Tallinna Ülikool
Matemaatika – loodusteaduskond
Informaatika osakond

Margo Poolak

Veebi raamsüsteemid: ülesehitus ja prototüübi loomine

Bakalaurusetöö

Juhendaja: Jaagup Kippar

Autor: ..... „....”.....
Juhendaja: ..... „....”.....
Osakonna juhataja: ..... „....”.....

Tallinn 2007

Sisukord

1. Sissejuhatus

1.1. Ülevaade

Vajaduste ja nõudmiste suurenemisega kaasneb süsteemide aina keerulisemaks muutumine ja nende lahendamiseks on vaja arendada välja paremaid ja mugavamaid lahendusviise, mis muudaks töö lihtsamaks nii arendajatele ja kasutajatele võimalikult mugavaks. Üks peamisi vahendeid veebi rakenduste arendamisel on veebi raamsüsteemid (ingl. k. *web frameworks*), mis praegusel hetkel elab läbi väga intensiivset arengut ja muudatusi. Praegusel hetkel võib internetist leida väga laias ulatuses veebi raamsüsteeme igas populaarsemas programmeerimise keeles. Nende vahel valida on üsna keeruline ja aeganõudev töö.

Iga edasijõudnud arendaja on kokku puutunud ühe või teise veebi raamsüsteemiga, et oma arendustöid muuta kiiremaks ja lihtsamaks. Tihti kasutatakse neid raamsüsteeme seetõttu, et ei ole aega uurida alternatiive, teha selgeks kuidas kasutada või välja arendada endale sobilikum raamsüsteem. On hulk inimesi kes on leitud raamsüsteemiga peaaegu „rahul" või „harjunud".

Kindlasti on tekkinud raskusi ja probleeme mida vastav raamsüsteem ei suuda täita või eeldab millegi juurde kirjutamist või halvemal juhul terve raamsüsteemi välja vahetamist.

Ja kui vastavaid raamsüsteemi kitsendusi ei ole, siis võib olla probleemiks aeganõudev rakenduse loomine mis ühel hetkel viib mõtted alternatiividele.

Sellisel hetkel tuleb otsustada, et kas võtta kasutusele järgmine raamsüsteem või luua ise täiesti nullist vastav raamsüsteem. Esimesel puhul on oht, et kui piisavalt ei uurita antud raamsüsteemi siis võib ühel hetkel uuesti sarnastesse probleemidesse langeda. Teisel puhul on ohuks aeg ja teadmised.

Enda raamsüsteemi loomine on aeganõudev töö ning vajab laialdasemaid teadmisi.
Et aga hinnata, mis on hea raamsüsteem ja mis halb, peab omama ülevaadet, mis on üldse raamsüsteem ja millest ta koosneb. Mis on raamsüsteemile esitatavad nõuded ja vajadused mida arendajad ootavad? Kuidas kasutatakse ning kuidas toimib? Mis teenuseid ja tehnoloogiaid peaks ta toetama? Mis probleeme peaks lahendama? Kuidas luua jätkusuutliku veebi raamsüsteemi?

Nendel ja teistele küsimustele katsun oma töös vastused anda, eesmärgiga muuta rakenduste loomine ja kasutamine läbipaistvamaks, mugavamaks ja laiendustele avatumaks.

1.2. Aktuaalsus ja tähtsus

Hetkel kui arendati välja esimene Javascript tuginev raamsüsteem Prototype[1] ning võeti kasutusele Ajax[2] tehnoloogia , võib pidada veebi raamsüsteemide uuesti sünniks, sest see andis võimaluse arendada veebe tunduvalt interaktiivsemaks ja kasutajasõbralikemaks.

Siiamaani oli veebi rakenduste loomine orienteeritud peamiselt staatilistele lehekülgedele, mitte aga dünaamiliselt muutuvatele sisudele.

HTTP ja HTML on lehele orienteeritud tehnoloogiad. Eelnevalt pakkusid selliseid dünaamilise veebi arenduses võimalusi ColdFusion, ASP ja JSP jt. taolised keeled ja nendel tuginevad raamsüsteemid.

Seega on praegu tehnoloogiliselt aeg küps arendamaks välja raamistiku, mis vastaks just nendele ootustele ja vajadustele mida tarvis hea ja kasutajasõbraliku veebi rakenduste loomisel ja loomiseks, mis ei tegeleks üleliigsete „featuuride" arendamisega, vaid tegeleks ideoloogiliselt hea rakenduse ja võimaluste pakkumisega.

Praegune veebi raamsüsteemide vohamine näitab selgelt, et puudub see „õige" lahendus probleemile (ingl. k. *„The current proliferation of web frameworks is a great indication that there's no "right" solution to the problem")* [8].
Oma töös katsun leida vajadused, mis võiks iseloomustada seda „õiget" raamsüsteemi.

Lisaks sellele, et on olemas lihtsad raamsüsteemid ja ülesehitused puudutan arhitektuuri mustreid ning kasutamist veebi raamsüsteemides.

[1] Prototype on arendatud juba 2005 aastast ja aktiivsemalt võeti kasutusele 2006 aastal

[2] Ajax tehnoloogia jõudis massidesse 2006 aastal peale Google Suggest tutvustust

2. Eesmärk

Eesmärk on saada ülevaade veebi raamsüsteemidest kuidas nad toimivad või kuidas üles ehitatud. Mis teenuseid peaksid veebi raamsüsteemid sisaldama ja kuidas need toimivad.

Üritan oma töös ennetada probleeme, mis võivad tabada veebi rakenduse loomisel ning leida neile lahendus.

Luua eesti keelne materjal ja ülevaade probleemidest, mis seonduvad antud teemaga.

2.1. Tulemus

Antud töö üleüldiseks tulemuseks on:

- Saada ülevaade antud teemast ja luua süstematiseeritud materjal
- Kirjutada lahti antud teemaga seotud mõisted ja terminid
- Luua ülevaade praegustest veebi raamsüsteemidest
- Luua prototüüp veebi raamsüsteem
- Anda teema ideesid teistele asjast huvitatutele

2.2. Meetodid

Peamiseks materjalide ja teemade kohta info otsimiseks kasutasin internetist leitavaid materjale ja nendes olevaid viiteid. Kuna eesti keelset materjale antud vallas peaaegu puudus või leidus kaudselt, siis lisaks uurisin ja kogusid teemaga seotud teadustöid seminari,- bakalauruse- ja ka magistritöödest.

Kogusin materjale ja viiteid kõikidest enamlevinud ja populaarsematest veebi raamsüsteemidest. Lisaks uurisin kasutatavaid arhitektuuri ülesehituse mustreid ja tehnoloogia mida kasutatakse.

Peamiselt lähtun oma töö sisu ülesehitamiseks järgnevas raamatus ja artiklis:

- Schlossnagle, G. (2004). *Advanced PHP Programming: A practical guide to developing large-scale Web sites and applications with PHP5.*
- Pollak D. (7. dets. 2006). *Why the world needs another web framework.* [5. aprill 2007], http://blog.lostlake.org/index.php?/archives/25-Why-the-world-needs-another-web-framework.html

Raamsüsteemide analüüsimisel kasutan lähenemiseks liivakasti (ingl. k. *sandbox*) loomist, kus sees üritan valmis teha näidisrakenduse, et näha kuidas erinevad raamsüsteemid toimivad ja mis on nende eripärad. Lisaks katsun uurida, mis tehnoloogiad antud raamsüsteemid kasutavad.

Raamsüsteemide analüüsimisel üritan leida põhjendatud failisüsteemi ülesehituse struktuuri, mis oleks hõlbus kasutada erinevates tingimustes.

„On fakt, et kui toimub vohamine mingis „asjas", siis tähendab ükski neist ei ole adresseeritud nii, et see täidaks vajadusi ja on aeg vaadata mis need vajadused on ja neid adresseerida. See oli juba 1990 keskel kui tollel ajal oli umbes 30 erinevate 32-bitilist operatsioonisüsteemi töölaua jaoks. Täna on meil aga Windows, Mac OS X ja Linux. Sellel ajal oli ka vohamine. Vähesed jäid alles." [8].

On aeg vaadata, mis need veebi raamsüsteemi vajadused on ja need täita.

Kuna vajadusi võib olla sama palju kui inimesi, siis katsun leida mõne tähtsama enda ja teiste jaoks. Antud vajaduste põhjalikumaks uurimiseks võib luua oma uurimustöö. David Pollak kirjutas oma artiklis „hea" veebi raamsüsteemi kriteeriumitest „Veebi raamsüsteemi manifest" (ingl. k. „*Web Framework Manifest*") [9] kust ta läheneb raamsüsteemile kui „featuuride" rohkele rakendusele. Antud töös lähenen asjadel hoopis arhitektuurilisest vaatevinklist. Üritan leida õiget veebi raamsüsteemi failistruktuuri ja rakenduse kasutusviisi.

2.3. Ülesehitus

Bakalaurusetöö koosneb sissejuhatavast osast, eesmärgi kirjeldamisest, ülevaatlik osa, ülevaade raamsüsteemidest, prototüüp rakenduse loomine ja kokkuvõttev osa. Lisaks inglise keelne resümee ja kasutatud kirjanduse loetelu ning tutvumiseks lisa materjal, et näha erinevate rakenduste failisüsteemi ülesehitust ja struktuuri.

3. Ülevaade teemaga seonduvast olemasolevast teabest

Järgnevas peatükis kirjeldan lahti teemaga seotud terminid ja mõisted nagu raamsüsteem, rakendus, rakenduse programmeerimise liides, Ajax jne., mis annavad parema ülevaate ja arusaamise teemast.

3.1. Raamsüsteem (Framework)

Raamsüsteem (ingl. k. *framework*) on organiseeritud ja struktureeritud abivahend, mille abil organiseeritakse ja arendatakse programmi arendus projekte.

Eesmärgiks on lihtsustada ja aidata arendustööd. Tegemist on n.ö. toetusüksusega, mis lahendab mingeid kindlaid korduvaid probleeme. Seob omavahel erinevaid objekte ja komponente, et luua keerulisemaid lahendusi. Sunnib arendajal looma koodi nii, et see oleks järjekindel, sisaldaks vähem vigu ja oleks paindlikum. Peab andma igaühel võimaluse testida ja vigu otsida koodist mida ta pole ise kirjutanud [1].

Raamsüsteem sisaldab alamprogramme[1] klassiteeke (ingl. k. *class library*), mingit teist kirjasüsteemi (ingl. k. *script*) või teisi programme, et aidata arendada ja ühendada omavahel erinevaid programmi osasid üheks tervikuks. Raamsüsteem võib koosneda ühest või mitmest rakendusest (ingl. k. *application*).

Raamistik on kogum klassidest või hoidlatest mida kasutatakse, et võtta kasutusele rakenduse standardseid struktuure mingis kindlas operatsioonisüsteemis.

Klassikalisest rakendusraamistikust rääkides mõistame me Objekt- Orienteeritud (OO) rakendusi. Antud töös vaatleme peamiselt OO raamistike.
Peamised eesmärgid mida OO rakendus raamistik annab arendajale on:

- **Moduleeritus** (ingl. k. *Modularity*) - raamsüsteemid parandavad programmi kvaliteeti kapseldades enda sisse keerulisi rakenduse osasid peites selle n.ö. stabiilse liidese taha. Selline lähenemine peaks vähendama vajadust aru saada raamsüsteemi enda ülesehituse iseärasusi ja arenduse vajadusi.
- **Korduvkasutatav** (ingl. k. *Reusability*) **–** annab võimaluse luua objekte ja komponente, et edaspidi oleks võimalik luua edasiarendusi ja keerulisemaid rakendusi vähesema vaevaga.
- **Laiendatavus** (ingl. k. *Extensibility*) – laiendatavus peab toimima lihtsalt läbi sidumise meetodite, mis säilitaks üleüldise rakenduse stabiilse toimimise.

[1] Programmeerimises nimetatakse valmiskompileeritud alamprogrammi nimega **teek**

- **Ümberpööratud kontroll mehhanism** (ingl. k. *Inversion of Control*) – taoline arhitektuur annab rakendusele võimaluse käia läbi sammud, et haldaja saaks kohandada sündmusi objektidele, mis on raamsüsteemi poolt välja kutsutud. Kui sündmus toimub, siis raamsüsteem reageerib sellele käivitades seotud meetodi eelnevalt registreeritud objektis, mis siis täidab vajaliku protseduuri. Seega annab see raamsüsteemi võimaluse otsustada mida mingi tegevuse korral tuleb teha (nt. veateade kasutajaliidesest või akna sulgemise sündmus sulgeb ka teatud andmebaasi ühendused jne.).

Raamsüsteemi ülesehitus toetab arendatava süsteemi korduvaid eripärasid.

Nt. Veebi lehe arendus puhul luuakse veebi raamsüsteem, mis lihtsustab HTML koodi kirjutamist ja lehekülgede loomist koos abivahenditega hallata nt. Javascripti jne.

Raamsüsteeme saab jaotada nende suuruse ja kasutusotstarbe järgi järgmiselt [3]:

- **Süsteemi infrastruktuuri raamistikud (**ingl. k. *System infrastructure frameworks***) -** Need raamistikud hõlbustavad teisaldatavate ja tõhusate süsteemide infrastruktuuri arendust nagu operatsioonisüsteemid ja kommunikatsiooni raamistikud.

- **Kesktaseme integratsiooni raamistikud (**ingl. k. *Middleware integration frameworks*) **-**
  Selliseid raamistike kasutatakse peamiselt, et integreerida jaotatud rakendusi ja komponente. Üks enim arenev raamistike jaotus.

- **Ettevõtte rakendus raamistikud (**ingl. k. *Enterprise application frameworks*) **-** Taolised raamistikud on mõeldud tervele rakenduste domeenile (nt. telekommunikatsioon, tootmine, finants arendus, teenindus jne.) ja on sellisel juhul firma äritegevuse nurgakiviks.

Oma töös tuleb juttu peamiselt süsteemi infrastruktuuri ja kesktaseme integratsiooni raamistikest.

3.2. Rakendus (Application)

Rakendus on reaalselt toimiv programmi osa või liides. Ta on otseselt seotud lõpp-kasutajaga ning täidab tema poolt ette antud ülesandeid. Et kasutajal oleks võimalik lihtsamini suhelda taoliste rakendusega, siis lisandub neile ka kasutajaliides. Rakendus programmi kutsutakse ka lõppkasutaja programmiks (ingl. k. *end-user programm*).

Rakendus võib olla seotud ka mõne teise programmiga mille korral täidab samu funktsionaalsusi nagu tavakasutaja puhul ainult kasutajaliides puudub või on väga minimaalne.

Rakenduste vaheliseks suhtlemiseks on väga mitmeid erinevaid meetodeid millest me räägime ka alljärgnevas töös. Alustades sellest kuidas rakendused suhtlevad omavahel lokaalses masinas ja kuidas väliste rakendustega üle interneti. Ei saa ka mainimata jätta, et nende vahelises suhtluses tuleb arvestada andmete tüübiga ja mis mahus ning mis keskkonnas nad seda teevad.

Kindlasti tuleks arvestada, et enamike rakendusi kasutab operatsioonisüsteemi ja tema teenuseid ning teisi tugirakendusi.

Näiteks: joonistamine, teksti töötlemine, andmemudelid, veebileht jne.

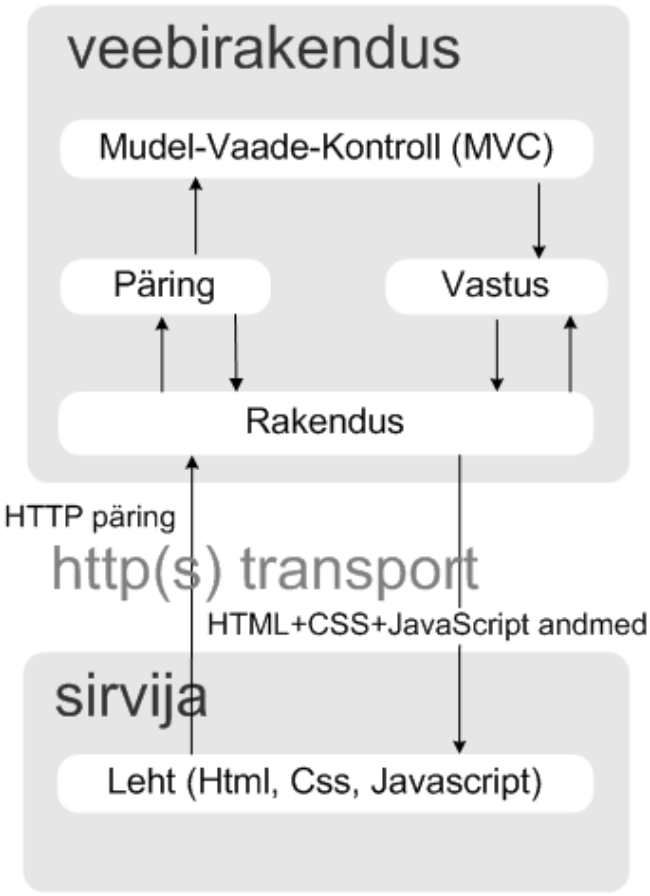
3.3. Veebi rakenduse raamsüsteem (Web application framework)

Selle all mõistetakse programmi tööriistu ja koodi kogusid, mille eesmärgiks on toetada dünaamilise veebi ja rakenduste arendamist.

Antud juhul peaks antud rakendus lihtsustama ja kiirendama arendustööd. Selle tarvis luuakse eraldi klasside kogumeid, mis täidavad veebiga seotud iseloomulike ülesandeid.

Näiteks: templiit, andmebaasi ühendus, sessioon, identifitseerimine, stiili leht, ajax jne.

Peamine veebi raamsüsteemi ülesehitus on lihtne rakendus (ingl. k. *Application*) klass, mis juhib kogu protseduuri. Järgnevalt püüab päringu (ingl. k. *Request*) klass kinni kasutaja sirvijast saadetud päringu soovi ehk siis lehekülje aadressi. Edasi pöördutakse serveris asuva templiit faili poole, mida enamasti haldab MVC programmeerimise muster ja mida kuvada läbi vastus (ingl. k. *Response*) klassi sirvijasse tagasi (vt. Joonis 1).



Joonis 1. Standardne veebi raamsüsteemi rakenduse ülesehitus

3.4. Rakenduse programmeerimise liides (Application Program Interface)

Aplikatsiooni programmeerimise kasutajaliides (ingl. k. *Application Programming Interface*, edaspidi lühendina API) on abivahend programmeerijale, et kirjutada rakendust.

Peamine API mõiste kasutus on koodi klasside ja liideste kasutamise moodus, et luua rakendust mingil kindlal viisil.

Enamik suuremaid ja kommertskasutuses olevaid rakendusi sisaldavad endas API-t. Üks osa API funktsionaalsust võib olla Application-Level Iteraction, mille kaudu saavad teised rakendused kasutada ligipääsu vajalikele abivahenditele.

API võib olla ka osa programmi arendamise komplektist (ingl. k. *Software Development Kit*, lühend SDK)

3.5. Graafiline kasutajaliides (Graphical User Interface)

Selle all mõistetakse graafilist liidest mille kaudu kasutaja saab suhelda arvutiga. Enamasti on graafiline liides väga palju seotud operatsioonisüsteemiga endaga aga kui tegemist on veebiga, siis graafiline kasutajaliides (ingl. k. *Graphical User Interface*, edaspidi lühendina

GUI), kui selline on praktiliselt täielikult eraldatud OS'ist. Veebis saab graafiline liides endale täiesti uued mõõtmed ja võimalused.

GUI elementideks loetakse: aknad, avatavad menüüd, nupud, ikooni pildid, visardid, hiire kursor jpt. taolisi asju. Aina suureneva multimeedia toega kuulub paljude GUI rakenduste juurde: muusika, hääled, liikuvad videod ja virtuaalreaalsed kasutajaliidesed

Hetkel kus kõik suuremad operatsioonisüsteemid on läbimas suuri muudatusi oma kasutajaliideste arendamisel on veeb juba aegade algusest saadik läbinud kiireid ja mitmekülgseid arenguid. Alustades sellest, et tegemist oli puhtalt teksti kuvamisega ja piltidega on need nüüd praktiliselt saanud iseseisva väljanägemise ja võimalused.

Veebi rakenduste arendamise lihtsustamiseks on tihti välja arendatud viise kuidas väljundit standardiseerida ja sellest tulenevalt on vägagi palju rakenduse osasid seotud ka graafilise liidesega ja kuvamisega.

Ise arvan, et veebi rakenduste arendamisel on rohkem võimalusi, kui tava operatsioonisüsteemi puhul. Ei ole kaugel ka aeg kus on võimalik luua 3-dimensionaalseid veebilehti, sest juba praegu on olemas rakendus, mis kasutavad 3D mootorit. Seda küll veel .Net 3.0 või Java platvormidel. Kindlasti jõuab taoline tehnoloogia avatud rakendustesse ja üldkasutatavate veebide osadeks.

Näide: Photosynth on küll Microsofti Live Labs poolt välja arendatud tehnoloogia aga see annab ettekujutuse, mis moodi on võimalik luua kolme dimensioonilist fotoalbumit.

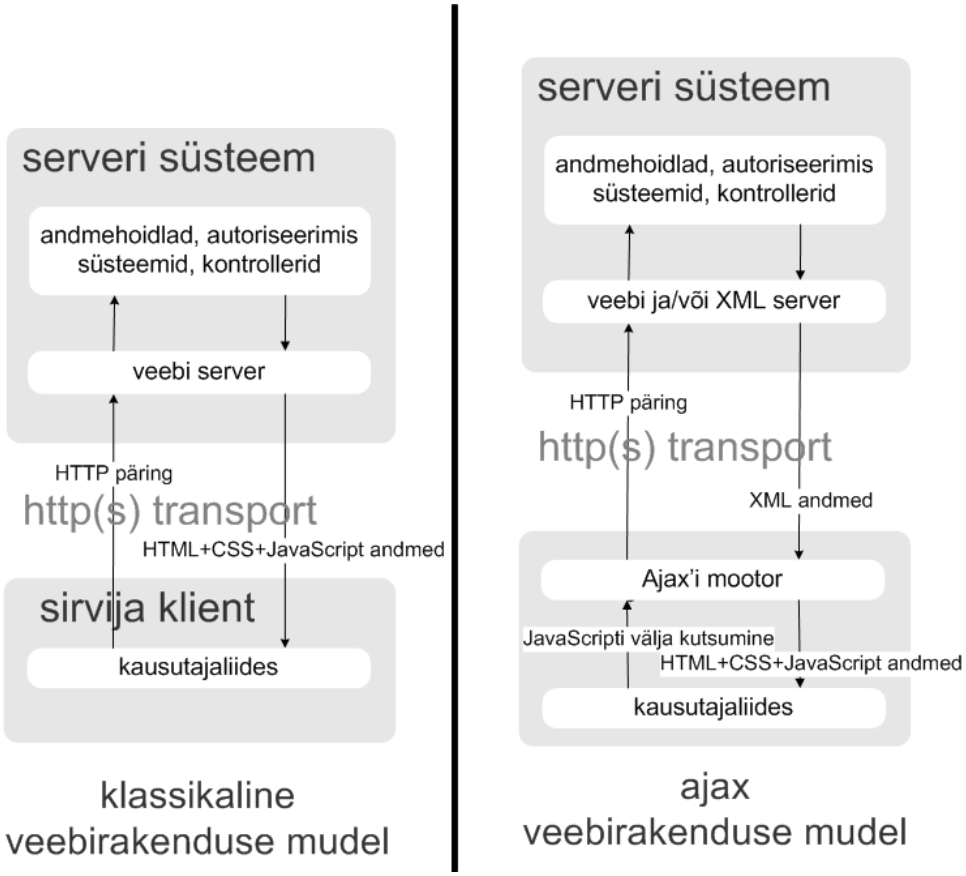
(vt. http://labs.live.com/photosynth/)

Tehnoloogia poolest on näiteks Ajax kaasa toonud hoopis uue taseme veebi rakenduste loomisel. Sest kuni selle ajani oli graafiliste liideste loomiseks vaja üsna palju ressursse, nii interneti ühendus kui ka kasutaja arvuti ei olnud võimelised normaalse ajaga laadima alla suuremas mahus graafikat ja andmetehulka.

3.6. Ajax (Asynchronous JavaScript and XML)

Tegemist on veebi arenduses väga laialt levib tehnika arendamaks interaktiivsemaid veebilehekülgesid. Eesmärgiks on anda kasutajaliidesel võimalus suhelda serveriga nii, et see ei takistaks kasutaja tööd, vaid töötaks taustal.

Nimelt toimuvad päringud ilma, et kasutaja peaks laadima kogu lehte uuesti kõigi päiste ja jalustega. Ajaxit kasutades saab edastada serveri ja kliendi vahel ainult neid andmeid mida tal reaalselt vaja on. Suhtlus toimub Javascripti kaudu, mis kasutab XmlHttpRequest kanalit (vt. Joonis 2). Praeguseks hektkeks toetavad XmlHttpRequest'i enamik uuemaid sirvijad (ingl. k. *browser*).



Joonis 2. Traditsiooniline veebi rakenduse mudel (vasak) ja Ajax rakenduse mudel (parem) [18]

XmlHttpRequest kasutati esimesena 2000 Internet Explorer 5 peal. Laiema populaarsuse saavutas see tehnoloogia 2005 aastal, kui Google kasutas seda oma Google Suggest sisestus lahtris (ingl. k. *input box*).

Antud rakenduse toimimiseks on vaja kliendi liideses asuvat Ajax mootorit milleks võib olla Prototype (vt. teema „Prototype") ja serveri poolel piisab sellest kui on lihtsalt üks lehekülg mille poole pöördutakse.'

3.7. Käsurea kasutajaliides (Command Line Interface)

Tegemist on käsurea liidesega e. konsooliga (ingl. k. Command Line Interface, edaspidi lühend CLI), mille kaudu on võimalik kasutajal sisestada käske mingite tegevuste sooritamiseks. Selliseid käsuridasid võib leida erinevates operatsioonisüsteemides. Põhiliselt kasutame me neid mingite lisaparameetrite edastamiseks mingi konkreetse programmi käivitamisel või kasutamisel.

Nüüdseks on palju veebirakendus raamsüsteemid võtnud kasutusele käsurea kasutamise. Seda kasutatakse erinevate korduvate tegevuste toetamiseks nagu uue projekti loomine, andmebaasi struktuurid, testimise keskkonda jne.

Plussiks võib lugeda siin seda, et tõesti kui sul on vaja luua kiiresti projekt koos kõigi vajalike failidega, siis piisab ühest käsureast. Või kui sul on vaja testida terveid katalooge, siis taoline käsurea pealt on võimalik saada kiire tulemuse.

Miinuseks loen seda kui taoline lähenemine ainuvõimalik ehk siis ilma selleta ei ole võimalik luua rakendust või seda hallata.

3.8. URL kaardistus (URL mapping)

URL (ingl. k. *Unified Resource Locator*) kaardistuse (ingl. k. mapping) all mõistetakse aadressi rea n.ö. kaardistamist. Eesmärgiks on muuta aadressi rea väljanägemist vastavalt vajadusele. Esialgne URL ei kao kuhugi, vaid see asendatakse teise kujuga.

Üks lahendus on kasutada Apache veebiserveri mod_rewrite moodulit. Salvestades root kataloogi „.htaccess” faili, kuhu sisse kirjutada all järgnev kood:

```
# Rewrite rules
<IfModule mod_rewrite.c>
  RewriteEngine on

  # Rewrite URLs of the form 'index.php?q=x':

  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule ^(.*)$ index.php?q=/$1 [L,QSA]

</IfModule>
```

Antud kood kirjutab tava aadressi ümber nii, et kaotab ära „?” , „&” ja „=” märgid ning asendab need „/” kaldkriipsuga.

Muudetud URL saadetakse ta üle kui „q" parameeter, mis tuleb rakenduse poolt kinni püüda.

- Esialgne URL:

```
http://localhost/?id=1
```

- Muudetud URL:

```
http://localhost/?q=/id/1
```

3.9. Puhver (Cache)

Oma töös ma ei peatu pikalt vahemälule või salvestusele, sest antud juhul on see seotud rohkem platvormide endaga, kui loodavate rakenduste enda probleem. Kui võtta arvesse, et ei ole võimalik platvormi muuta kiiremaks läbi selle, et ehitada kuhugi uus kiht, mis tegelikult teisendab programmi tagasi algkeeleks mida kasutatakse nt. Smarty puhul. Aga selle ära jätmisel oleme võitnud rakenduse ressursside ja kiiruse poolest. Lisaks teeb taoline salvestus meetod süsteemi keerulisemaks ja aeglasemaks, sest raisatakse lisaressursse antud vahemälu haldamiseks ja salvestamiseks. Seega pigem propageeriksin kasutama algset keelt nii nagu ta on.

Väga raske on saada taoliste kõrgtaseme keelte nagu PHP, Python, Ruby, Java jne. puhul haldusesse mälu ja selle kasutust, sest enamasti on see automatiseeritud.

Kui võtta arvesse, et enamik veebi rakendusi on dünaamilised ja antud veebide n.ö. sisud ja andmed võivad muutuda väga tihti, siis oleks väga keeruline taolisi süsteeme salvestada, kui juba järgmisel hetkel tuleb võtta kasutusele uued andmed ja neid töödelda.

Seega tasub luua süsteeme nii, et dünaamika oleks sisse ehitatud nagu algselt oli mõeldud dünaamilisel kujul ehk siis massiivide töötlemine, Javascript või Ajax kasutamine, jne.

Hulga ressursse kasutab liigne loogika mida saaks optimiseerida ja vale koodi kasutus. Väga suurt rolli määrab andmebaaside kasutamine. Näiteks Mssql andmebaasi ühendus tegemine võtab aega vähemalt 1 sekund, kui mitte rohkem, rakenduse tööst. Lisaks sellele tuleb arvestada andmebaasi päringutele mida need tagastavad jne.

Ainuke asi, mis annaks mingi kiiruse juurde on andmebaaside käivitajad (ingl. k. *triggerid*), mis annaksid teada muudatustest andmebaasi sees ning alles seejärel toimuks andmebaasidest andmete pärimine. Sellest kirjutamine aga on juba teine teema.

4. Välised teenused (Remote Services)

Iga rakendus ei piirdu ainult sisemistele andmetele ja nende töötlemisele. Globaliseeruvas interneti maailmas hakkab aina enam levima jaotatud ressursside kasutamine üle interneti. Selle jaoks on välja töötatud terve rida tehnoloogiaid millest ma allpool lähemalt räägin. Peamiselt on tegemist suhtlusprotokollidega, sest HTML ise ei täida struktureeritud andmete edastamiseks ja vastu võtmiseks vajaminevaid nõudeid, vaid selle jaoks on XML või lihtsustatud kujul olev JSON andmestruktuurid.

4.1. XML-RPC (Extensible Markup Language-Remote Procedure Call)

XML-RPC on kaugprotseduurkutsung (ingl. k. *Remote Procedure Call*, edaspidi RPC), mis toimib üle interneti. See on protokoll, mis võimaldab kasutaja programmil täita teises n.ö. serverarvutis olevaid programme.

XML-RPC teade antakse edasi kui HTTP-POST päring. Päringu sisu on XML formaadis. Sisaldab suhteliselt vähe andmetüüpe, mis tegelikult on teiste analoogsete teenuste eeliseks. Töötab erinevatel operatsioonisüsteemidel nagu Unix, Windows ja MacOS.

Toimimiseks on vaja klienti (ingl. k. *client*) ja serverit (ingl. k. server) esimene neist teeb päringu teises arvutis olevasse server programmi, mis töötleb andmed ümber ning tagastab need.

Päringu näide kus päritakse teisest serverist kasutajanime id järgi:

XML-RPC päring (koos HTTP päisega):

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: localhost
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>User.getUserNameById</methodName>
  <params>
    <param>
        <value><i4>10</i4></value>
      </param>
  </params>
</methodCall>
```

XML-RPC vastus (koos HTTP päisega):

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: localhost WINNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
        <value><string>John</string></value>
    </param>
  </params>
</methodResponse>
```

4.2. ORB (Object Request Broker)

ORB on kesktaseme tehnoloogia, mis manageerib kommunikatsiooni ja andmevahetus objektide vahel.

See pakub mehhanisme, et nähtamatult kommunikeerudes kliendi päringuga, et saada kätte ja võtta kasutusele objekt. ORB võimaldab hajus-programmeerimisel eraldada kliendi ja meetodi kutsumise detaile. Kliendi päring paistab sellisel juhul nagu lokaalne protseduur.

Raamistikud, mis hõlbustavad kommunikatsiooni lokaalsete ja väliste objektide vahel. ORB raamistikud on CORBA, DCOM, Java RMI [21].

4.2.1. CORBA (Common Object Request Broker Architecture)

CORBA on standardse arhitektuuri spetsifikatsioon *objekti päringu vahendaja* (ORB) jaoks. Tüüpiliselt kasutatakse seda mitmekihiliste firma rakendustes. CORBA pakub ideaalset liidese defineerimise keelt (ingl. k. *Interface Definition Language*, edaspidi IDL), mis võimaldab defineerida loetavat objekt-orienteeritud API'sid.

Sellegi poolest on CORBA keeruline kuna vajab väga tõsist õppimist ennem kui kasutusele saab võtta. Sobib rohkem firmapõhistele rakendustele.

4.2.2. DCOM (Distributed Component Object Model)

Tegemist on Microsoft Corporation arendusega, mis pakub raamistiku, et integreerida komponente. Antud juhul on võimalik teistel arendajatel luua süsteeme, mis seovad omavahel uuesti kasutatavaid komponente erinevate tootjate vahel, mis siis suhtlevad läbi COM'i. DCOM on COM laiendus, mis võimaldab võrgupõhiste komponentide sidumist.

4.2.3. Java/RMI (Remote Method Invocation)

RMI on standard jaotatud objektide arvutamiseks Java's. See toimib Java ja Java tehnoloogia baasil töötavate rakenduste vahel, kus siis välis Java objekti meetodid saavad välja kutsuda teisest Java virtuaalsest masinast või teisest serverist.

RMI teisendab objekti jadaks, et koondada ja jagada parameetreid ilma neid tühjendamata. Seeläbi lubab ta ka objekt-orienteeritud polümorfismi.

Kasutatakse veebipõhistes sisevõrkudes ja üldistes hajutatud süsteemidega arhitektuuris.

4.3. Veebiteenused (Web services)

> **Definitsioon:** Veebi teenus on tarkvara süsteemi, mis on disainitud toetama opereerima masin-masin vaheliseks suhtluseks üle interneti [25].
> *(*ingl. k. *Definition: A Web service is a software system designed to support interoperable machine-to-machine interaction over a network.)*

Veebiteenuse all mõistetakse standardiseeritud suhtlusviisi läbi erinevate programmi rakenduste, mis toimivad erinevatel platvormidel või raamsüsteemidel, mille kaudu toimub kliendipoolne asünkroonne suhtlus masinate vahel. Suhtluse kirjeldamiseks kasutatakse XML struktuuri ja SOAP kirjeldusel põhinevat protokolli.

- **Toimib**
  - Üle HTTP protokolli
- **Koosneb**
  - SOAP (Simple Object Access Protocol);
  - UDDI (Universal Description, Discovery and Integration);
  - WSDL (Web Services Description Language);

- **Kasutatakse**
    - Andmete vahetamiseks erinevate rakenduste ja platvormide vahel;
    - Integreerimaks rakenduste andmeid linkides omavahel andmeid;
    - Korduvate komponentide pakkumine läbi teenuse
      Nt.: valuuta konverteerimine, ilmateate raportid või tõlkimine jne.;

4.3.1. SOAP (Simple Object Access Protocol)

Tegemist on kommunikatsiooni protokolliga, et suhelda erinevate rakendustega.
SOAP arendati välja peale seda, kui tekkisid probleemid turvalisuse ja sidususega RPC andmevahetuses. Kuna HTTP protokollil puudusid taolised probleemid, siis arendati selle tarbeks SOAP, sest see toimib enamikes veebiserverites ja sirvijates.

Toimib üle HTTP protokolli ja tugineb XML keelel. SOAP teada sisaldab SOAP päist (ingl. k. *header*) blokki ja SOAP sisu (ingl. k. *body*)

SOAP päringu formaat:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2007-04-20T14:00:00-08:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

4.3.2. WSDL (Web Services Description Language)

WSDL on XML dokument, et kirjeldada veebiteenuseid ja nende kasutamise võimalustest. Sellega saab spetsifitseerida teenuse asukohta ja operatsioone (või meetodeid) mida teenus pakub.

4.3.3. UDDI (Universal Description, Discovery and Integration)

UDDI kasutatakse ärides, kus saab registreerida ja otsida veebiteenuseid, kui kataloogi teenusena. See on kataloog kuhu saab salvestada veebiteenuste informatsiooni. Kommunikatsioon toimub läbi SOAP. Antud kataloogi kirjeldatakse WSDL abil.

UDDI aitab lahendada järgnevaid probleeme:

- Võimaldab leida just õige äri võimalikest miljonitest internetti ühendanute vahel;
- Defineerib kuidas on võimalik aktiveerida teenust, kui vajalik äri on leitud;
- Jõudes uue kliendini ja suurendades ligipääsu antud kasutajatele;
- Suurendades pakkumisi ja laiendatakse turu haardeulatust;
- Lahendades kliendist tulenevaid vajadusi eemaldades takistusi lubades kiiret osavõttu globaalses interneti äris;
- Kirjeldades teenuste ja äride protsesse programmiliselt ühes, avatud ja turvalises keskkonnas;

Näiteks:

Lennuhindade kontroll ja reserveerimissüsteem , kus lennufirmad saavad registreerida oma teenused UDDI kataloogi. Sealt aga on võimalik reisifirmadel otsida lennufirmade reserveerimise kasutajaliidest. Kui see leitakse, siis reisifirma saab kohe kommunikeeruda selle teenusega otse kuna kasutatakse täpselt defineeritud reserveerimise kasutajaliidest [26].

4.4. JSON (JavaScript Object Notation)

Koduleht asub aadressil (vt. http://www.json.org/)

JSON on andmevahetus formaat, mis võimaldab inimloetavalt kirjeldada objekte ja teisi vastavaid andmestruktuure. See on teksti formaat, mis on täielikult keelest sõltumatu. Antud formaat sobib igasse olukorda kus on vaja rakendustel on vaja omavahel andmeid vahetada või salvestada struktureeritud andmeid, kui tekst.

4.4.1. Eesmärk

Ta on lihtne alternatiiv XML kasutusele, et teostada asünkroonset andmevahetust kliendi ja serveri masina vahel, kus toimub siis n.ö. jadaks teisendamine (*inglise k. serialize, unserialize*). Põhi rakendust on ta leidnud Ajax veebi rakenduste programmeerimisel.

4.4.2. XML v. JSON

Erinevus XML ja JSON vahel on selles, et XML eesmärgiks oli kirjeldada struktureeritud andmeid aga JSON eesmärk on lihtsustada ja kiirendada andmete edastamist nii serveri kui kliendi vahel. Seega on XML tunduvalt keerukama süntaksiga, kui JSON.

Näide kuidas samasuguseid andmeid kuvatakse XML ja kuidas JSON'iga.

XML päringu formaat

```
<?xml version='1.0' encoding='UTF-8'?>
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

JSON päringu formaat

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

4.5. YAML (Yet Another Markup Language)

Koduleht asub aadressil (vt. http://www.yaml.org/)

Alguses võeti YAML kui „veel üks kirjeldus keel" aga nüüdseks on sellest välja arenenud täiesti arvestatav uus keel andmete edastamiseks ja nüüdseks kirjeldatakse seda ka kui „YAML ei ole kirjeldus keel" (ingl. k. *YAML Ain't Markup Language*).

Eesmärk oli luua rohkem andmetekeskne, kui dokumendi põhine keel. Selle asemel, et kasutada XML andmete teisendamisega, tasuks just kasutada YAML, kui „kergkaalu kirjeldus keelt".

4.5.1. Eesmärk

YAML on arvuti poolt töödeldav andmete teisendus formaat ja disainitud inimloetavaks. Integreeritav kirjakeeltega nagu Perl ja Python. Optimiseeritud andmete teisenduseks, konfigureerimise seadete, log failide, interneti teadete ja filtreerimise jaoks.

YAML iseloomustavad omadused:

- Ta on hästi loetav ja on eelkõige mõeldud nimekirjade, võõrsõnastike, teksti, numbrite jada jne. talletamiseks.
- Järjekindel infomudel;
- Ühtib hästi teiste kirjakeeltega;
- Jadal-põhinev töötlemine (ingl. k. *Stream-Based Proccesing*)
- Väljendusrikas ja laiendatav;
- Lihtne kasutusele võtta;
- Saab hakkama enamike andmetüüpidega ja struktuuridega;

YAML lugemiseks ja kirjutamiseks on laiendus nimega Syck
(vt. http://whytheluckystiff.net/syck/), mis on välja arendatud Ruby poolt.

4.5.2. YAML v. JSON

Peamine erinevus seisneb selles, et YAML kasutab tühikuid tekstis. Samuti kasutab YAML andmete kirjeldamiseks taandridu. Tuleb välja, et JSON on YAML poolt täielikult aktsepteeritav (v.a YAML ei luba /* */ kommenteerimist, seega vajab ta eelnevat töötlemist) kui ei kasutata tühimike ja taandridu [27].

JSON koodi näide

```
{ 'foo' => 'whatever',
  'bar' => [
    {
        'fruit' => 'apple',
        'name' => 'steve',
        'sport' => 'baseball'
    },
    'more',
    {
        'python' => 'rocks',
        'perl' => 'papers',
        'ruby' => 'scissorses'
    }
  ]
}
```

YAML koodi näide

```
foo: whatever
bar:
 -
   fruit: apple
   name: steve
   sport: baseball
 - more
 -
   python: rocks
   perl: papers
   ruby: scissorses
```

5. Arhitektuuri mustrid (Architectural patterns)

- Mis on arhitektuuri muster?
- Mis ülesehituse mustreid on kasutatud?
- Mis on mustrite eesmärk?

Programmi arhitektuuri muster määrab väga palju ära kuidas on võimalik antud rakendust kasutada ning kui hästi ta suudab lahendada mingeid tüüp situatsioone.

Võrreldes ülesehituse mustriga vaadeldakse süsteemi tunduvalt laiemalt. Arhitektuur võib koosneda väga mitmetest erinevatest ülesehituse mustritest, mis määravad ära kogu süsteemi toimimise.

Antud punktis on väga tähtis eesmärk või probleemi tuvastamine, et mida soovitakse saavutada ja kuidas. Selles faasis defineeritakse ära alamsüsteemid, seosed ja vastutused.

Järgnevalt toon välja põhilisemad ja enamkasutatavad mustrid. Nende kasutus sõltub vajadustest ja eesmärkidest, millest kirjutan lähemalt. Seega on vaja arhitektuuri paika panemiseks käia läbi tüüpsituatsioonid või probleemid millega rakendus peab tegelema.

5.1. Rakenduse automatiseerimise kiht (Application Automation Layer)

Eesmärk on automatiseerida mingi osa rakendusest, et lihtsustada ja toetada arendustööd. Samuti kasutatakse taolist lähenemist suurte projektide puhul, kus tingimused võivad muutuda aja jooksul, kas siis tehnoloogia uueneb või kliendi soovid ja nõuded muutuvad [28].

Rakenduse automatiseerimise kihi eelised projekti täitmisel on:

- Komponentide põhine arendus;

- Silutud (ingl. k. *debugged*) ja testitud komponentide kasutamine;
- Sisse ehitatud vahend, mis lubavad jälgida kõiki tegevusi;

Rakenduse automatiseerimise kiht koosneb mitmest tehnoloogiast:

- Andmejaotur (ingl. k. *Data Hub*);
- Protsessihaldur (ingl. k. *Process Manager*);
- Komponendihaldur (ingl. k. *Component Manager*);
- Vahendite pakett (ingl. k. *Instrumentation Package*);

Täiendavad featuurid on n.ö. „tehnoloogilised komponendid":

- Graafilise kasutaja liidese kontrollerid (ingl. k. *GUI controls*);
- Andmebaasi kasutajaliides (ingl. k. *Database Interface*);
- Olekuhaldur (ingl. k. *State Manager*);
- Täiendavad tehnoloogiliste komponentide liidesed (ingl. k. *Additional technology component interface*);

5.2. ORM (Object-Relational Mapping)

Selle all mõistetakse programmeerimise tehnikat, et siduda omavahel mitte ühilduvate andmebaaside andmeid ja objekt-orienteeritud keel. Antud lähenemine annab võimaluse suhelda andmebaasidega läbi objektide ja objektidevahelise seoste. Sellised objektid on võimelised looma antud päringuid ning tegevusi mida tuleb rakenduses täita. Alternatiivina oleks kirjutada andmebaasi päringuid iseseisvalt iga tingimuse kohta ise. Peamised andmebaasi tegevused on defineeritud CRUD[1] alusel.

Andmebaaside seoseid kaardistatakse erinevalt vastavalt eesmärgile kas vertikaalne, horisontaalne või filtreeritud.

- **Vertikaalne** - siin on tegemist omavahel erinevate seotud klassidega ja iga klassi leht on seotud oma põhi klassiga;
- **Horisontaalne** – erinevad klassi, mis on iseseisvad ja nad ei oma kokkupuute punkte;
- **Filtreeritud** – üks klass mille andmed on kuvatud valikuliselt;

[1] CRUD sisaldab Lisa (ingl. k. *Create*), Loe (ingl. k. *Read*), Uuenda (ingl. k. *Update*) ja Kustuta (ingl. k. *Delete*) tegevusi.

Arvestada tuleb võimalike seostega tabelite vahel läbi viida. Võimalikud seosed on **üks-ühele**, **üks-mitmele**, **mitu-mitmele**. Lisaks veel **viidatud** seosed mille korral on olemas küll viit teise klass aga ei omata teisi klasse, et muudatusi edasi kanda ja **kontroll klass**, mis on võimeline muudatuse korral seotud klassidesse muudatusi edasi kandma. Taolisi objekte nimetatakse aktiivne salvestus objekt (ingl. k. *Active Records*).

5.3. SOA (Service Oriented Architecture)

Veebiteenuseid saab rakendada ka arhitektuuris nagu teenusele-orienteeritud arhitektuur (SOA), kus peamine kommunikatsiooni vahend on teade, mitte kui operatsioon. Tihti on viidatud seda ka kui teatele-orienteeritud (ingl. k. *Message-Oriented*) teenus. Peamine suhtlus tugineb XML formaadile ja sellele tuginevatele tehnoloogiatele.

5.4. MVC (Model-View-Control)

Kasutatakse lihtsate GUI rakendustes. Antud arhitektuur on sündmusest-juhitud, mis tähendab, et kõik tegevused saavad alguse mingist tegevusest, mis võib olla omakorda jällegi ajendatud mingist muust sündmusest [29].

Antud arhitektuur koosneb kolmest osast:

- **Mudel (Model)** – sisaldab informatsiooni mudelit või vahendab andmeid teistele objektidele ja komponentidele;
- **Vaade (View)** – kasutatakse, et kuvada komponenti ehk siis tekitab väljundi. See võib olla osaline väljund nagu teksti väli või nupp;
- **Kontroller (Controller)** – juhib ja haldab tegevusi;

Kuna selle kohta on väga palju informatsiooni erinevate raamsüsteemide dokumentatsioonis kui ka eraldi seega ei peatu ma sellel pikemalt.

5.5. PAC (Presentation-abstraction-control)

Edasiarendus MVC (Mode-View-Controller) arhitektuurist. MVC on piiratud lihtsusega Graafiline kasutaja liides ühe või enama vaatega (ingl. k. *View*) sama mudelit (ingl. k. *Model*) kasutades [30].

Iga PAC komponent koosneb järgmistest elementidest:

- **Esitlus (Presentation)** – on täpselt sama nagu MVC mustris on vaade. Kuvab informatsiooni üldistus elemendist;
- **Üldistus (Abstraction)** – sisaldab andmeid nagu MVC mustris. Samas aga võib ta olla ainuke üks väike osa kogu rakenduse andmestruktuurist ja ei oma aktiivset osa muudatuste korral;
- **Juhtimine (Control)** – on enamvähem sama tähendusega nagu MVC kontroller (ingl. k. *Controller*) juhtides väliseid sündmusi ja uuendades mudelit. Samas uuendab ta ka otse presentatsiooni osa. MVC mustrist erineb ta selle poolest, et muudatused antakse edasi tema vanem (ingl. k. *Parent*) PAC komponendile;

5.6. Kolmekihiline programmi arhitektuur (Three-tier Software Architecture)

Spetsiaalne kliendi ja serveri vaheline arhitektuur, mis koosneb kolmest eraldi defineeritud protsessist, mis jooksevad erinevatel platvormidel.

- **Kasutajaliides (User Interface)** – väljund mida kasutaja näeb ja saab kasutada ning toimib **kasutaja masinas**;
- **Protsessi haldur (Process Managment)** – vahekiht **rakendus serveris**, mis sisaldab kogu vajaminevat loogikat ja mis omakord suhtleb andmebaasi ja kasutajaliidesega ning täidab vastavaid ärireegleid;
- **Andmebaasi haldur (Database Managment)** – erinevad andmebaasi mootorid ja nende driverid, mida siis kutsutakse **andmebaasi serveriks**;

5.7. Sündmusest-juhitud programmeerimine (Event-driven programming)

Sündmusest-juhitud programmeerimine on paindlik viis lubada programmil reageerida mitmele erinevale sisendile ja sündmusele [31].

Sündmusest-juhitud programmeerimise korral on võimalik kinni püüda kasutaja poolt aktiveeritud tegevusi. Staatiliste veebide puhul nagu HTML toimuvad taolised tegevused kasutaja poolel ja selle kinni püüdmiseks on Javascript.

Sündmustest-juhitud programm toimib põhimõttel:

- Sündmusest juhitud programm sisaldab tsüklit, mis püüab tegevuse kinni ja edastab
- Sündmuse tsüklid avastavad ainult sündmuse aga mitte seda mida nad sisaldavad
- Sündmuse haldur tegeleb ühe sündmusega korraga

Näide: Javascriptiga edastakse sündmuse tegevus

```
<script type="text/javascript">
function syndmus()
{
   alert("Saime kätte kirje: " + document.sisu.kirje.value);
}
</script>
<form name='sisu'>
   <input type='text' name='kirje' onChange='syndmus()'>
</form>
```

Kuidas aga oleks võimalik antud sündmusi edastada rakendustesse?

Lähtume sellest, et rakendus koosneb kasutajaliidesest ja serveri poolsest rakendusest. Et sündmus jõuaks kasutajalt serverisse ja sealt edasi rakendusse ja tagasi peab olema suhtluskanal. Suhtluskanaliks sobib väga hästi Ajax kasutamine. Prototype saame võtta aluseks, et luua kasutajaliidese poolne rakendus, mis edastaks andmed rakendusele. Meil on vaja kinni püüd tegevused mis toimuvad kasutajaliideses. Selle jaoks on Prototype loonud sündmuse (ingl. k. *Event*, edaspidi Event) klassi. Antud klass jälgib mingi objektiga toimuvaid tegevusi.

JavaScripti sündmused määravad ka veebi raamsüsteemi võimekuse midagi teha. Antud tegevused on toodud alljärgnevalt:

- **onabort** – pildi laadimine on katkestatud
- **onblur** – element kaotab fookuse
- **onchange** – muudab mingi välja sisu andmeid
- **onclick** – hiirele on tehtud klikk ehk nupuvajutus
- **ondblclick** – hiirega on tehtud kahekordne nupuvajutus
- **onerror** – kui lehe või pildi laadimisega on tekkinud viga
- **onfocus** – element satub fookusesse

- **onkeydown** – klaviatuuri nuppu on alla vajutatud
- **onkeypress** – klaviatuuri nuppu on vajutatud
- **onkeyup** – klaviatuuri nupp on lahti lastud
- **onload** – kui leht või pilt on laetud
- **onmousedown** – hiire nupp on alla vajutatud
- **onmousemove** – hiirt on liigutatud
- **onmouseout** – hiir on liikunud üle elemendi välja
- **onmouseover** – kui hiir on sattunud elemendi peale
- **onmouseup** – kui hiire nupp on lahti lastud
- **onreset** – reset nuppu on vajutatud
- **onresize** – aknal on muudetud suurust
- **onselect** – antud teksti on selekteeritud
- **onsubmit** – kinnitus nuppu on vajutatud
- **onunload** – lehekülg suletakse (tegelikult tähendab see lehelt lahkumist nt. lehe uuendamisel või uuele lehele minemist)

Javascripti õppematerjal (vt. http://www.w3schools.com/jsref/jsref_events.asp)

5.8. Tellingud (Scaffold)

Tellingud on meetod andmebaasi toetusega rakendustele. Seda tehnikat toetavad mõned Model-View-Controller raamistikud, kus programmeerija saab kirjutada spetsifikatsiooni kuidas andmebaasi kasutada. Rakendus saab kasutada seda spetsifikatsiooni, et kasutada andmebaasi kirjete lisa (ingl. k. *create*), loe (ingl. k. *read*), uuenda (ingl. k. *update*) ja kustuta (ingl. k. *delete*) tegevusi andmebaasi kirjete puhul.

Tihti kutsutakse antud objekte ka aktiivsed andmeobjektideks (ingl. k. *Active Record*), mis on vaheliidesed andmebaasi ja loogika vahel võimaldades teha tabelile iseloomulike tegevusi nagu lisamine, lugemine, uuendamine ja kustutamine. Seda kõike aga objektina ja läbi meetodite. Põhimõtteliselt on vaja andmebaasi tabeliga suhtlemiseks luua ainult üks klass, mis on laiendatud ActiveRecord klassiga. See annab eelnevalt tehtud klassile võimaluse kõikidele nendele tegevustele.

6. Standardiseerimine

Koodi standardiseerimine on üks tähtsamaid osasid hea rakenduse loomisel. Tuleks järgida n.ö. hea tava kohast kodeerimist ja stiili. Lisaks igale keelele omasele koodi kirjutamisel lisandub tavaliselt rakenduse sees kirjutatava koodi nõuded kuidas seda tuleb teha. Nt. dokumenteerimine, testide kirjutamine, failide ja klasside vaheline seos jne.

Erinevatel keeltel on oma koodi kasutamise standardid:

- PHP koodi standard (vt. http://www.dagbladet.no/development/phpcodingstandard/)
- C++ koodi standard (vt. http://www.possibility.com/Cpp/CppCodingStandard.html)
- Java koodi standard (vt. http://www.cs.rit.edu/~f2y-grd/java-coding-standard.html)

6.1. Nimeruum (Namespaces)

Nimede komplekt või rühm, mis defineeritud mingi kindla kokkuleppe alusel.
Tegemist on abstraktse kirjeldusega, mis võimaldab jaotada osasid erinevatesse kontekstidesse. Üheks eesmärgiks on jaotada asju avalikeks (ingl. k. *public*) ja privaatseteks (ingl. k. *private*) aladeks. Asukohtadel ei tohi olla rohkem, kui üks tähendus. Iga asukoha puhul on tegemist unikaalse asjaga, sest muidu ei ole võimalik identifitseerida elementi.

Mängib üsna tähtsat rolli rakenduste loomisel ja arendustöödel, sest see määrab ära rakenduse arenduse nõuded, et kuidas midagi tuleb teha.

Asukohta kasutatakse erinevates tingimustes. On olemas lame (ingl. k. *flat*) ja hierarhiline (ingl. k. *hierarchy*) nimeruum. Neid kasutataks erinevates tingimustes, mis on toodud alljärgnevalt.

Näiteks:

- Pildi asukohad on lame nimeruum (täismõõdud ja väikesed)
- Kataloogide asukohad hierarhiline nimeruum (grupeerimiseks ja funktsionaalsuste jaotamiseks)
- Kategooriate asukohad hierarhiline nimeruum (navigeerimiseks)
- Abi ja eelistuste lehed on lame nimeruum (kui eesmärgiks on kuvamine)
- Internetis on hierarhiline nimeruum, mis jaotub tipptaseme domeenideks (.ee, .com, .net jne.), teise taseme domeeniks, kolmanda jne.
- Erinevad operatsioonisüsteemid kasutavad erinevat kataloogi puud (ühel on „/ linux” ja teisel on „\ windows”)

Üheks nimeruumi kasutamise eesmärgiks on ühtlustada üle operatsioonisüsteemide failide kasutamist.

6.2. Dokumenteerimine (Documentation)

Iga kood tuleks dokumenteerida ja sama käib raamsüsteemi kohta. Puuduliku dokumentatsiooni korral on raskendatud raamsüsteemi tundma õppimine ja kasutamine. Dokumenteerimiseks on erinevad standardid ja ettekirjutused. Koodi dokumenteerimise näide on toodud lisa 11. Selle dokumenteerimiseks on kasutatud phpDocumenter põhimõtet.

7. Turvalisus

Ei ole võimalik tänapäeval, et me ei puutu kokku turvalisuse küsimusega. Oli aegu tagasi, kus interneti lehekülgedel oli võimalik väga lihtsate vahenditega varastada teise kasutaja identiteet ja kõik muu sellega seonduv. Lisaks sellele oli võimalik rikkuda veebilehti ilma administraatori liidest kasutamata jne. Teen hetkel ülevaate kahest raamsüsteemiga kokkupuutuvat turvalisuse riski nagu Cross-site request forgery ja Cross-site Scripting Attack mida on võimalik vältida. Mõned turvalisuse riskid:

- SQL süst (ingl. k. *SQL injection),* mille korral lisatakse SQL lausesse teine SQL lause;
- peidetud skripti rünnak (ingl. k. *Embedded Script Attack*) mille korral on võimalik lisada avalikesse foorumitesse või kommentaaridesse skripte, mis takistavad või häirivad veebilehe tavapärast tööd.;
- illegaalsed sümbolid (ingl. k. *Illegal symbols*) kus üritatakse muuta lehekülje sisu sisestades läbi avalike postituste illegaalseid koodi osasid HTML või muu näol;

Turvalisuse riske on olemas veel teisigi, millest antud töös ei jõua kirjutada.

7.1. Cross-site request forgery (CSRF or XSRF)

Antud rünnet teatakse ka kui ühe nupu vajutuse rünnak või sessiooniga ratsutamine. Tegemist on teiste kasutajate vastu suunatud ründega, kus varastatakse nende identiteet või privileegid,

et kasutada ohvri asemel vajalike funktsionaalsusi. Selle teostamiseks on vaja, et ohver laeks lehe, mis sisaldab kuritahtlikke päringuid.

Näiteks: Muutmaks kasutaja konto parooli või andmeid, sooritada oste, suhelda variisikuna teiste kasutajatega jne. See on üsna palju seotud „Cross-site Scripting" (XSS), kasutades sama tehnikat sessiooni varastamiseks.

Lahendus

(vt. http://en.wikipedia.org/wiki/Cross-site_request_forgery)

7.2. Cross-site Scripting Attack (XSS or CSS)

Tegemist on veebilehe haavatavusega mille tagajärjel on võimalik veebilehele paigutada kuvamiseks sisusid, mis on kasutaja poolt edastatud ning mis ei kuulu antud veebilehe konteksti.

Näiteks:

Ründaja võib panna veebilehe lingi, mis sisaldab kuritahtliku skripti kuhugi avalikku foorumisse. Selle skripti eesmärgiks on rünnata teisi foorumi kasutajaid, kes sellele lingile vajutavad, mille tagajärjel võib antud skript varastada vastava kasutaja küpsise. Antud küpsise varastamisel on võimalik kätte saada kasutaja sessiooni identifikaatori, mille kasutamisel saab kaaberdaja endale võtta teise kasutaja sessiooni, mis võimaldab omakorda teostada „Cross-site request forgery".

Lahendus

(vt.http://www.imperva.com/application_defense_center/glossary/attack_prevention/cross_site_scripting.html)

8. Koodi optimiseerimine

Surnud koodi elimineerimine (ingl. k. *Dead code elimination***)**

See sisaldab endas koodi osade eemaldamist täielikult milleni rakendus kunagi ei jõua või ei saa kasutada .

Näiteks:

```
if(0){ };
```

Konstantide summeerimine (ingl. k. *Constant-folding***)**

Kui kasutatakse grupp konstante või mitte korduvaid elemente mida tuleb omavahel arvutada, siis on sul võimalik seda tegevust viia ühe kordse tegevuseni, kus toimub ainult defineerimine.

Näiteks:

```
$sekundid = 24*60*60;
```

Siis on võimalik seda eelnevalt arvutada ja anda kohe vastus:

```
$sekundid = 86400;
```

Mustaaugu optimiseerimine (ingl. k. *Peephole optimizations***)**

Lokaalsed optimiseerimised, mida on võimalik rakendada, et muuta koodi efektiivsust.

Näiteks:

```
$i = $count++;
```

Taoline arvutus muudab $count väärtust alles pärast seda, kui on antud väärtus edasi $i muutujale. See aga tähendab sisemiselt, et keel või vastav platvorm on sunnitud meelde jätma $count väärtuse. Tunduvalt otstarbekam oleks see, kui antud muutujat ei ole vaja hiljem kasutada teha ennem arvutus ja seejärel tagastada teisele muutujale

Näiteks:

```
$i = ++$count;
```

8.1. Väljundi sisu pakkimine (Output content compression)

Kasutades mod_gzip lahendust on võimalik 30% vähendada ülekandekiiruse utiliseerimisel (ingl. k. *bandwith utilized*) ja samas saadi üleüldine tulemuse tõus peaaegu 10%, kui ilma lehe sisu pakkimiseta [13].

Seega kasutades lehe päises päringut, mis annab sirvijale teada, et sisu on kokku pakitud:

```
Content-Encoding: gzip, deflate
```

Vajab sirvija poolset aktsepteerimist, et kokku pakitud andmeid vastu võtta ja uuesti lahti pakkida.

8.2. Puhvrisõbralik rakendus (Cache-friendly application)

Et saada kasu vahesalvestusest ja sirvija vahemälust, siis tasub oma rakendused muuta selle vastavaks kasutades HTTP päiseid. Seda kasutades on võimalik rakendustel seadistada sirvijaid nii, et vahesalvestused ka eesmärki täidaksid. Eemärgiks on vähendada korduvaid päringuid rakenduse suunas, mis koormaksid serveri tööd ja ressursside kasutust nagu andmebaas kasutus ja piltide edastamine.

Kuupäeva formaat peab olema järgmine:

```
D, d M Y H:i:s
```

- **Last-Modified** : aeg millal viimati lehekülg loodi või muudeti. Toimib UTC (Universal Time Coordinated, eelnevalt GMT) aja järgi.
- **Expires** : aeg millal antud sisu ei ole ajakohane.
- **Pragma**:
    - no-cache – tuleks määrata kui ei soovita objekte meelde jätta
- **Cache-Control:**
    - **public** – vastus on salvestatud kõigi kasutajate poolt;
    - **private** – vastus on salvestatud ainult sama kasutaja poolt;
    - **no-cache** – ei salvestata mitte mingil juhul. Kasutada kui andmed on äärmiselt sensitiivsed;
    - **must-revalidate -** kõik lehe andmed vahemälus tuleb uuendada;
    - **proxy-revalidate** – ainult kõigile jagatud andmeid tuleb uuendada;
    - **max-age** = „sekundid” - aeg kui kaua hoitakse andmed mälus;
    - **s-maxage** = „sekundid” – maksimaalne aeg kui kaua hoitakse kõigile kätte saadavat vahesalvestust meeles;

Cache-Control parameetreid on võimalik omavahel siduda eraldades neid komaga.
Salvestatakse ühe kasutaja andmed kuni 1 minutiks ehk 60 sekundiks php koodiga

```
<?php
$now = time();
$gmt_time = gmdate(,D, d M Y H:i:s',$now).' GMT';
$gmt_expire_time = gmdate(,D, d M Y H:i:s',$now+60).' GMT';
header(„Expires: $gmt_expire_time");
header(„Last Modified: $gmt_time „);
header(„Cache-Control: private, max-age: 60");?>
```

9. Ülevaade avatud lähtekoodiga raamsüsteemidest

Järgnevalt annan ülevaate praegu vabalt kätte saadavatest veebi raamsüsteemidest. Nende kõikide ühine eesmärk on lihtsustada ja toetada veebi rakenduste loomist ja arendamist. Kuna oleks väga keeruline ise üles märkida kuidas toimib mingi raamsüsteem, siis pigem toon näited selle kohta kuidas see raamistik kasutajale näib.

Oma töös ei kavatse ma lihtsalt koostada nimekirja võimalustest mida antud rakendused lubavad, sest kindlasti on nad võimelised täitma just neid ülesandeid milleks nad on loodud. Iga raamsüsteemi üles panekuks kasutan dokumentatsiooni ja muid abimaterjale, mis antud kodulehtedele on üles pandud.

Lisaks on minu raamsüsteem avatud lähtekoodiga. Seega kui tahad lisa funktsionaalsusi, siis lihtsalt lisa need (ingl. k. *Plus my framework is open source, so if you want features, just add them* [8]).

Arvan, et avatud lähtekoodiga raamsüsteem on elujõulisem, sest seda saavad muuta ja kasutada kõik teised arendaja ning teha muudatusi või soovitusi, mis omakorda annab võimaluse arendada raamsüsteemi kiiremini ja efektiivsemalt kui kinnise koodiga. Seetõttu kasutan oma töös ainult avatud lähtekoodiga raamsüsteemide.

Rakendused on loodud Windows keskkonnas.

9.1. Prado

Koduleht asub aadressil (vt. http://www.pradosoft.com/)

9.1.1. Kirjeldus

PRADO (**P**HP **R**apid **A**pplication **D**evelopment **O**bject-oriented) on komponentidel põhinev ja sündmustest juhitud programmeerimise raamsüsteem, et arendada veebi rakendusi. Kasutab MVC programmeerimise mustrit. Rakenduse globaalsete parameetrite seadistamiseks on

võimalik kasutada XML faili „application.xml”. Lõppkasutaja suhtleb otse leheküljega (ingl. k. *page*). Raamsüsteem toetab CLI ja sisse on ehitatud Prototype ja Scriptaculouse.

9.1.2. Struktuur

Prado raamsüsteemi failistruktuuri on näha Lisa 1.

9.1.3. Liivakast

Luua "helloworld” kataloog kuhugi veebi serverisse avalikult kättesaadavasse kohta. Seda on võimalik teha kasutades käsurea tööriista „prado-cli.php”. Mine kataloogi kuhu tahad helloworld kataloogi luua.

```
php ../prado/framework/prado-cli.php -c helloworld
```

Seejärel on tekkinud kataloog „helloworld”, mis peab koosnema järgmistest kataloogidest. Tegemist on minimaalse kataloogi kujuga.

- **`assets`** – kataloog avalikustatud privaatsetele failidele. Kataloog peab olema kirjutatav veebi serveri poolt.
- **`protected`** – rakenduse põhi kataloogi asukoht salvestamaks rakenduse infot ja privaatseid skripti faile. Kataloog peab olema konfigureeritud nii, et see ei oleks lõpp kasutajale otse kättesaadav.
- **`runtime`** – rakenduse käitusaja (ingl. k. *runtime*) asukoht, et salvestada rakenduse käitusaja informatsiooni nagu rakenduse seisund (ingl. k. *state*), vahemälu andmed jne.
- **`pages`** – põhi asukoht hoidmaks kõiki PRADO lehekülgi.

Prado „helloworld” rakenduse kataloogi struktuur on näha Lisa 2.

„Hello World” rakenduse jaoks on vaja kolme faili **index.php**, **Home.page** ja **Home.php**.

1. „index.php” on vajalik rakenduse algkäivituse tegemiseks

```
<?php
require_once('path/to/prado.php');  // include the prado script
$application=new TApplication;      // create a PRADO application instance
$application->run();                // run the application
?>
```

2. „Home.page” – templiit leht mida töödeldakse kuvamiseks e. presenteerimiseks. Võib sisaldada HTML formaati või Pradole iseloomulike märgendeid (ingl. k. *tag*'s), mis võivad sisaldada komponentide, templiidi konfigureerimise, kommenteerimise, dünaamilise sisu ja dünaamilise omaduse kuvamiseks.

```
<html>
  <body>
    <com:TForm>
      <com:TButton Text="Click me" OnClick="buttonClicked" />
    </com:TForm>
  </body>
</html>
```

3. „Home.php” – loogika fail, mis töötleb mingit tegevust

```
<?php
class Home extends TPage
{
    public function buttonClicked($sender,$param)
    {
        // $sender refers to the button component
        $sender->Text="Hello World!";
    }
}
?>
```

9.2. Code Igniter

Koduleht asub aadressil (vt. http://www.codeigniter.com/)

9.2.1. Kirjeldus

Code Igniter on üsna väike ja lihtne veebi raamistik, mis töötab alates PHP 4 versioonist. Antud raamsüsteem on väga sarnane Ruby on Railsile. Antud raamsüsteem toimib MVC programmi ülesehituse mustril. Vajab iga rakenduse jaoks eraldi kogu raamsüsteemi kasutamist ning rakenduse loomine toimub raamsüsteemi kõhu sisse. Antud rakenduse loomine on vaba igasugustest konfigureerimistest, käsurea kasutamistest. Rõhub lihtsusele ja kiirusele.

9.2.2. Struktuur

Failistruktuur on toodud Lisa 3.

9.2.3. Liivakast

Et luua näidis rakendus on vaja luua vajalikud kolm faili. Osad failid tuleb iseseisvalt käsitsi luua. Ennem seda on vaja, et oleks olemas Code Igniter kogu raamistiku failisüsteem olemas ja töötama. Antud raamistik tuleb panna kuhugi avaliku kohta koos index.php failiga. Muudatused tuleb teha Code Igniteri raamistiku sees olevas „system/application" kataloogis.

1. „index.php" faili, kus tehakse pöördumine Code Igniteri raamsüsteemi poole.

Antud faili sisu on toodud ära Lisa 10. Tegemist on üsna keerulise sisuga failiga ning vajab eraldi uurimist, et kuidas kasutada erinevate rakenduste kokku panemisel.

2. „system/application/controllers/helloworld.php" - kontroller

```
<?php
class Helloworld extends Controller
{
    function index()
    {
        $this->load->view('hello');
    }
}
?>
```

3. „system/application/views/hello.php" – templiit fail

```
<html>
<head>
<title>Hello World</title>
</head>
  <body>
      <h1>Hello World!</h1>
  </body>
</body>
```

9.3. Ruby on Rails

Koduleht asub aadressil (vt. http://www.rubyonrails.org/)

9.3.1. Kirjeldus

Ruby on Rails on avatud lähtekoodiga Ruby raamsüsteem, et luua veebi rakendusi, mis kasutavad andmebaase.

Tugevuseks loetakse, et kasutab Ruby keelt, mis on omalaadne OO keel sarnaste seas, mis võimaldab luua rakendusi väiksema hulga koodiga. See omakorda võimaldab kiiremini arendada ja vähem vigu teha.

Ruby on Rails ei sisalda XML stiilis konfigureerimise faile. Pigem lähtutakse koodi kirjutamisel, et arenduse ja avastuse käigus saadaks selgusele mida on vaja ja mis moodi teha. Ühesõnaga rakenduse kood sisaldab kõike seda mida on vaja, et kasutada andmebaase või muid raamsüsteemi osasid.

Raamsüsteemi loomiseks on vaja kasutada CLI liidest.

Arendustööd tuginevad MVC programmeerimise mustrile ja andmebaasi rakendustele. Lisaks sellele on vaadetele toeks abistajad, mis võimaldavad juba eelnevalt loodud funktsionaalsustega lisada HTML ja Javascripti funktsionaalsusi ilma suurema probleemita.

9.3.2. Struktuur

Ruby on Rails raamsüsteemi failistruktuur on ära toodud Lisa 5.

9.3.3. Liivakast

„helloworld" näidisrakenduse failistruktuur on lisatud Lisa 6.

Et luua rakendust on vaja failistruktuuri mida Ruby on Rails kasutab. Seda tuleb teha käsurealt ja „ruby/bin" kataloogis:

```
rails helloworld
```

Antud kataloog tehakse samasse kataloogi. Et näha antud rakendust, siis tuleb käivitada server mida saab teha „helloworld/script/" kataloogis käivitades serveri:

```
ruby ./server
```

Järgmisena tuleb luua kontroller käsurealt:

```
ruby ./script/generate controller Helloworld
```

Järgmisena tuleb luua kontrolleri sisse tegevuse tagastus lisade kontrollerisse vastavad read:

```
class HelloworldController < ApplicationController
        def index
                render :text => "Hello World"
        end
end
```

Et kasutada templiit faili kuvamiseks, siis tuleb see luua „helloworld\app\views\helloworld\" kataloogi. Nimeks panna „index.rhtml"

Hello World templiit failist!

Samal ajal kontrollerist tuleks eemaldada defineeritud „index” funktsioon.

NB! Kindlasti jälgida, et kõik rakendused oleksid Ruby on Rails „bin” kataloogis. Vastasel juhul võivad rakendused mitte töötada.

Tasub vaadata sarnaseid inglise keelseid õpetusi ja juhendeid aadressilt:
(vt. http://www.rubyonrails.org/screencasts)

9.4. Zope

Koduleht asub aadressil (vt. http://www.zope.org)

9.4.1. Kirjeldus

Zope on avatud lähtekoodiga Pythoni raamsüsteem millega saab luua sisu haldus süsteeme, sise, portaali ja individuaalseid rakendusi.

Tugevuseks on arenduse keskkond, mis võimaldab üle interneti teha muudatusi oma Zope rakendustes. Sisaldab sisse ehitatud turva mudelit, mis annab võimaluse saada veebi osade üle kontrolli organisatsiooni või isiku tasandil.

Lisaks sisaldab Zope iseenda HTTP, FTP, WebDAV ja XML-RPC haldamise võimalust.

Zope sees on DTML (ingl. k. *Document Template Markup Language*), mis võimaldab turvalise skripti abil genereerida Zope objekte ja kasutada dünaamilist sisu.

9.4.2. Struktuur

Zope raamsüsteem failistruktuur on toodud lisa 7. Antud struktuur on üsna suur ja lai. Selle detailne uurimine võtaks suure osa ajast.

9.4.3. Liivakast

DTML kasutades kontrollitakse kas kasutaja „Fred” on sisse loginud.

Näide:

```
The title of this document is: <!--#var document_title-->.
<p>
<!--#if "AUTHENTICATED_USER=='Fred'"-->
  Hello Fred!
```

```
<!--#else-->
  Hello stranger!
<!--#/if-->
```

9.5. Prototype

Koduleht asub aadressil (vt. http://www.prototypejs.org/)

9.5.1. Kirjeldus

Prototype on Javascripti infoobjektide kogum e. teek. Tegemist on standardse klasside kogumiga, et luua rikkalikke ja interaktiivseid veebilehti. Selle raamsüsteemi välja arendamine on viinud paljude veebilehtede arenduse uuele tasemele lubades üsna kerge vaevaga arendada väga huvitavaid lahendusi.

Peamisteks omadusteks on klasside loomine ja Ajaxi klassi kasutamine. Selle abil on väga hõlbus kasutada HTML elemente. Lisaks viimastele on võimalik kasutada palju teisi funktsionaalsusi.

9.5.2. Struktuur

Prototype koosneb ainult ühest failist kuhu on kõik objektid sisse kirjutatud. See annab võimaluse saada kätte kõik asjad korraga mis vajalik. Taoline lähenemine aga võtab natukene rohkem ressurssi kui eraldi vastavaid klasse ja objekte sisse laadida.

9.5.3. Liivakast

Prototype kasutamiseks piisab kui lisada javascripti kood oma leheküljele.

Ajax rakendus toimib lokaalses masinas:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/2002/REC-xhtml1-20020801/DTD/xhtml1-
transitional.dtd">
<html>
<head>
<script type="text/javascript" src="/js/prototype.js"></script>
</head>
<body>
</body>
</html>
```

Kuidas püüda kinni sündmused ja edastada need Ajaxiga rakendusele ja tagasi.
Selle jaoks on vaja kasutada sündmuse jälgimise klassi ja Ajax klassi selle edastamiseks vastava rakenduse poole.

Näide: Prototype püüab kinni veebilehe akna laadimise sündmuse ja seejärel pöördub teise lehe poole

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/2002/REC-xhtml1-20020801/DTD/xhtml1-
transitional.dtd">
<html>
<script src="javascripts/prototype.js" type="text/javascript"></script>
<script type="text/javascript">
<script>
/*<![CDATA[*/
Event.observe(window, 'load', function()
        {
            Ajax.Updater
                (
                        {success:'result'},
                        window.location.href,
                        {
                                parameters:'&rs=UserPage.onLoad'
                        }
                );
        }
);
</script>
<body>
Tere!
<div id="result" style="display:none"><div>
</body>
</html>
```

9.6. Script.aculo.us

Koduleht asub aadressil (vt. http://script.aculo.us/)

9.6.1. Kirjeldus

Script.aculo.us on uue põlvkonna javascripti raamistik, mis baseerub Prototype raamistikul, et luua uusi ja huvitavaid interaktiivseid kasutajaliideseid.

Antud raamistik annab võimaluse luua visuaalseid efekte HTML/CSS kuvamise tarbeks. Sisse on ehitatud tegevuste kontroll vahend, et luua lohistatavaid (ingl. k. *draggable*), kukutatavaid (ingl. k. *droppable*), sorteeritavaid (ingl. k. s*ortable*), libistatavaid (ingl. k. *slider*) objekte.

Lisaks sellele on võimalik luua dünaamiliselt DOM elemente. Kõigi nende rakenduste testimiseks on olemas komponentide testimise (ingl. k. *Unit Testing*) vahend mida on võimalik valikuliselt lisada lehele juurde, et kontrollida rakenduste töökindlust ja toimimist mingitest tingimustes.

9.6.2. Struktuur

Script.acolo.us raamsüsteemi failistruktuur on ära toodud Lisa 8.

- **scriptaculous.js** – laeb ülejäänud failid juurde ja kontrollib, et prototype oleks vähemalt versioon 1.5 või uuem;
- **builder.js** – saab luua DOM elemente;
- **effects.js** – saab kuvada ja peita elemente erineval viisil;
- **dragdrop.js** – võimaldab hallata liigutamist ja elemendi paigutamist;
- **controls.js** – sisaldab ise lõpetavat (ingl. k. *autocomplete*) teksti välja; samas kohas editeerimist Ajax'i abil;
- **slider.js** – libistamise abivahend, et liigutada hiirega objekte;
- **unittest.js** – testimise keskkond;

9.6.3. Liivakast

Kasutusõpetus (vt. http://wiki.script.aculo.us/scriptaculous/show/Usage)

Näide: div kihi peitmine ja kuvamine:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/2002/REC-xhtml1-20020801/DTD/xhtml1-
transitional.dtd">
<html>
<head>
<script type="text/javascript" src="/js/prototype.js"></script>
<script type="text/javascript" src="/js/scriptaculous.js"></script>
</head>
<body>
  <input type="button" name="peida" value="Näita/peida" onclick="new
Effect.toggle($('sisu'), 'blind',{duration: 2});return false;" />
  <div id="sisu">
    Peida ja kuva sisu.
  </div>
</body>
</html>
```

9.7. ADOdb

Koduleht asub aadressil (vt. http://adodb.sourceforge.net/)

9.7.1. Kirjeldus

ADOdb (ActiveX Data Objects) on andmebaasi abstraktsiooni teek. On olemas nii PHP kui ka Pythoni toetusega versioon. ADOdb sisaldab

- Toetab väga mitmeid erinevaid andmebaase nagu: *MySQL, PostgreSQL, Interbase, Firebird, Informix, Oracle, MS SQL, Foxpro, Access, ADO, Sybase, FrontBase, DB2, SAP DB, SQLite, Netezza, LDAP, and generic ODBC, ODBTP. The Sybase, Informix, FrontBase and PostgreSQL, Netezza, LDAP, ODBTP jne.*
- Toetab *laialdast porditavust* (ingl. k. exstensive *portability*) nagu kuupäeva ja tüüpkohtlemine
- Disainitud võimalikult kiireks.
- Lihtne õppida. Eriti kui on vaja kasutada ADO ühendust Windowsi masinas.
- Sisaldab ORM kasutades ADOdb_Acitve_Records klassi.

9.7.2. Struktuur

Raamsüsteemi failistruktuur on ära toodud lisa 9.

9.7.3. Liivakast

Näide: Ühenduse tegemine eraldi parameetritega:

```
<?php
include('adodb/adodb.inc.php');
$db = ADONewConnection($dbdriver); //ab draiverid 'mysql' or 'postgres' jne
.
$db->debug = true; // kuvatakse päringud välja
$db->Connect($server, $user, $password, $database);
if(!$db)
{
    die(„Puudub andmebaasi ühendus”);
}
$rs = $db-
>Execute('select * from some_small_table where key=?', array($key));
if(!$rs)
{
   die($db->ErrorMsg());
}
print "<pre>";
while(!$rs->EOF)
{
   var_dump($rs->fields);
   $rs->MoveNext();
}
print "</pre>";
?>
```

Näide: Ühenduse tegemine „DSN” (ingl. k. *Data Source Name*) kirje abil:

```php
<?php
include('adodb/adodb.inc.php');
// $dsn = $draiver://$kasutaja:$parool@server/$ab?options[=value]
$db = NewADOConnection($dsn);
$db->debug = true; // kuvatakse päringud välja
if(!$db)
{
    die(„Puudub andmebaasi ühendus”);
}
$rs = $db->Execute('select * from some_small_table');
if(!$rs)
{
   die($db->ErrorMsg());
}

print "<pre>";
while(!$rs->EOF)
{
   var_dump($rs->fields);
   $rs->MoveNext();
}
print "</pre>";
?>
```

Antud näidete põhjal on tegemist kõige kiiremini toimiv tsükkel kõigi andmete kuvamiseks.

10. Ülevaade ja süntees

Alljärgnevalt lahkan antud raamsüsteemide probleeme ja teen vajalike järeldusi.

Enamikel juhtudel tuleb rakendus luua raamsüsteemi enda sisse. Soov on aga eraldada rakendus raamsüsteemist, et vajadusel oleks võimalik uuendada ja arendada raamsüsteemi üle mitme rakenduse. Seda kasutab praegusel juhul *Ruby on Rails* aga ei kasuta *Code Igniter*, mis on esimese analoog.

Paljud raamsüsteemid kasutavad oma rakenduste loomiseks käsurida, et rakendust luua. Ühest küljest on see hea, et asju automatiseerida. Teisest küljest jätab ta arusaamatuks rakenduse ülesehitust ja vajadusi mida luua. Rakendus peaks arenema järkjärgult teadliku vajaduste poole mitte automaatselt suurte sammudena. Lisaks tähendab see seda, et rakenduse loomine on keeruline ja mahukas. Prado ja Ruby on Rails võimaldavad või eeldavad seda.

Paljudel rakendustel on palju toetusvahendeid nagu: abistaja (ingl. k. *helper*), kogum (ingl. k. *library*), ühenda (ingl. k. *plugin*), mudel (ingl. k. *model*), kontroller (ingl. k. *controller*), konfiguratsioon (ingl. k. *config*), viga (ingl. k. *error*), vahesalvestus (ingl. k. *caching*), vaated (ingl. k. *views*), andmebaas (ingl. k. *database*) jne. Kõik need tuleks võimaluse korral lihtsustada ja kokku viia ühtseks toimivaks tükiks. Ühenda ja

konfiguratsiooni saaks näiteks liita mudeli alla või kogumisse, sest kui vastavaid koode kasutada siis nad on faili sees konfigureeritud ja lokaliseeritud.

Sündmusest-juhitud programmeerimine on samuti hea viis hallata kasutajaliidese tööd läbi rakenduse ja vastupidi. Seda kasutab antud juhul *Prado* ning selles osas on ta üsna mugav kasutada. Samas antud rakenduste loomine on pealtnäha kerge aga samas kui on vaja hakata midagi keerulisemat rakenduse tasemel, siis võib hätta jääda. Puudub nähtav eraldus kasutajaliidese ja rakenduse vahel. Lihtsam on luua rakendus kus Javascript koos sündmusi juhtiva koodiga on kasutajaliideses ja toimimismehhanism eraldi.

Prado rakenduse tegemine on üsna lihtne ning ei vaja väga põhjaliku uurimustööd millegi loomiseks. Lihtsate asjadega saab kergesti hakkama. Keerulisemate lahenduste tegemine vajab lisauurimist.

Code Igniter on tõsiselt lihtne rakendus. Isiklikult ei meeldi mulle tema juures see, et rakendus ehitatakse raamsüsteemi sisse nii, et viimast oleks raske uuendada või välja vahetada uuema vastu.

Prototype kasutatakse enamikes veebi raamsüsteemides. Peamiselt kasutatakse *Ajax*'i tuge ja arendamaks uusi klasse oma rakenduste jaoks. *Prototype* kasutamine on viinud veebirakendused uuele tasemele, mis on tunduvalt interaktiivsem ja kasutajasõbralikum. Antud raamsüsteemi välja arendamist on väga raske alahinnata, sest kuni selle ajani oli javascripti kood suhteliselt ekspertide kasutada ning ise midagi välja arendada oli suhteliselt keeruline ja aega nõudev.

Ruby on Rails on kõvasti laineid löönud oma raamsüsteemiga. Sellest on võtnud malli väga mitmed erinevad teised raamsüsteemid. Põhiline rõhk on pandud templiitide ja andmebaasi rakenduste loomisele. Ise ma andmebaasile tähelepanu ei pööranud seega see osa jäi välja. Viimast välja jättes ei saagi ma aru selle raamsüsteemi eripära või fenomeni. Samuti on dokumentatsioon üsna segane.

Zope arenduskeskkonna olemasolu võib pidada antud raamsüsteemi tugevuseks kui ka miinuseks sest arendus ilma selleta ei ole ette nähtud. Lisaks vajab kasutamine eelnevat õppimist. Materjali on selle jaoks on rohkem kui vaja ja kasutada oskaks.
Lisaks pean probleemiks ise seda, et arenduskeskkonna enda areng on piiratud. Paljud süsteemid juba toetavad osaliselt neid võimalusi ja isegi parema kasutusmugavusega. Loomulikult taolist veebile ja selle arendusele orienteeritud analoogset süsteemi ei ole.

11. Prototüüp veebi raamsüsteemi loomine

Põhjendus uue süsteemi loomiseks on mitmeid. Peamine on õppida ise sellest ja teiseks luua eelkõige lihtne ja vähese ressurssi nõudlikkusega raamsüsteemi, mis töötaks hästi nii arendaja kui kasutaja käe läbi.

Siiani oli kasutajaliides tehniline kitsendus rakenduste loomisel. JavaScript juhib rakenduse võimalusi kasutaja poolel. Kui võtta loogiliselt, siis ei ole võimalik UI (ingl. k. *User Interface*) võimalusi viia kaugemale, kui JavaScripti võimalused. See aga seab rakendusele piirangud. Praegusel hetkel oli piiranguks puuduv Javascripti raamsüsteem, mis hõlbustaks arendustöid. Selle tarbeks on loodud juba Prototype ja Scriptaculous, mis võimaldavad kasutada suurema osa Javascripti funktsionaalsustest.
Ülevaade kasutatavatest tehnoloogiatest ja neist sobivama valiku põhjendus

Ajax, mis kasutab *XmlXttpRequest* on ja saab kõigi raamsüsteemide osaks, sest selle kasutamine on üsna lihtne aga sellest tulenev efektiivsus ja kasulikkus on veel suuremad.

Antud raamsüsteemi loomisel lähtun mitmest erinevast raamsüsteemist ja tehnoloogilistest vahenditest:

- Prado sündmusest juhitud programmeerimist;
- Prado failistruktuur ja kasutus;
- Prado loogiline kood ja kasutus;
- Prototype ja Scriptaculoust kasutamine;
- Vähene seadistamine;
- Lihtne failistruktuur;
- Lihtne kasutamine ja loogika;
- Aktiivsete andmebaasi objektide kasutamine (ActiveRecords);
- Ajax kasutamine;

11.1. Eesmärk

Prototüübi loomisel lähtume eelnevatest süsteemidest ja arhitektuuridest.
Oma raamistiku loomisel lähtun järgnevatest põhimõtetest:

- Hoian süsteemi võimalikult lihtsana;
- Väldin üleliigsete kihtide lisamist, et vältida keerukust;

- Kasutan juba teiste poolt välja arendatud pool-rakendusi ning loon rakendust, et oleks võimalik teisi rakendusi siduda lihtsasti loodavaga;
- Standardiseerin veebi rakenduse arendust;
- Kasutan Javascripti, et siduda kasutaja liidese ja serveri tööd omavahel;

Raamistikud peaksid sisaldama kõike seda mida arenduseks on vaja. See tähendab tegelikult terve pakett programme ja uuesti kasutatavat koodi.

Mida sisaldavad endas raamsüsteemid? Raamsüsteemid peaksid sisaldama täpselt neid etteantud vajadusi mida talt nõutakse ja seda lihtsustatud kujul. Samuti peab tagama raamsüsteem koodi taaskasutuse.

Raamsüsteemid on tihedalt seotud erinevate lähenemisviisidega, mis võivad koosneda järgmistest asjadest:

- **Mustrid (Patterns**) – korduvad lahendused arenduse probleemidele mingis kindlas kontekstis;
- **Klasside nimekiri (Class libraries)** – otseselt rakendusega seotud klassid ja kaudsed või poolvalmis klassid;
- **Komponendid (Components)** – iseseisvad instantsid ja abstraktsed andmetüübid, mida saab kokku panna, et moodustada terve rakendus.

Lähtun oma süsteemi arenduses leheküljest (ingl. k. *Page*), mis on interaktiivne liides kasutaja ja serveri vahel. Läbi tema saab kasutaja kätte just need vajalikud sündmused mida tal vaja on.

11.2. Kasutatavad mustrid

Iga raamsüsteem kasutab mingeid iseloomulike arhitektuurilisi mustreid. Kirjeldan ja põhjendan ära mõned mustrid mida kasutan oma raamsüsteemis.

- **Laisk väljakutsumine (Lazy Initialization) –** annab võimaluse hoida kokku ressursside kulutamist. Asju ei kutsuta ennem välja kui vaja. Kõik moodulid või komponendid on võimalik läbi selle välja kutsuda.
- **Kaardistamise muster (The Mapper Pattern) –** seosed erinevate andmebaaside vahel. Hõlbustab andmebaasi objektide kasutamist omavahel määrates seoseid.
- **Aktiivne salvestus muster (Active Record Pattern) –** andmebaasidega suhtlemise hõlbustamiseks. Annab võimaluse teostada objektiga andmebaasi päringuid.

- **Üksik (Singleton) –** arvestades, et rakendusel peab olema üks tuumik, siis rakenduse välja kutsumisel rohkem rakendusi välja kutsuda ei saa ning ülejäänud rakenduse osad toimivad läbi ühe ja sama objekti.
- **Templiidid (Templates) –** staatilise HTML ja dünaamilise PHP lehe sidumine.
- **Nimeruumi kasutamine (Namespace)–** selle jaoks võtsin kasutusele Prado's juba tuttava „using" funktsiooni. See võimaldas lihtsalt määrata kataloogi asukoha ja selle kasutuse.
    - Raamsüsteemi klassid sisaldavad prefiksit Ctrl, et eristuda teistest rakenduse osadest.
    - Kataloogi puu on üles ehitatud loogilises järjestuses, et aru saada n.ö. rakenduse toimimise viis (näidatud Joonisel 3).

11.3. Struktuur

Raamsüsteemi ülesehituse juures lähtun kuulsast Hollywoodi printsiibist: „Ära helista meile, me helistame ise teile". Ehk siis raamsüsteemi loomine käib vastavalt vajadusele. Ennetavaid samme ei tee, vaid pigem teen raamsüsteemi valmis täpselt nii palju, et asi toimiks. Lisaks on iga klass jaoks sama nimega fail, et oleks võimalik aru saada, mis failis mis on.

Põhilisemad faili laiendid:

- **.php** – Klassidest ja funktsioonidest koosnevad failid;
- **.phtml** - PHP + HTML sisaldav fail;
- **.css** - Stiili ja kujundus fail;
- **.js** - .JavaScripti fail;
- **.sql** – Andmebaasi struktuuri ja andmete päringu fail;

Failide, klasside ja teiste taoliste ressursside kasutamiseks on olemas funktsioon mille vahendusel määratleda kasutusruum. Funktsioon on „kasutades"(ingl. k. *using*).
Vastava objekti välja kutsumine toimub laisalt ehk alles siis kui vaja. Selle jaoks otsitakse klassi fail üles ja siis kutsutakse objekt välja ning tagastatakse.

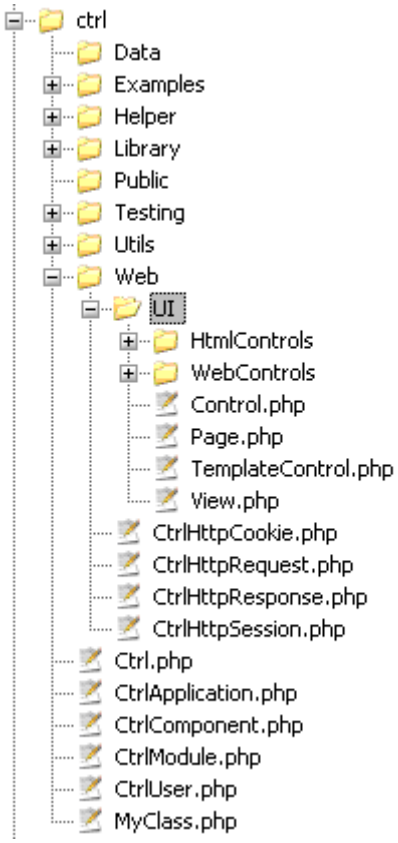
Näide: kasutada kogu raamsüsteemi kataloogis asuvaid faile. See ei anna aga ligipääsu teistesse kataloogidesse, et oleks võimalik saada ülevaade kõikides kasutuses olevaid katalooge.

```
<?php using('System')?>
```

Näide: kui on soov saada kasutada oma projekti templiit lehekülgi, siis tuleb sisestada kogu tee veebiserveris defineeritud avalikust kataloogist tavaliselt „htdocs" kataloog (joonis 5 põhjal).

```
<?php using('projects.sandbox.pages')?>
```

Raamsüsteemi ctrl failistruktuur on lihtne ja loogiline ning ei võta enda alla meeletut failide hulka joonis 3.



Joonis 3 Ctrl raamsüsteemi failistruktuur

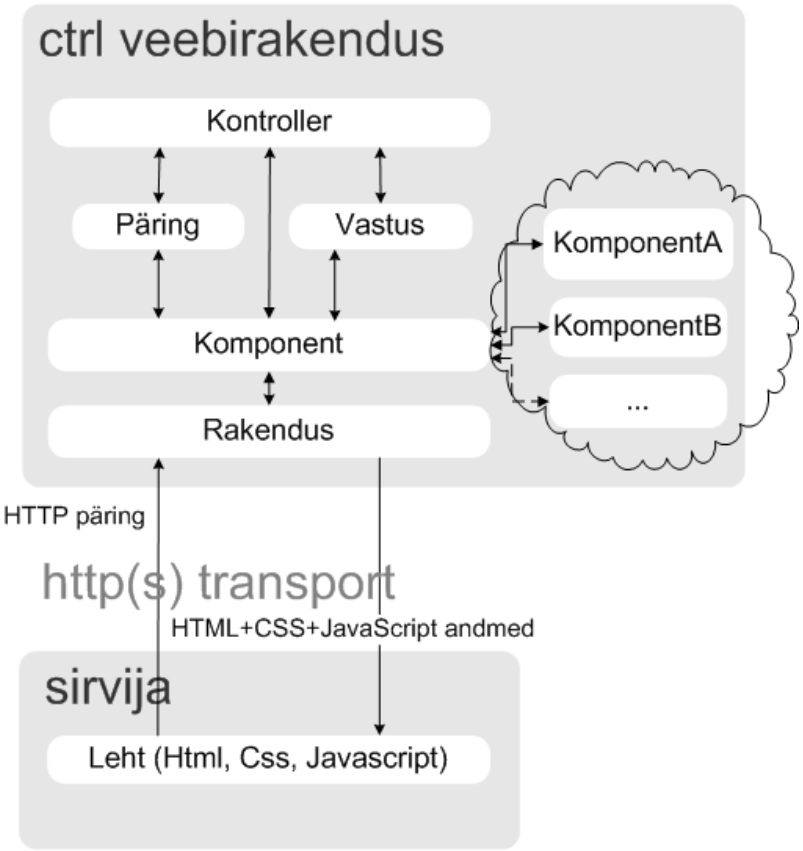
11.4. Raamsüsteem (Framework)

Raamsüsteemi ülesehitus järgneb järgnevate punktide näol. Iga peatükk iseloomustab n.ö. ühte kataloogi kus midagi asub.

- **Ctrl** – Ctrl klassiga saab kutsuda välja rakenduse. Koodi näidis on Lisa 10.
- **CtrlApplication** – rakenduse klass, et käivitada rakenduse andmete töötlemine ja kuvamine.

- **CtrlComponent** – kuna rakendus koosneb komponentidest, siis kõik rakendusega seotud klassid on seotud komponendiga, mis jagab n.ö. ressursse.

Läbi komponent klass laiendades on võimalik saada ligipääsu teistesse komponentidesse ja neid kasutada ja vastupidi. See annab väga hea paindlikkuse rakenduste loomisel. Ei teki sulu seisu mingite klasside vahel. Rakendus ise jälgib, et rakendus oleks unikaalne ning vaatab, et iga komponent mida vaja ka leitakse üles ning edastatakse rakendusele. Antud toimimisviis on näidatud joonisel 4.



Joonis 4 raamsüsteem koosneb komponentidest, mis saavad omavahel suhelda

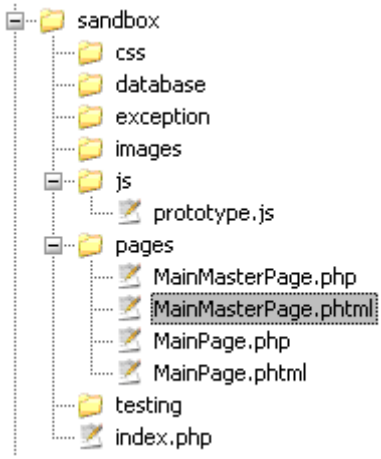
Objektide kätte saamiseks on vaja tagastus (ingl. k. getter) ja seadistus (ingl. k. setter) defineerida. Objekt on võimalik kätte saada raamsüsteemi raames või klassi raames.

11.5. Rakendus (Application)

Failisüsteemi ülesehituse tähtsus? Kui vaadelda üleval toodud veebi raamsüsteemide failistruktuure, siis leiame, et need erinevad omavahel märgatavalt. Kui ühtedel on failistruktuur ehitatud raamistiku sisse, siis teistel on rakendus eraldatud. Ise eelistan, et

rakendus on eraldatud raamistikust, et anda parem ülevaade ning vajadusel oleks võimalik uuendada raamistiku ilma, et tuleks vahetada välja kogu failistruktuur.

Luues uue rakenduse peaks looma kataloogi vastava nimega nt. „sandbox". Sellele järgneb kataloogi ja failistruktuuri loomine. Seega rakenduse failistruktuur peaks olema lihtne ja asuma kättesaadavas kohas (pilt failistruktuurist Joonis 5.):



Joonis 5 Projekti failistruktuur

- **pages** – leheküljed mida kuvatakse kasutajale; Lehed koosnevad kontrollerist ja templiit failist (nt. *MainPage.php* ja *MainPage.phtml*);
- **database** – andmebaasi objektid või mudelid suhtlemiseks;
- **exception** - veateadete erandid kuvamiseks;
- **testing** – testid antud rakenduse kontrollimiseks;
- **css** – stiililehtede asukoha kataloog;
- **images** – kõik pildid ja ikoonid asuvad antud kataloogis;
- **upload** – kasutaja poolt üles laetud failid;
- **js** – vajalikud javascripti failid asuvad selles kataloogis k.a. prototype ja scriptaculouse failid;
- **index.php** – fail, kus sees kutsutakse välja raamistik;
- **.htaccess** – fail, mis keelab osade kataloogide kuvamise nt. (pages, database, exception, tests). Samuti kirjutab vajadusel URL üle;

Näide: Kuidas luua veebi lehe klassi ja pöördumist objektide poole.

MainPage.php faili sisu:

```
<?php
using("System.Library");

class MainPage extends Page
{
        public function onInit()
        {
                $this->MasterPage = "MainMasterPage";
                $this->set("teade","Hello World");
        }
        public function onLoad()
        {
        }
}
?>
```

MainPage.phtml faili sisu:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/2002/REC-xhtml1-20020801/DTD/xhtml1-
transitional.dtd">
<html>
<head>
<script type="text/javascript" src="/js/prototype.js"></script>
</head>
<body>
 <?=$teade?>
</body>
</html>
```

11.6. Abiline (Helper)

Abiline sisaldab funktsioone, mis on vajalikud toetamaks mingite klasside kasutust. Siia kogutakse kõik sellised funktsioonid, mis on edukad toimima iseseisvalt klassidest sõltumatuna, kuid annavad abistava käe korduvate tegevuste tegemiseks.

- **HtmlHelper –** antud failis võivad olla Html koodi kirjutamiseks vajaminevad funktsioonid nagu icon_tag,

11.7. Andmed (Data)

See on osa raamistikust mis sisaldab andmebaasi ühendus või andmetega manipuleerimise funktsionaalsust. Andmebaasidega suhtlemiseks on loodud iseseisvad raamistikud, mis lihtsustavad suhtlust erinevate andmebaasidega.

Näiteks: ADOdb, PEAR DB, PhpLib, ActiveRecords jne.

Antud juhul on ADOdb üks kiiremaid ja paremini dokumenteeritud raamsüsteeme mitme erineva andmebaasi toega (vt. http://phplens.com/lens/adodb/).

11.8. Erandid (Exception)

Igal OO rakendusel on võimalik kinni püüda n.ö. erandid (ingl. k. *exception*), mis kajastavad mingi olukorra tekkimist. Olukorrad võivad olla erinevad alates sellest, et mingi objekti sisend on vigane või puuduvad vajalikud parameetrid. Erandite kinni püüdmine annab võimaluse grupeerida erinevaid juhtumeid kokku ning muuta nende väljundi kujundust.

Erandeid kasutatakse mitmel moel. Osad kasutavad erandeid ainult vigade teavitamiseks ja osad iga erandi tekkimisel.

- **CtrlException** – viga Ctrl raamsüsteemi endaga;
- **NoPageFoundException** – viga kui ei leita kontrollerit või templiit faili;
- **UnknownException** - teadmata viga;

11.9. Vigade haldamine (Error handling)

Vigade haldamine on raamsüsteemi tähtis osa. Vigu võib olla mitmeid, sõltuvalt rakendusest ja keelest jne. Põhilisem mida kasutatakse on vead mida arendaja saab määrata ise oma rakenduse seest välja kutsudes. Neid nimetatakse kasutaja veaks (ingl. k. *user error*). Teised vead on seotud rohkem keele süntaksi või fataalsete vigade tekkimisega.

Vead võivad olla peidetud, kuvatud kasutajale või salvestatud erinevatesse vormidesse nagu andmebaas, fail või emaili peale saadetuna.

11.10. Teek (Library)

Teek on koht kus hoitakse täiesti iseseisvaid klasside kogumeid. Need on üks osa toetavatest klassidest või alamprogrammidest mida kasutatakse sageli.

- **Calendar –** Kalendri kasutamise rakendus;
- **Browser –** sirvija andmete haldur;

11.11. Html kontrollerid (HtmlControls)

Html kontrollerid on grupp klasse, mis võimaldavad luua HTML standardseid objekte ja nendega manipuleerida. Abistamiseks kasutavad need klassid Helpereid.

- **Table –** tabeli objekti loomise klass
- **TableCell –** tabeli lahtri haldamise klass
- **TableRow –** tabeli rea haldamise klass
- **TableHeader –** tabeli päise lahtri halduse klass

11.12. Internatsionaliseerimine (I18n) ja Lokaliseerimine (L10n)

Igal rakendusel peaks olema võimalus, et oma veebileht viia igale kasutajale, kes seda lehte vaatab. Selle jaoks on vaja aga eelnevalt kasutaja lokaliseerimine, kust ta pärit on ja mis keelt ta valdab. Selliste olukordade lahendamiseks on lokaliseerimine ja internatsionaliseerimine.

Eelnevalt tehakse kindlaks kasutaja ja tema omadused ja seejärel viiakse läbi veebilehe internatsionaliseerimine ehk siis tõlgitakse veebilehte või viiakse õigesse ajatsooni.

- **Lokaliseerimise eesmärk:**
    - Asukoht;
    - Keele tõlget;
    - Erinevate keelte murded;
    - Kindel toetus mingile kindlale keelele nagu Eesti keel;
    - Kohalikud kombed;
    - Kohalikud omapärad;
    - Sümbolid;
    - Sorteerimise järjestus;
    - Esteetika;
    - Kultuuriline väärtus ja sotsiaalne kontekst;
- **Internatsionaliseerimise eesmärk:**
    - Keel;
    - Kuupäeva formaat ja erinevate kalendrite kasutamine;
    - Numbrite formaat;

- Aja tsoonid;
- Valuuta;
- Pildid ja värvid;
- Lipud ja embleemid;
- Nimed ja pealkirjad;
- Valitsuse poolt määratud identifikaatorid või numbrid nagu (ID kaart, Passi number, Sotsiaalkindlustuse number USA-s jne.);
- Telefoni numbrid, aadressid, postiindeksid;
- Mõõtühikud ja kaaluühikud;
- Paberi suurused;

Erinevus internatsionaliseerimisel ja lokaliseerimisel on marginaalne kuid siiski oluline. Internatsionaliseerimise puhul mõistetakse mingi eseme või objekti kasutamist virtuaalselt ükskõik mis kontekstis ehk siis on võimalik panna ta ükskõik mis keskkonda.

Lokaliseerimise all mõistetakse omaduste kogumit, mis on kasutusele võetud mingis spetsiifilises asukohas ehk kasutaja identifitseerimine. Et süsteem toimiks ühtsena tasuks kasutada neid mõlemaid lähenemisi üheaegselt, et saada parim globaliseeritavus.

Kasutaja lokaliseerimine on üsna keeruline. See vajab kasutaja sirvija parameetrite lugemist ja teiseks parameetriks peaks olema **IP põhine lokaliseerimine** (ingl. k. *IP locator*), millega oleks võimalik positsioneerida üsna täpselt kasutaja asukohta maailmas.

11.13. Kasutajaliides (UI)

Me ei kujutaks ette ühtegi veebilehte ilma kasutaja liideseta. Selle tarvis on vaja kindlasti kasutajaliidese tarvis minevat klasside gruppi. Taoline grupp annab võimaluse luua veebi lehe põhiosa. Veebilehte ennast (Inglise k. Page)

Kasutades selle jaoks Model->Controller->View ülesehituse mustrit leiame antud grupist taolised klassid.

- **Model –** andmemudeli kontroller;
- **Control –** kontroller tempiit ja lehe jaoks
- **Page –** lehekülje kontroller, mis on kontrolleri laiendus;
- **View –** vaate kontroller;

11.14. Kujundus (Themes)

Kujundust on võimalik muuta HTML ja CSS kasutades. Kujundus toimib lehekülgede ja vaadete disaini muutmise raames. Kuigi osadel raamsüsteemidel on see eraldi jaotatud. Antud kujundus sisaldab endas tegelikult lehekülje kujundust ja struktuuri. Seda sama võib hoida ka „page" kataloogis kus on kõik muud kujundusega seotud failid.

11.15. Vaated (Views)

Vaadete all mõistetakse rakenduse osasid, mis on võimelised oma väljundit transformeerima ükskõik millisesse teise formaati.
Antud klass peaks võimaldama luua andmetest erinevaid väljundeid nagu: **pdf, html, xml, xls, doc, rss, txt, cvs.**

11.16. Veeb (Web)

Antud grupist peaks olema võimalik leida kõiki selliseid objekte mis on veebi põhiosadeks ning milleta veebi rakenduse loomine oleks raskendatud, ebaloogiline või raskendatud. Antud grupi olemasolu on väga tähtis hästi toimiva rakenduse puhul.

Järgnevalt toon välja objektid, mis on loogilise veebi osad:

- **Päring (Request)** – päring, mis on tehtud antud rakendusele;
- **Kasutaja (User)** – kasutaja, kes on teinud päringu antud rakendusele;
- **Sessioon (Session)** – sessiooni loomine ja haldamine;
- **Küpsis (Cookie)** – küpsise loomine ja haldamine;
- **Vastus (Response)** – vastus või tagasiside kasutajale peale rakenduse tegevust ja väljundi kinni püüdmist, et see kasutajale saata;

Kokkuvõte

Olulisemad tulemused ja järeldused on, et saadi valmis üsna kompaktne raamsüsteemi mida on võimalik kasutada igal ühel ning seda edasi arendada. Taoliste veebiraamsüsteemide kasutamine on tunduvalt lihtsam kuna ei eelda väga põhjaliku raamsüsteemi enda tundmist, vaid pigem loogilist mõtlemist või siis sarnasuste leidmine näiteks Javascriptiga.

Kuna töö üheks eesmärgiks oli luua hea ja lihtsa ülesehitusega süsteem, siis selles osa sai eesmärk täidetud. Javascripti raamsüsteem, mis toimib ainult kliendi poolel. Prototype vajab arendamist, et algkäivitus oleks kiire. Võiks sisaldada samuti laisa kutsumise meetodit.

Raamistike loomisel on tähtis kui hästi on selle kood kirjutatud. Selle tagamiseks on standardid või ettekirjutused. Piisaks ka sellest, kui kood oleks igal poole ühes stiilis kirjutatud. See teeb lihtsamaks tulevikus parandusi teha ja ülevaadet saavutada, mis moodi miski asi toimib.

Sain aru, et nimeruumi kasutamine ja mõistmine rakenduse ülesehitamisel on tähtis. See annab võimaluse luua tunduvalt paremini erinevatesse operatsioonisüsteemidesse rakendusi. Töö täitis oma eesmärki andes ülevaate raamsüsteemidest ja teemast üldiselt ja võimaluse luua antud raamsüsteemi.

Nagu enamike töödega, siis ka see töö ei saanud täies ulatuses valmis ja eesmärgid olid tunduvalt suuremad. Samas näitab see seda, et teema on tunduvalt laiem ja mahukam kui see eelnevalt paistis. Ehk siis üks lihtne veebileht või endas peita ja sisaldada hoopis suuremat ja keerulisemat süsteemi kui lihtsalt teksti ja pildi kuvamist.

Ma arvan, et selles osas sai töö eesmärk täidetud, et ülevaade on saadud ning antud töö lugejad said tunduvalt parema ülevaate kui neil ennem oli ning ka ideid kuidas midagi mingis olukorras lahendada.

Üheks minu poolseks lootuseks oleks olnud luua mingi ühtne standard. Oma töös ma ei jõudnud selgelt standardite välja toomisega. Üheks võimaluseks oleks antud arendustöö standardiseerida. Teiseks lootuseks oleks luua ühtne koodibaas, mida kõik antud valla inimesed saaksid täiustada ning kasutada. See vähendaks tunduvalt tühja töö tegemist ja tõstaks arenduse kvaliteeti. Võimalik oleks otsida arendusgrupp või luua ise kommuun kellega luua arenduskeskkond ja hästi argumenteeritud raamsüsteem. See kiirendaks ja teeks raamsüsteemi tunduvalt paremaks, kui ühe inimese arendus.

Ettepanekud edasiarenduseks või uurimustöödeks antud valdkondades.

Hetke suund on veebile orienteeritud raamistikel kasutajaliidesele ja selle parem ära kasutamine.

Iga alateema võimaldab teha pikemat ja põhjalikumat analüüsi. Kindlasti on võimalik tuua alternatiivne antud raamsüsteemile.

Tasuks uurida mis tehnoloogiaid veel on olemas ja mida saaks kasutada nii kasutajaliideses kui ka serveri poolel näiteks MSSQL andmebaasi kasutuses olevad käivitajad (ingl. k. *trigger's*)

Suuremad arendused liiguvad oma rakendustega veebi, et olla kõigile kättesaadav ja globaalne. Kui 1990ndatel arendati operatsiooni süsteeme arvutitele, siis nüüdseks loome me n.ö. „OS" internetis iseseisvalt toimima. Seda küll kasutajaliidese vormis, mis jookseb kellegi sirvijas.

Oleks vaja täpsemat vajaduste kaardistamine, mida ootavad arendajad ning mida soovivad tavakasutajad veebidelt ning kuidas neid oleks võimalik ühtlustada

Kui keegi suudab antud raamsüsteemi struktuuri ja ülesehitust veel rohkem lihtsustada ja panna toimima rakendus veel vähesemate objektidega sama sujuvalt, siis kindlasti tasub sellest teistele märku anda. Lisaks tasub kindlasti tuua antud töö kohta parandusi ja ettepanekuid, sest iga normaalne süsteem peab olema pidevas arengus.

Web frameworks: structure and prototype creation

Web pages have taken a big step during the last few years. Starting from static pages they have evolved into more dynamic and user friendly solutions. To support all that a complicated web framework is needed. Framework's goal is to make the development of web pages more easy on the fly. Should the frameworks be only easy to use? Right now there is a proliferation of frameworks. You can find a number of web frameworks in different languages or platforms with different build-ups that are used to build the applications.

In fact, when there is a proliferation of particular "things" it means that none of them addresses enough needs and it's time to see what the needs are and address them. This was true in the mid 1990's when there were about 30 different 32-bit operating systems available for the desktop. Today, we have Windows, Mac OS X, and Linux. Back then, there was a proliferation. A few survived [8].

It means that we have to study what are the exact needs for the developer and for the user. How should the frameworks work and how they should be built? What are the needs for a web page and what the user can do with it. All these concerns and many others will be discussed in the work to give an overview of every aspect in the matter.
Every new technology will lead to further development of web frameworks to build required web applications. For example the usage of Ajax that is used all most every framework is the cornerstone for these solutions.

For better understanding of the thesis I will give an overview of different notions and acronym's like JSON, SOA, ORM, YAML etc. I will make overview of the architectural patterns and their usage. To communicate with different data sources i will present a write-up about remote services.

As every application encounters security problems I will also make a brief overview how to avoid XSS or XSRF attacks.

For me it is very important the quality of the code and the way it is written. For that reason I will focus on the optimization and standardization of programming code.
Furthermore I will give an overview of other web frameworks like Prado, Code Igniter, Ruby on Rails and Zope. I have also included the ADOdb framework to explain the database supporting library. Ajax and other client side featured are explained using Prototype and Scriptaculouse.

For conclusion I have made a light, easy to use and extensible prototype web framework where I have used all the knowledge together to give an overview of the aspects discussed in this thesis.

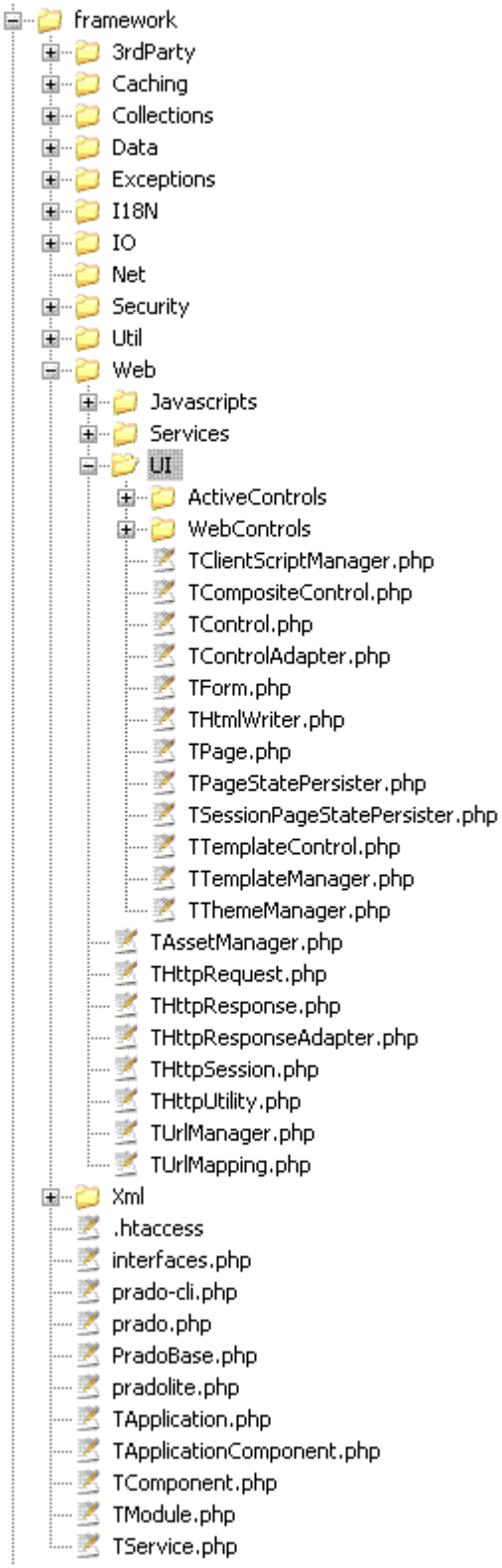
Kasutatud kirjandus

1. Clifton M. (2003, 4. november). *What Is A Framework.* [5. aprill 2007], *http://www.codeproject.com/gen/design/WhatIsAFramework.asp*
2. CodeIgniter (2001-2007). *Open Source PHP Web Application Framework.* [5. aprill 2007], http://www.codeigniter.com/
3. Fayad M., Schmidt D. C. (1997). *Object-Oriented Application Frameworks*. [5. aprill 2007], http://www.cs.wustl.edu/~schmidt/CACM-frameworks.html
4. Microsoft Live Labs (k.p). *Microsoft Live Labs: Photosynth*. *Veebirakendus fotode kuvamiseks ruumiliselt* [5.aprill 2007], http://labs.live.com/photosynth/.
5. Mägi M. (2005). *Bakalurusetöö: Avatud lähtekoodiga tarkvara arendus*. [18. aprill 2007], http://www.cs.tlu.ee/osakond/opilaste_tood/seminari_ja_proseminari_tood/2005_sugis/Marek_Magi/Marek_Magi_Seminari_Too.pdf
6. Normak M. (2001). *Strutsi raamistik ja selle kasutamine veebirakenduste teostamisel.* [18. aprill 2007], http://www.cs.tlu.ee/osakond/opilaste_tood/seminari_ja_proseminari_tood/2001-2002/Mikk_Normak/Mikk_Normak_Proseminari_Too.pdf
7. Nurme A. (2006). *Seminaritöö:PHP-põhise tarkvaraarenduse abivahendid.* [18. aprill 2007], http://www.cs.tlu.ee/osakond/opilaste_tood/seminari_ja_proseminari_tood/2006_kevad/Ahti_Nurme/Ahti_Nurme_Seminari_Too.pdf
8. Pollak D. (7. dets. 2006). *Why the world needs another web framework.* [5. aprill 2007], http://blog.lostlake.org/index.php?/archives/25-Why-the-world-needs-another-web-framework.html
9. Pollak D. (21. nov. 2006). *Web Framework Manifesto.* [19. aprill 2007], http://blog.lostlake.org/index.php?/archives/16-Web-Framework-Manifesto.html
10. Poolak M. (2006). *Seminaritöö:Programmi ülesehituse mustrid.* [18. aprill 2007], http://www.cs.tlu.ee/osakond/opilaste_tood/seminari_ja_proseminari_tood/2006_kevad/Margo_Poolak/Margo_Poolak_Seminari_Too.pdf
11. Prado Software (2006, 2007). *PRADO PHP Framework.* [5. aprill 2007], http://www.pradosoft.com/
12. Ruby on Rails (k.p) *Open-source web framework.* [5. aprill 2007], http://www.rubyonrails.org/
13. Schlossnagle, G. (2004). *Advanced PHP Programming: A practical guide to developing large-scale Web sites and applications with PHP5.*
14. Vallaste H. (2007). *e-Teatmik: IT ja sidetehnika seletav sõnaraamat. Inglise keelsete terminite tõlked eesti keelde.* [17. aprill 2007], http://www.vallaste.ee/
15. Vilgot A. (2004). *Bakalaurusetöö: Mudelipõhine tarkvaraarendus.* [18. aprill 2007], http://www.cs.tlu.ee/osakond/opilaste_tood/magistri_tood/2004_sugis/Anu_Kurm/Anu_Kurm_Mag_Too.pdf)
16. ADOdb (2000-2004). *Andmebaasi abstraktsiooni teek.* [23. aprill 2007], http://adodb.sourceforge.net/

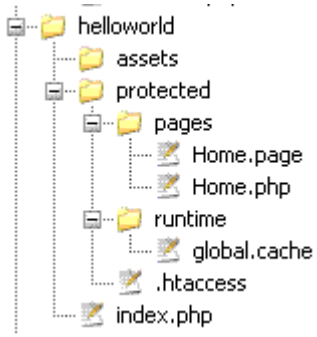
17. Zope (2006). *Open-source application server for building content managment systems*. [5. aprill 2007], http://www.zope.org
18. Garrett J.J. (8. veebruar 2005). *Ajax: A New Approuche to Web Applications.* [24. aprill 2007], http://www.adaptivepath.com/publications/essays/archives/000385.php
19. Winer D. (15. juuni 1999). *XML-RPC Spetsifikatsioon.* [26. aprill 2007], http://www.xmlrpc.com/spec
20. Kidd E. (12. aprill 2001). *XML-RPC kuidas kasutada*. [26. aprill 2007], http://www.faqs.org/docs/Linux-HOWTO/XML-RPC-HOWTO.html#xmlrpc-howto-php-client
21. Schmidt D.C. (2006). *Ülevaade CORBA'st.* [26. aprill 2007], http://www.cs.wustl.edu/~schmidt/corba-overview.html
22. MageLang Institute (1998-1999). *Introduction to CORBA*. [22. aprill 2007], http://java.sun.com/developer/onlineTraining/corba/
23. Carnegie Mellon University (2007). *Component Object Model (COM), DCOM, and Related Capabilities*. [21. aprill 2007], http://www.sei.cmu.edu/str/descriptions/com.html
24. Sun Microsystems (2007). *Remote Method Invication (RMI).* [21. aprill 2007], http://java.sun.com/developer/onlineTraining/rmi/RMI.html#UsingRMI
25. Booth D. (2004-2007). *Web Service Architecture.* [20. aprill 2007], http://www.w3.org/TR/ws-arch/#introduction, 1.4 „What is a Web service?"
26. Refsnes Data (1999-2007). *WSDL kasutusõpetus.* [19. aprill 2007], http://www.w3schools.com/wsdl/wsdl_uddi.asp
27. why (8. aprill 2005). *YAML is JSON*. [27. aprill 2007], http://redhanded.hobix.com/inspect/yamlIsJson.html
28. Clifton M. (22. juuli 2003). *Application Automation Layer* [27. aprill 2007], http://www.blogstudio.com/aal/
29. Bergen P. (k.p). *MVC programmeerimise mustri tutvustus* [27. aprill 2007] http://www.dossier-andreas.net/software_architecture/mvc.html
30. Buschmann F., Meunier R, Rohnert H., Sommerlad P., Stal M. (20. oktoober 2000). *Presentation-Abstraction-Control* [27. aprill 2007], http://www.vico.org/pages/PatronsDisseny/Pattern%20Presentation%20Abstra/
31. Gregory A. (22. oktoober 2005). *Event-Driven Programming* [27. aprill 2007] , http://www.scss.com.au/family/andrew/pdas/psion/toolbox/tutorial/eventdp/
32. Gauld A. (21. jaanuar 2007). Event Driven Programming [27. aprill 2007], http://www.freenetpages.co.uk/hp/alan.gauld/tutevent.htm
33. Ferg S. (2. august 2006). Event-Driven Programming: Instruction, Tutorial, History [27. aprill 2007], http://eventdrivenpgm.sourceforge.net/event_driven_programming.pdf

Lisad

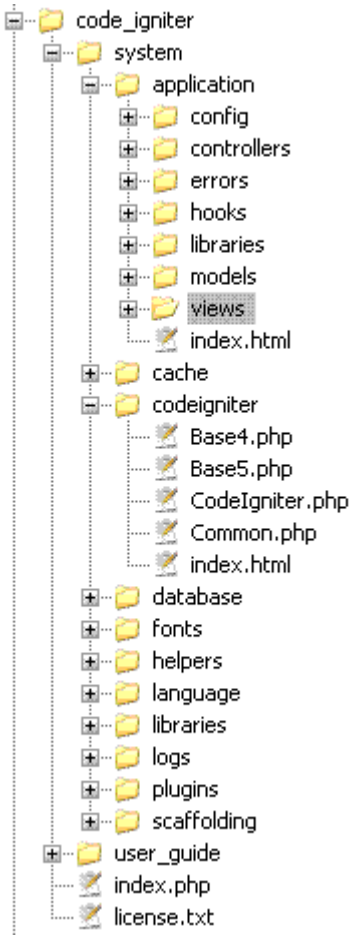
1. **Prado raamistiku failistruktuur**



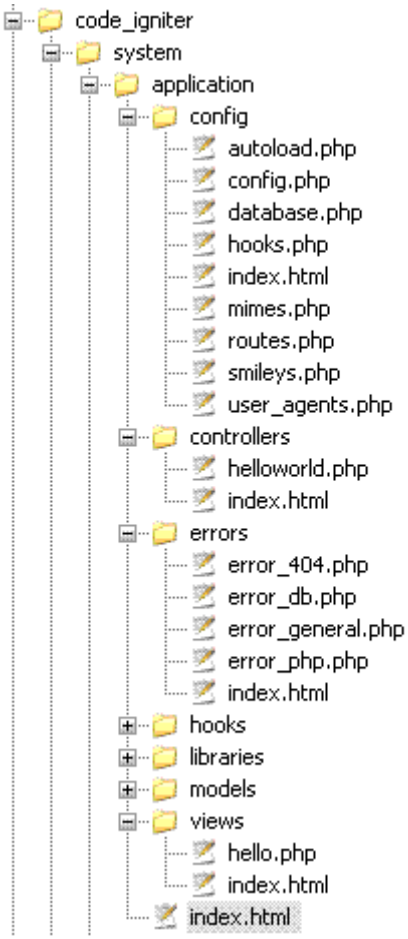
2. **Prado „helloworld” rakenduse failistruktuur**



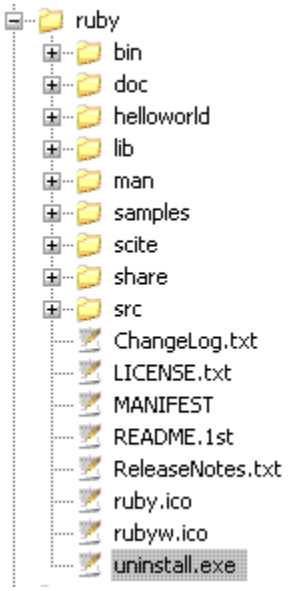
3. **Code Igniter raamsüsteeemi failistruktuur**



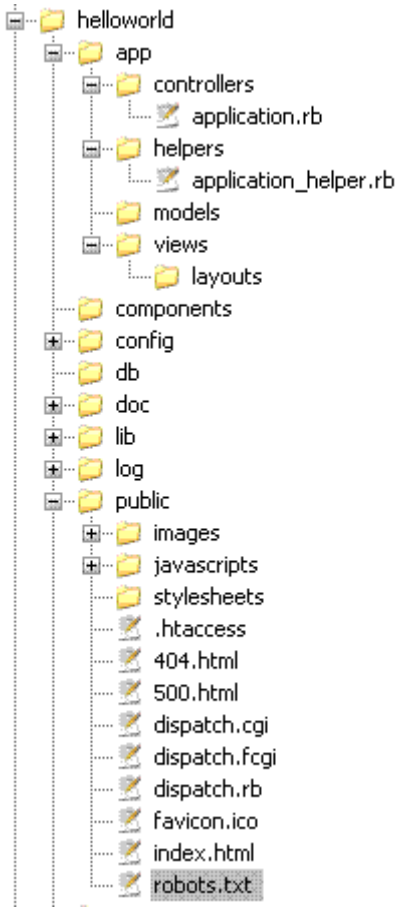
4. **Code Igniter „helloworld" rakenduse failistruktuur**



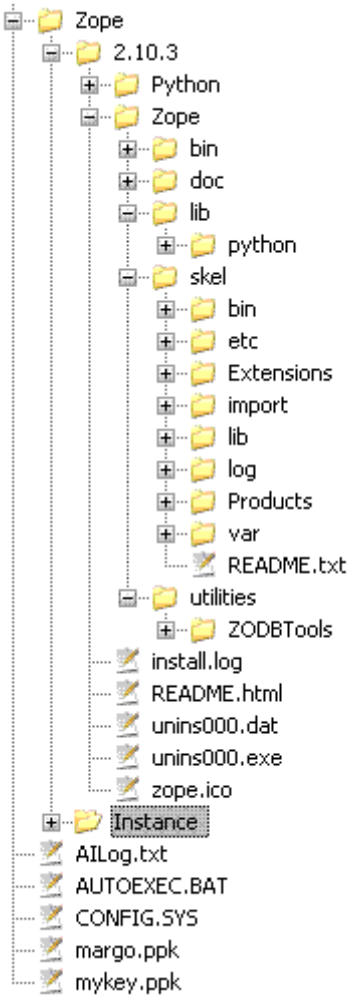
5. **Ruby on Rails rakenduse failistruktuur**



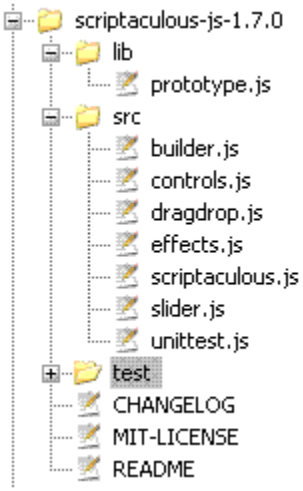
6. **Ruby on Rails „helloworld" rakenduse failistruktuur**



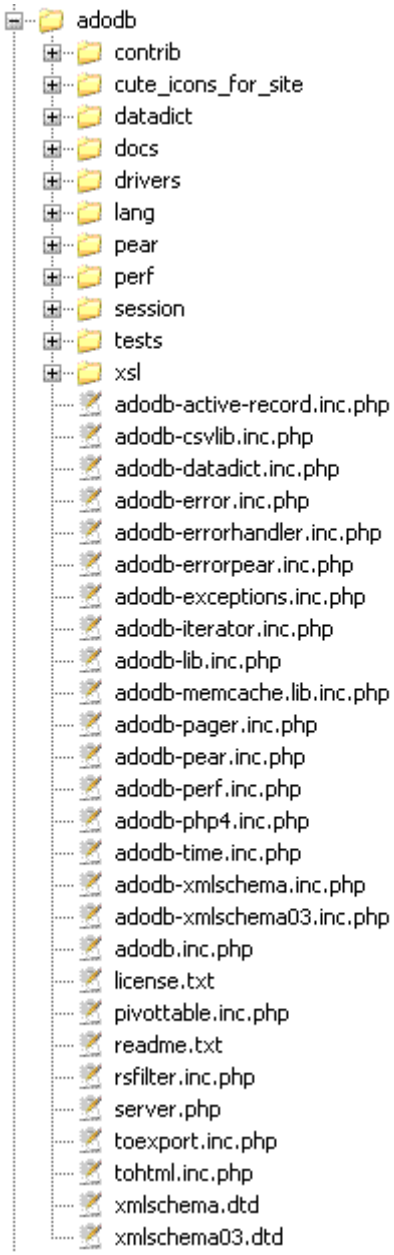
7. **Zope rakenduse failistruktuur**



8. **script.aculo.us raamsüsteemi failistruktuur**



9. **ADOdb raamsüsteemi failistruktuur**



10. ***Code Igniter „index.php" faili sisu***

```
<?php
/*
|---------------------------------------------------------
--------
| PHP ERROR REPORTING LEVEL
|---------------------------------------------------------
--------
*/
    error_reporting(E_ALL);
/*
|---------------------------------------------------------
--------
| SYSTEM FOLDER NAME
```

```
|---------------------------------------------------------
--------
*/
    $system_folder = "system";
/*
|---------------------------------------------------------
--------
| APPLICATION FOLDER NAME
|---------------------------------------------------------
--------
*/
    $application_folder = "application";
/*
|---------------------------------------------------------
--------
| SET THE SERVER PATH
|---------------------------------------------------------
--------
*/
if (function_exists('realpath') AND @realpath(dirname(__
FILE__)) !== FALSE)
{
    $system_folder = str_replace("\\", "/", realpath(dir
name(__FILE__))).'/'.$system_folder;
}
/*
|---------------------------------------------------------
--------
| DEFINE APPLICATION CONSTANTS
|---------------------------------------------------------
--------
|
| EXT        - The file extension.  Typically ".php"
| FCPATH     - The full server path to THIS file
| SELF        -
 The name of THIS file (typically "index.php)
| BASEPATH    -
 The full server path to the "system" folder
| APPPATH     -
 The full server path to the "application" folder
|
*/
define('EXT', '.'.pathinfo(__FILE__, PATHINFO_EXTENSION)
);
define('FCPATH', __FILE__);
define('SELF', pathinfo(__FILE__, PATHINFO_BASENAME));
define('BASEPATH', $system_folder.'/');

if (is_dir($application_folder))
{
    define('APPPATH', $application_folder.'/');
}
else
{
    if ($application_folder == '')
    {
        $application_folder = 'application';
    }

    define('APPPATH', BASEPATH.$application_folder.'/');
}
```

```
/*
|---------------------------------------------------------
--------
| DEFINE E_STRICT
|---------------------------------------------------------
--------
*/
if ( ! defined('E_STRICT'))
{
    define('E_STRICT', 2048);
}
/*
|---------------------------------------------------------
--------
| LOAD THE FRONT CONTROLLER
|---------------------------------------------------------
--------
*/
require_once BASEPATH.'codeigniter/CodeIgniter'.EXT;
?>
```

11. Koodi dokumenteerimise näide

```
<?php
/**
 * MyClass file
 *
 * <code>
 * using('System.MyClass');
 * </code>
 * @name MyClass
 * @package MyClass
 * @date 22.03.2007
 * @author Margo Poolak <margo 'dot' poolak 'at' mail 'd
ot' ee>
 * @internal This class is invented to do something for
me
 * @see CtrlApplication
 */
/**
* MyClass class
*
* This is my own invented class
* <code>
* <?php
* // Can use as follows
* $my_class = new MyClass();
* echo $my_class-
>Hello('John');// will output "Hello John"
* echo $my_class-
>Goodbye();// will output "Goodbye John"
* echo $my_class-
>Goodbye('Willy'); // will output "Goodbye Willy"
* ?>
* </code>
*
* @name MyClass
* @package MyClass
* @version 1.0
```

```
*/
class MyClass
{
    /**
    * @var string to save name
    */
    public $name;
    /**
    * @var string
    * @access private
    */
    private $_name;
    /**
    * @var string to save word
    * @access protected
    */
    protected $__word;

    /**
    * Say Hello to someone
    * This will say Hello to someone and will save his n
ame
    * @param string $name
    * @return string Hello $name
    */
    public function Hello($name)
    {
        $this->$_name = $name;
        return "Hello $name";
    }
    /**
    * Say Goodby to someone
    * This will say Goodby to someone \nIf the name not
input then will say to the saved name
    * @param string $name
    * @return string Goodbye $name
    */
    public function Goodbye($name='')
    {
        if(!!($name))
        {
            return "Goodbye $name";
        }
        return "Goodbye $this->_name";
    }
    /**
    * What user says
    * You can save the word to say
    * @param string word
    * @access protected
    */
    protected function Say($word)
    {
        $this->__word = $word;
    }
}
?>
```

12. **Ctrl klass**

```php
<?php
/**
* Is the Ctrl defined
*/
define('CTRL',true);
/**
* What is the ctrl version
*/
define('CTRL_VERSION','2.0');
/**
* System directory handler can use to access system directory
* <code>
* using('System.Web.UI.HtmlControls');
* </code>
*/
define('SYSTEM','System');
/**
* Class extention parameter, i think .php is good extention
*/
define('CLASS_EXT',             '.php');
/**
*Current system application location
*/
define('SYSTEM_PATH',         dirname(__FILE__));
/**
* Document root where is the other dirs and files
*/
define('DOCUMENT_ROOT',         $_SERVER['DOCUMENT_ROOT']);
/**
* Determing operating system
*/
if (substr(PHP_OS, 0, 3) == 'WIN')
{
    define('OS_WINDOWS', true);
    define('OS_UNIX',    false);
    define('OS',         'Windows');
    define('INCLUDE_PATH_SEPARATOR', ';');
}
else
{
    define('OS_WINDOWS', false);
    define('OS_UNIX',    true);
    define('OS',         'Unix'); // blatant assumption
    define('INCLUDE_PATH_SEPARATOR', ':');
}
/**
* All errors to show
*/
error_reporting(E_ALL);
/**
* New controlled error handler on kustutatud
*/
/**
* If class is called out first time then first require a file
* If u call out new class object then it will run it here
* It calls out function using that searches the class file from predefined
directories
* {@source}
*/
if(!function_exists("__autoload"))
{
```

```
    function __autoload($class_name)
    {
        import($class_name);
    }
}
/**
* Returns the include paths
* @return array of inluce_path
*/
function get_include_paths_array()
{
    $include_path = ini_get('include_path');
    return explode(INCLUDE_PATH_SEPARATOR, $include_path);
}
/**
* Translates the path system.ui.web to system/ui/web
* If the string contains System then it will be replaced with the document
root
* @param string
* @returs string of directory
*/
function translate_ctrl_path($path)
{
    $dir = str_replace('.',DIRECTORY_SEPARATOR,$path);
    /*
    * If the path consist System then replace it with document root path
    */
    if(strstr($path,SYSTEM) === false)
    {
        $dir = str_replace('/',DIRECTORY_SEPARATOR,$dir);
        $document_root = str_replace('/',DIRECTORY_SEPARATOR,DOCUMENT_ROOT)
;// change all / directory separators
        if(substr($dir,-1) == DIRECTORY_SEPARATOR)
        {
            $dir = $dir;
        }
        else
        {
            $dir = $document_root.DIRECTORY_SEPARATOR.$dir;
        }
    }
    else
    {
        $dir = str_replace(SYSTEM,SYSTEM_PATH,$dir);
    }
    return $dir;
}
/**
* Can we find file from include paths
* @paramr string file name
* @return boolean true|false
*/
function is_file_exists($file)
{
    $paths = get_include_paths_array();
    /**
    * Search for the file
    */
    foreach($paths as $path)
    {
        if(is_file($path.DIRECTORY_SEPARATOR.$file))
```

```
        {
            return true;
        }
    }
    return false;
}
/**
* Loads the class file
* @param class name
* @return if require succeeded true|false
*/
function import($className)
{
    $namespaces = get_include_paths_array();
    // if class allready exists
    if(class_exists($className,false))
    {
        return true;
    }
    $filename = $className.CLASS_EXT;
    if(is_file_exists($filename))
    {
        require_once($filename);
        return true;
    }
    return false;
}
/**
* Sets predefined direcotryes from where to require files
* Searches the class file from the predefined directories
* Can use System directory where is the frameworks source.
* Other directorys wil used DOCUMENT_ROOT
* @param string path name to include
* @return boolean true|false
* <code>
* using('Dir.Second.Third');// will use directory as /Dir/Second/Third on U
nix or C:\Dir\Second\Third in Windows
* </code>
*/
function using($name)
{
    if(!$name)
    {
        return false;
    }
    $namespaces = get_include_paths_array();
    /**
    * Allready defined namespace
    */
    if(in_array($name,$namespaces))
    {
        return true;
    }
    else
    {
        $path = translate_ctrl_path($name);
        if(is_dir($path))
        {
            ini_set('include_path',ini_get('include_path').INCLUDE_PATH_SEP
ARATOR.$path);
            return true;
```

```
        }
        elseif(is_file($path.CLASS_EXT))
        {
            require_once($path.CLASS_EXT);
            return true;
        }
        else
        {
            trigger_error('The namespace is invalid: '.$path, E_USER_ERROR)
;
        }
    }
}
/**
* Ctrl singleton
* @date 9.03.2007
* @name Ctrl
* @author Margo Poolak <margo 'dot' poolak 'at' mail 'dot' ee>
* @copyright Copyright &copy;2007 Margo Poolak. All Rights Reserved.
* @version 2.0
* @package System
* @category
* @static
* Static methods
* Usage of this Ctrl is simple
* <code>
* <?php
* require_once('../Ctrl.php');
* Ctrl::run();
* ?>
* </code>
*/
class Ctrl
{
    /**
    * @static
    * @var string application
    * @access private
    */
    private static $_application;
    /**
    * Construct must be protected
    * @access protected
    */
    protected function __construct()
    {
    }
    /**
    * Runs the application
    * @static
    */
    public static function run()
    {
        self::getApplication()->run();
    }
    /**
    * Sets application only once
    * @static
    * @param object application
    * @return null;
    */
```

```
    public static function setApplication(CtrlApplication $app)
    {
        if(isset(self::$_application))
        {
            trigger_error('The applicatin is allready set', E_USER_ERROR);
        }
        self::$_application = $app;
    }
    /**
    * If the application is not set then it will be triggered
    * @static
    * @return object the application
    */
    public static function getApplication()
    {
        if(!isset(self::$_application))
        {
            return new CtrlApplication();
        }
        else
        {
            return self::$_application;
        }
    }
    /**
    * Localize text
    * @param string translation text
    * @return string of translated text
    */
    public static function localize($text)
    {
        return $text;
    }
}
/**
* Using namespaces
*/
using('System');
using('System.Web');
using('System.Web.UI');
?>
```

13. CtrlApplication klass

```
<?php
/**
 * CtrlApplication file
 * @date 9.03.2007
 * @name CtrlApplication
 *
 * @author Margo Poolak <margo 'dot' poolak 'at' mail 'dot' ee>
 * @copyright Copyright &copy;2007 Margo Poolak. All Rights Reserved.
 * @package System
 *
 * Application file
 */
if(!defined('CTRL'))
{
    exit();
}
```

```php
/**
 * CtrlApplication class

 * @name CtrlApplication
 *
 * @version 2.0
 * @internal This class is extended to CtrlComponent to get use of all the
methods in init
 *
 * Runs the system by running module and page using the request
 */
class CtrlApplication extends CtrlComponent
{
    /**
    * @var object request handler
    * @access private
    */
    private $_request;
    /**
    * @var object response handler
    * @access private
    */
    private $_response;
    /**
    * @var object session handler
    * @access private
    */
    private $_session;
    /**
    * @var object cookie handler
    * @access private
    */
    private $_cookie;
    /**
    * @var array property handler
    * @access private
    */
    private $_propertys = array();
    /**
    * Sets application to Ctrl singleton
    */
    public function __construct()
    {
        Ctrl::setApplication($this);
        $this->getSession();
    }
    /**
    * Sets new property to application
    * Overrides the parameter set before
    * @param string property name
    * @param mixed property value
    */
    public function setProperty($name,$value)
    {
        $this->_propertys[$name] = $value;
        return true;
    }
    /**
    * Gets property if it set
    * If property is not set then trys to import class object
    * @param string variable name
```

```
    * @return mixed if failed then false else the object or value
    */
    public function getProperty($var)
    {
        if(isset($this->_propertys[$var]))
        {
            return $this->_propertys[$var];
        }
        else
        {
            /**
            * Lets try to import a class object
            */
            if(import($var) == true)
            {
                $object = new $var();
                $this->SetProperty($var,$object);
                return $object;
            }
            else
            {
                // didnt make it
                trigger_error("Unable to load $var", E_USER_ERROR);
            }
        }
        return false;
    }
    /**
    * Sets Request object
    * @param CtrlHttpRequest object
    */
    public function setRequest(CtrlHttpRequest $request)
    {
        $this->_request = $request;
        return true;
    }
    /**
    * GetsRequest object
    * @return CtrlHttpRequest object
    */
    public function getRequest()
    {
        if(!$this->_request)
        {
            $this->_request = new CtrlHttpRequest();
        }
        return $this->_request;
    }
    /**
    * Sets Response object
    * @param CtrlHttpResponse object
    */
    public function setResponse(CtrlHttpResponse $response)
    {
        $this->_response = $response;
        return true;
    }
    /**
    * GetsResponse object
    * @return CtrlHttpResponse object
    */
```

```
    public function getResponse()
    {
        if(!$this->_response)
        {
            $this->_response = new CtrlHttpResponse();
        }
        return $this->_response;
    }
    /**
    * GetSession object
    * @return CtrlHttpSession
    */
    public function getSession()
    {
        if(!$this->_session)
        {
            $this->_session = new CtrlHttpSession();
        }
        return $this->_session;
    }
    /**
    * SetSession object
    * @param CtrlHttpSession object
    * @return true
    */
    public function setSession(CtrlHttpSession $session)
    {
        $this->_session = $session;
        return true;
    }
    /**
    * GetCookie object
    * @return CtrlHttpCookie
    */
    public function getCookie()
    {
        if(!$this->_cookie)
        {
            $this->_cookie = new CtrlHttpCookie();
        }
        return $this->_cookie;
    }
    /**
    * SetCookie object
    * @param CtrlHttpCookie
    */
    public function setCookie(CtrlHttpCookie $cookie)
    {
        $this->_cookie = $cookie;
        return true;
    }
    /**
    * Starts running application
    */
    public function run()
    {
        $page = $this->getRequest()->getRequestedPage();
        $method = $this->getRequest()->getRequestedMethod();
        /**
        * If the $mode has set get or post
        */
```

```
        if (!!($page))
        {
            if ($this->getRequest()->isPostBack() == true)
            {
                $args = $this->getRequest()->get("rsargs",array());
            }
            else
            {
                // Bust cache in the head
                header ("Expires: Mon, 26 Jul 1997 05:00:00 GMT");    // Da
te in the past
                header ("Last-
Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
                // always modified
                header ("Cache-Control: no-cache, must-
revalidate");  // HTTP/1.1
                header ("Pragma: no-
cache");                                // HTTP/1.0
                $args = $this->getRequest()-
>get("rsargs",array());
            }
        }
        /**
        * Import page
        */
        if(import($page)==false)
        {
            trigger_error("Could'nt find requested <b>".$page."</b> page",E
_USER_ERROR);
        }
        /**
        * Generate page object
        */
        $object = new $page();
        if(!$object instanceof Page)
        {
            trigger_error("$page is not instance of Page", E_USER_ERROR);
        }
        /**
        * Call page method
        */
        if(is_callable(array($object,$method)) == false)
        {
            trigger_error( "Method: <b>$method</b> is not callable in ".$pa
ge, E_USER_NOTICE);
        }
        /**
        * Call page object method
        */
        if(!!($args))
        {
            $result = call_user_func_array(array($object,$method), $args);
        }
        else
        {
            $result = $object->$method();
        }
        exit($result);
    }
    /**
    * Ends the application
```

```
    */
    public function __destruct()
    {
        //echo $this->getResponse()->getTimeLeft().' sec';
        unset($this);
    }
}
?>
```

14. CtrlComponent klass

```
<?php
if(!defined('CTRL'))
{
    exit();
}
/**
* CtrlComponent
*
* Component will manage the magic methodis __set and __get for as propertys
*
* @date 9.03.2007
* @name CtrlComponent
* @author Margo Poolak <margo 'dot' poolak 'at' gmail 'dot' com>
* @copyright Copyright &copy;2007 Margo Poolak. All Rights Reserved.
* @version 2.0
* @package System
* @category
*
*/
class CtrlComponent
{
    /**
    * Dummy construct
    */
    public function __construct()
    {
        $this->execute();
    }
    /**
    * Setter for components
    * Will set class variables:
    * <code>
    * $this->DefaultPage = 'DefaultPage';
    *  //will use:
    * $this->setDefaultPage($value);
    * </code>
    * @param string $var variable name
    * @param string $value variable value
    */
    public function __set($var,$value)
    {
        $setter = 'set'.$var;
        /*
        * If has setter method then sets the value to it
        * If not then sets a new property to application
        */
        if(method_exists($this,$setter))
        {
```

```
            $this->$setter($value);
        }
        else
        {
            Ctrl::getApplication()->setProperty($var,$value);
        }
    }
    /**
    * Getter for component
    * Can use as:
    * <code>
    * $this->Request->constructUrl();
    * //Uses this method as:
    * return $this->getRequest()->constructUrl();
    * //If that method doesnt exists then uses application property as
    * Ctrl::getApplication()->getProperty($var);
    *</code>
    * @param string $var variable name
    */
    public function __get($var)
    {
        $getter = 'get'.$var;
        /*
        * If found getter method from component then use it
        * If no getter found then use application property getter
        */
        if(method_exists($this,$getter))
        {
            return $this->$getter();
        }
        else
        {
            return Ctrl::getApplication()->getProperty($var);
        }
    }
    /**
    * Getter for Response
    * @return object CtrlHttpResponse
    */
    public function getResponse()
    {
        return Ctrl::getApplication()->getResponse();
    }
    /**
    * Getter for Request
    * @return object CtrlHttpRequest
    */
    public function getRequest()
    {
        return Ctrl::getApplication()->getRequest();
    }
    /**
    * Getter for Session
    * @return object CtrlHttpSession
    */
    public function getSession()
    {
        return Ctrl::getApplication()->getSession();
    }
    /**
    * Getter for Cookie
```

```
    * @return object CtrlHttpCookie
    */
    public function getCookie()
    {
        return Ctrl::getApplication()->getCookie();
    }
    /**
    * Returns application object
    * @return object CtrlApplication
    */
    public function getSystem()
    {
        return Ctrl::getApplication();
    }
    /**
    * Ends the component life
    */
    public function __destruct()
    {
    }
}
?>
```

15. CD sisukord: Digitaalse materjaliga

Autor: Margo Poolak 2007

Antud plaat on koostatud bakalaurusetöö "Veebi raamsüsteemid: ülevaade ja prototüübi loomine" lisana.

See plaat sisaldab kõiki vajalike faile ja katalooge,
mis on kogutud selle töö koostamisel ja tarbeks.

- "Loe mind.txt" on fail mis sisaldab antud infot

- "liivakast" kataloogis asuvad kõik näidisrakendused.
Seal asub hulk faile ja katalooge.

- "lisad" kataloogis asuvd kõik pildid raamsüsteemidest.

- "raamsüsteemid" kataloogis asuvad kõik veebi raamsüsteemid
millest juttu on tehtud, kas kokku pakituna või käivitatav fail.

Kataloogid ja failid

- Loe mind.txt

- Veebi raamsüsteemid ülesehitus ja prototüübi loomine.doc

- liivakast

    - adodb
    - code_igniter
    - prado
    - prototype
    - rubyonrails
    - scriptaculous

- lisad

    - adodb_structure.gif
    - ajax_web_application_framework.png
    - ci_framework_structure.gif
    - ci_helloworld_structure.gif
    - ctrl_framework_structure.png
    - ctrl_project_structure.png
    - ctrl_web_application_framework.png
    - prado_framework_structure.gif

- prado_helloworld_directory.gif
- ror_helloworld_structure.gif
- rubyonrails_directory.gif
- scriptaculouse_structure.gif
- symfony_directory.gif
- web_application_framework.png

- raamsüsteemid

    - ADOdb
        - adodb500beta.zip
        - ADOdb.Manual.chm
        - adodb-ext-504.zip
    - Code Igniter
        - CodeIgniter_1.5.0.1.zip
    - Ctrl
        - ctrl.zip
    - Prado
        - prado-3.1.0a.r1626.zip
    - Prototype
        - prototype-151-api.pdf
        - prototype.js
    - Ruby on Rails
        - ruby185-24.exe
    - Scriptaculous
        - scriptaculous-js-1.7.0.zip
    - Struts
        - struts-2.0.6-all.zip
    - Zope
        - python-2.5.msi
        - Zope-2.10.3-win32.exe
        - Zope-3.3.1.win32-py2.4.exe

    - php_coding_standard.doc
    - web_application_framework.png

- raamsüsteemid

    - ADOdb
        - adodb500beta.zip
        - ADOdb.Manual.chm
        - adodb-ext-504.zip
    - Code Igniter
        - CodeIgniter_1.5.0.1.zip
    - Ctrl
        - Ctrl_Prototype.zip
    - Prado
        - prado-3.1.0a.r1626.zip
    - Prototype
        - prototype-151-api.pdf
        - prototype.js
    - Ruby on Rails
        - ruby185-24.exe
    - Scriptaculous
        - scriptaculous-js-1.7.0.zip
    - Struts
        - struts-2.0.6-all.zip
    - Zope
        - python-2.5.msi
        - Zope-2.10.3-win32.exe
        - Zope-3.3.1.win32-py2.4.exe

    - php_coding_standard.doc