating object-orientation and domain specificness, then the easiest way to achieve it would be extending the *PHP* with domain-specific commands. In that way we would acquire other benefits that *PHP* has to offer like arrays and *PHP*-specific *HTTP* and *SOAP* extensions for implementing data interfaces between *Raintree* and other information systems.

For achieving this thesis's objective, the domain-specific language essence and characteristics were described in the first chapter. In the second chapter the architecture of *Raintree* application and *Raintree* scripting language was observed, also the language elements and syntax. In the third chapter the focus was on describing the integration between *Raintree* application and *PHP*. Also the *Raintree Core Library* framework's main classes and the benefits of using the framework were described. In the fourth chapter the *Raintree* scripting language and *PHP* was compared in the aspect of syntax, domain specificness, extensibility and language elements. Fifth, the last chapter connected the previous chapters by describing the advantages and disadvantages, when comparing *PHP* and *Raintree* scripting language, then the usage of these languages in *Raintree Systems, Inc.* and came into conclusion that the advantages of the both languages can be incorporated by extending the *PHP* in a way that it would be domain-specific as *Raintree* scripting language.

Kasutatud kirjandus

[1] D. Spinellis. *Notable design patterns for domain specific languages*. Journal of Systems and Software, Volume 56, Issue 1:91–99, 2001.

[2] M. Fowler. *Domain Specific Languages*. Addison-Wesley Professional, 2010

[3] M. Abel, Raamistik *PHP* programmeerimiskeele integreerimiseks ärilahendusena väljaspool veebi. Tartu, 2008. (Käsikiri)

[4] *Pascal ISO 7185:1990*

http://www.moorecad.com/standardpascal/iso7185.pdf (12.04.12)

[5] *QBASIC Keyword Reference – Alphabetical*
http://qb64.net/wiki/index.php?title=Keyword_Reference_-_Alphabetical (12.04.12)

[6] S. Tint. *Integrating PHP with Delphi in order to create a highly customizable application.* Tartu, 2008. (Käsikiri)

[7] *Raintree PHP Core Library*

http://files.rtedev.com/corelib/ (29.04.12)

[8] *History Of PHP*

http://ee.php.net/manual/en/history.php.php (29.04.12)

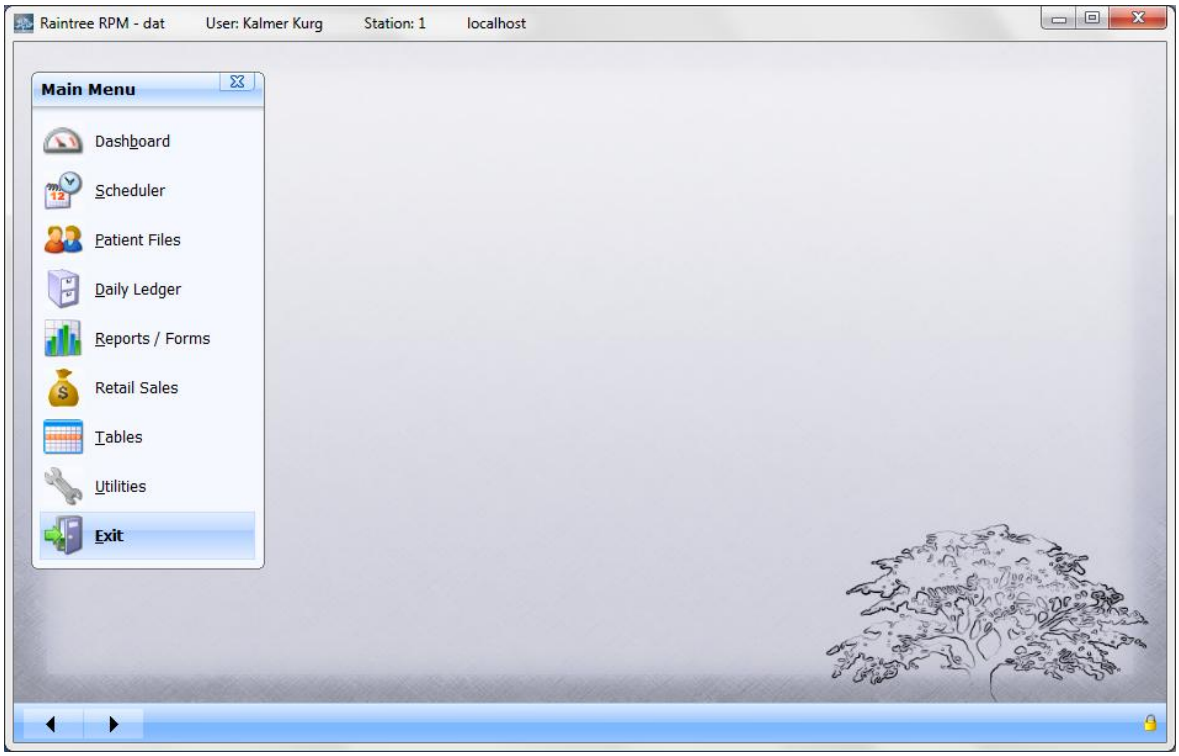
[9] M. Mernik, J. Heering, A. M. Sloane. *When and how to develop domain-specific languages*. Journal ACM Computing Surveys, Volume 37, Issue 4:317-344, 2005.

[10] *PHP: Language Reference - Manual*

http://www.php.net/manual/en/langref.php (30.04.12)

Lisad

Lisa 1. *Raintree* põhirakendus avanemisel peamenüüga



Lisa 2. Lihtne e-posti saatmise rakendus realiseerituna *RSL*-is

Rakenduse vaade (RTM):

Vastav RSL-i skript:

```
rem ----------------------------------------------------------------------------------------
rem Simple email sending program
rem Author: Kalmer Kurg, May 2012

rem Position to user-specific location
usefile "EMAIL"
index lanid

rem Main flow of the program
repeat
  set %finished = ""

  rem Show screen
  input "EMAIL"

  rem Handle user interaction
  if refresh = "Y" then
  begin
    if fldname = "BTNSEND" then
    begin
      set %ok = "Y"
      call ValidateFromAndTo(%ok)
      if %ok = "Y" then call ValidateSubjectAndMessage(%ok)
      if %ok = "Y" then call CustomSendEmail(%finished)
      if %finished = "Y" then
      begin
        message "Notice" "Email has been sent."
      end
      elsebegin
        if %finished = "N" then message "Error" "Email has been not sent." lasterrortext
      end
    end
  end
  elsebegin
    set %finished = "Y"
  end

until %finished = "Y"

end.


rem Check mandatory From and To Fields
localfunc ValidateFromAndTo(var %ok)
  if (trim(&email:fldFrom) = "") then
  begin
    message "Error" "From field cannot be empty."
    focus "FLDFROM"
    set %ok = "N"
  end
  elsebegin
    if (trim(&email:fldTo) = "") then
    begin
      message "Error" "To field cannot be empty."
      focus "FLDTO"
      set %ok = "N"
    end
  end
endfunc


rem Check optional Subject and Message fields
localfunc ValidateSubjectAndMessage(var %ok)
  if (trim(&email:fldSubject) = "") then
  begin
    set %msg = "Subject field"
    set %focus = "FLDSUBJECT"
  end
  elsebegin
    if (trim(&email:fldMessage:content) = "") then
    begin
      if %msg = "" then
```

```
      begin
       set %msg = "Message field"
       set %focus = "FLDMESSAGE"
      end
      elsebegin
        set %msg := "Subject and message fields"
      end
    end
  end

  if %msg <> "" then
  begin
    switch (doMessage("Warning", %msg . " are empty. Proceed?", "No|Yes"))
      case (0, 1)
        rem Escaped, "No"
        set %ok = "N"
        focus %focus
      case (2)
        rem "Yes", do nothing
    endswitch
  end
endfunc


rem Send email based on data entered on screen
localfunc CustomSendEmail(var %result)
  sendemailt &email:fldTo &email:fldFrom &email:fldSubject &email:fldMessage:content
             &email:fldAttachment
  if lasterror = 0 then set %result = "Y"
  else set %result = "N"
endfunc
rem ----------------------------------------------------------------------------------------
```

Lisa 3. Lihtne e-posti saatmise rakendus realiseerituna *PHP*-s

```
// Simple email sending program
// Author: Kalmer Kurg, May 2012

// EMAIL Module class
class Email_Module extends Module {

  public function init() { }

  // Function for displaying EMAIL screen
  public function run() {
    $scr = $this->getScreen('EMAIL');
    $scr->show(false);
  }
}


// EMAIL Screen class
class Scr_EMAIL extends Screen {

  // Handle Send button refresh
  function refresh_btnSend() {
    if ($this->ValidateFields()) {
      $result = $this->data->CustomSendEmail();

      if ($result === true) {
        Message::show('Email has been sent.');
        return self::EC_SAVE;
      }
      else {
        Message::show('Email has not been sent. \n' . $result, Message::WARNING);
      }
    }
  }


  // Validate email screen fields, returns true
  // if not passed validation, false otherwise
  function ValidateFields() {
```

```
    $focus = $this->data->ValidateFromAndTo();
    if (!$focus) {
      $focus = $this->data->ValidateSubjectAndMessage();
    }

    // Set focus to empty field
    if ($focus) $this->setFocus($focus);
    else return true;

    return false;
  }
}


// EMAIL Record class

class Rec_EMAIL extends Record {

  // Validate From and To fields
  function ValidateFromAndTo() {
    if (trim($this->getField('fldFrom')) == '') {
      Message::show('From field cannot be empty.', Message::WARNING);
      return 'fldfrom';
    }
    else {
      if (trim($this->getField('fldFrom')) == '') {
        Message::show('To field cannot be empty.', Message::WARNING);
        return 'fldto';
      }
    }

    return '';
  }


  // Validate Subject and Message Fields
  function ValidateSubjectAndMessage() {
    if (trim($this->getField('fldSubject')) == '') {
      $msg = 'Subject field';
      $focus = 'fldSubject';
    }
    else {
      if (trim($this->getField('fldMessage')) == '') {
        if ($msg == '') {
          $msg = 'Message field';
          $focus = 'fldMessage';
        }
        else {
          $msg = 'Subject and message fields';
        }
      }
    }

    if ($msg <> '') {
      $result = Message::showPrompt($msg . ' are empty. Proceed?', array('No','Yes'),
                                    Message::WARNING);

      if ($result) return $focus;
    }

    return '';
  }


  // Send email based on data entered on screen
  function CustomSendEmail() {
    return Common::sendEmail(
               $this->getField('fldTo'), $this->getField('fldSubject'),
               $this->getField('fldMessage'), $this->getField('fldAttachment'),
               $this->getField('fldFrom')
               );
  }
}
```