TALLINNA ÜLIKOOL
MATEMAATIKA – LOODUSTEADUSKOND
INFORMAATIKA OSAKOND

VEEBIRAKENDUSTE RIKASTAMINE VEEBIPÕHISE TESTIMISSÜSTEEMI NÄITEL

Autor: Avar Pentel
Juhendaja: Jaagup Kippar

Autor .....”…..” 2005
Juhendaja .....”…..” 2005
Osakonnajuhataja.....”…..” 2005

Tallinn 2005

Sisukord

1 Sissejuhatus

Suur osa lauaarvuti tarkvarast on püsinud ilma oluliste muudatusteta pikki aastaid. Millised on need funktsioonid kaasaegses tekstiredaktoris, mida ei saanud kasutada kümme aastat tagasi? Kui paljud inimesed neid kasutavad või neist üldse kuulnud on. Microsoft Word 2003, sisaldab 1500 käsku (Nielsen 2005), aga mitu neist reaalselt kasutust leiab?

Ometi inimesed maksavad selle eest, mida nad ei kasuta. Tõsi küll, tihti ostetakse ka muid asju igaks juhuks varuks, inimesed varuvad talveks kartuleid ja teevad hoidiseid. Ning vahest soetatakse ka suisa mõttetut kraami. Kuid kindlasti on tänapäeval kasvav hulk inimesi, kes on sellisest varumisharjumusest priiks saanud. Mina armastan, minna poodi, kui mul on midagi vaja. Võibolla on ennast ammendanud ka tarkvaraarenduse tee, milles projekteeritakse valmis kõike tegev arvatavalt ideaalne toode, mis peaks kõikide inimeste kõik vajadused rahuldama.

Pole ime, et viimasel paaril aastal ilmunud lemmikrakendusi meenutades, tõdevad paljud inimesed, et need rakendused ei asugi enam nende arvutis. Käesolev töö ongi ühest liigist selliste rakenduste seas. Nimetan neid veebirakendusteks (*Web Application*). Uurimuse üldisema probleemiasetuse võib lühidalt sõnastada järgmise kolmeosalise küsimusena:

- mis on veebirakendus,
- mis on selle puudused,
- kuidas neid ületada?

Puuduste ületamist käsitlen E-õppekeskkond IVA testimissüsteemi näitel.

Järgnevalt käsitlen neid kolme punkti natuke lähemalt. Esiteks küsimus, mis on veebirakendus, vajab vastamist selleks, et paremini mõista neid tingimusi ja nõudeid, mis veebirakenduse kohta kehtivad. Veeb ei ole ühtne tervik, mille kohta saab kirjutada igavesti ja igas olukorras kehtivaid juhtnööre Ei saa näiteks öelda, et kasutaja ootused on online ajalehe lugemisel ja veebimaili kasutamisel täpselt samad. Seetõttu

püüan võimalikult täpselt positsioneerida need alamhulgad, kuhu kõigepealt üldisemalt veebirakendus kuulub ja seejärel konkreetsemalt veebipõhine testimissüsteem.

Olles täpsemalt määratlenud uuritava objekti (peatükid 2 ja 3), uurin sellega seonduvaid võimalikke probleeme (peatükid 3 ja 4). Kas olemasolev süsteem on rahuldav, või oleks seal soovitav teha muudatusi. Kaardistanud soovitavad muudatused, asun otsima võimalikke lahendusteid. Analüüsin veebitehnoloogia arengut (peatükk 5), et välja valida lõpplahendusteks sobivaim tehnoloogia.

Seejärel (peatükk 6) käsitlen veebirakenduste probleemseid kohti juba valitud tehnoloogia võimalustest lähtuvalt, esitades esmalt üldiseid lahendusvõimalusi ja tutvustades nende ajalugu ja analüüsides eri lahenduste plusse ning miinuseid. Lõpuks (peatükid 7, 8, 9 ja 10) keskendun konkreetselt IVA testimissüsteemile ja rakendan esitatud üldised lahendused spetsiifilisemasse konteksti.

2 Veebirakenduse defineerimine

Palju on kirjutatud veebist ja sellest, mis see on ja milline see peab olema. Ning mitmed väljaöeldud mõtted on kristalliseerunud, neid on korratud kuni nad on saanud dogmaks. Kuid veeb ei ole nii ühtne ning muutumatu tervik, et selle kohta saaks kehtestada kivisse raiutud reegleid. Väidan, et infokeskne veeb ja veebirakendus erinevad teineteisest nagu raamat ja sulepea või muu tööriist. Kahjuks on populaarsed juhtnöörid veebi kohta kirjutatud valdavas enamuses lähtudes infokeskse veebi põhimõtetest ning see on avaldanud mõju ka veebirakendustele. Oma rolli on mänginud kindlasti see, et ajal, mil veebi kasutajakesksusest hakati rääkima ja kirjutama, veebirakendusi veel ei olnud. On autoreid (Krug 2006, xi), kes on tunnistanud, et nad pole õiged inimesed veebirakendustest kirjutama.

Jesse James Garret oma raamatus *Elements of User Experience* (Garret 2002, 28-29) kirjutab, et kui kasutajate veebikogemused hakkasid välja kujunema, siis eraldusid kaks arendajate kogukonda, kes rääkisid eri keelt. Üks grupp nägi veebiga seonduvas tarkvara disaini problemaatikat ja selle lahendamiseks püüdis kasutada viise, mis on tüüpilised lauaarvuti rakenduste ja üldse tarkvara arenduse valdkonnas (need omakorda rajanevad igasuguste muude toodete loomise praktikal, hõlmates valdkonda autodest jooksukingadeni). Samas teine grupp nägi veebi informatsiooni kogumina ja edastusvahendina ning rakendas probleemide lahendamiseks tehnikaid, mis on traditsioonilised publitsistika, meedia ja infoteaduse valdkonnas. See tekitas olukorra, kus kaks gruppi ei suutnud rääkida ühte keelt ja kokku leppida põhiterminoloogias. Vett ajas sogaseks veel asjaolu, et veebi ei saagi jagada väga selgelt ei rakendusteks ega hüperteksti kogumikeks, sest suur osa sellest moodustab justkui mingi hübriidi, sidudes endas mõlema omadusi.

Internet WWW kujul on olnud vaid lühikest aega meie ümber ja seetõttu oleme harjunud kutsuma veebilehtedeks kõike, mis meieni jõuab veebibrauseri vahendusel. Kuid eesmärkide järgi saab selgelt eristada kahte liiki veebilehti. Mõnede lehtede väärtuseks on nende sisu, olgu selleks andmed, mida otsime või mingi meelelahutuslik aspekt. Samas on lehti, mis võimaldavad meil teatud ülesandeid täita, näiteks pangaülekandeid teha, maile saata, oma ePortfooliot arendada, kuni komplitseeritud

autorsüsteemideni välja. Selgub, et veebilehed eristuvad ja liigituvad sisule orienteerituse (*Content-Oriented*) ning ülesannete täitmisele orienteerituse (*Task-Oriented*) järgi. Veebirakendus ongi veebikeskkond, mis on ülesannete täitmisele orienteeritud.

Käesoleva töö uurimisvaldkonnaks veebirakendused. Uurimisprobleem puudutab veebirakendusi sellest seisukohast, mis on veebirakenduste puudused ja kuidas neid ületada. Et konkretiseerida oma küsimuseasetust, siis tuleb täpsemalt määratleda, mis vahe on veebirakendusel ja veebilehestikul (*Web Site*) ja tuua välja neid eristavad olulised üksiktunnused. Kolmanda alaliigina lisanduvad veel veebiteenused, mis oma keerukuse tõttu sarnanevad tihti pigem rakendustega, kuid mille eesmärk on siiski edastada kasutajale mingit informatsiooni. Antud liigitus ei pretendeeri absoluutsele tõele vaid on oluline käesoleva töö piiritlemise seisukohalt. Järgnevalt esitan täpsema ülevaate tunnustest, mis rakendusi, teenuseid ja veebilehti eristavad.

2.1 Veebilehti, - teenuseid ja -rakendusi eristavad tunnused

Täpsustan, milles seisneb informatsioonile orienteeritus veebilehtede ja veebiteenuste puhul ning ülesannetele orienteeritus veebirakenduste puhul. Peamiseks eristavaks kriteeriumiks on see, kas kasutajale on võimaldatud vaid ligipääs andmetele või on tal võimalus andmete olukorda ka muuta (Baxley, 2003), see tähendab muuta eemalolevas serverisasuvaid andmeid. Nagu eelnevalt ütlesin, ei pretendeeri see definitsioon absoluutsele tõele. Lauaarvuti rakenduse definitsioon on märksa laiem. Kuid lauaarvutis on pole ka veebilehti. Siiski on seal mitmeid ainult lugemiseks mõeldud abifaile (*help*), mida veebilehtedega võrrelda võib. Kuid kuhu liigituks näiteks tavaline kalkulaatoriprogramm, millega andmeid muuta ei saa? Kas rakenduste või teenuste alla? Lauaarvutis oleks väga loomulik öelda selle kohta rakendus, kuid käesoleva töö kontseptsiooni järgi oleks see teenus.

Kui ainult lugemiseks mõeldud leht ja toimingute sooritamiseks loodud rakendus on kergelt eristatavad, siis teenuse ja rakenduse vahel pole üleminek nii kindlapiiriline. Näiteks ei saa selgelt piiritleda arvutimänge. Enamasti oleksid need pigem teenused, sest kui mingit andmete seisu permanentselt muudetakse, siis toimub see enamasti vaid tulemuste salvestamise eesmärgil. Küsimus on selles, kas tulemuse numbri muutmine

on kasutaja tahtlik tegevus, sest ka veebilehe külastaja võib muuta tahtmatult näiteks külastusstatistika numbrit ja muid andmeid mis permanentselt serveris talletuvad, aga see pole kasutaja eesmärk. Ega ühes vastust sellele küsimusele ei olegi, sest mängija kaudne eesmärk ongi head tulemust saavutada. Mängude piirid on väga laiad, ilma kahtluseta kujutavad mitmed neist rakendusi ka käesoleva definitsiooni piires.

Kuid mängude ja kalkulaatori näide on hea iseloomustamaks veebiteenuste ja veebilehtede vahet. Mäng on ju samuti tegevusele orienteeritud, kalkulaator teenindab meid arvutajana. Teenuse keskkonnas leiavad aset hoopis dünaamilisemad muudatused ja kasutaja kontroll toimuva üle on palju suurem kui tavaliste veebilehtede üle. Kindlasti võiks liigitada veebiteenuste alla mitmed kaarditeenused nagu Regio interaktiivne Eesti kaart (Regio), Eesti Maa-ameti kaardikeskuse teenus (Maa-Amet) ning Google kaardi teenus (Google Maps). Samamoodi kuuluvad sinna otsinguteenused nagu klassikaline Google ja selle mitmed uuemad lisateenused (Google Scholar, Google Suggest, Google Books).

Teine aspekt lehtede, rakenduste ja teenuste eristamisel seisneb selles, kelle vaatepunktist me süsteemi hindame. Veebisüsteemidel on erinevaid kasutajagruppe, kes esindavad erinevaid huvisid. Võib esile tõsta kolme põhigruppi:

- kasutaja lõppkasutaja tähenduses,
- kasutaja, kui omanik,
- kasutaja, kui tehnoloogia arendaja.

Vääritimõistmise vältimiseks toon iga kasutajatüübi kohta konkreetsed näited, kus sama asi võib osutuda ühele grupile rakenduseks ja teisele veebilehestikuks. Võtame näiteks e-õppe keskkonna. Selle kasutajad lõppkasutaja tähenduses on nii kursusi loovad õppejõud, kui õpilased, kes selle keskkonna kaudu mingeid ülesandeid täidavad. Nii esimesel kui teisel juhul on tegu lõppkasutaja jaoks rakendusega. Juhul kui õpikeskkond ei võimalda õpilastel teha muud, kui vaadata ja allalaadida materjale ning peale passiivse informatsiooni saamise puudub kasutajal võimalus informatsiooni seisu ise muuta, siis on kasutaja jaoks tegu veebiteenuse või veebilehestikuga.

Süsteemi omaniku jaoks võib olla varjatud rakendusi ka siis, kui kasutaja ise teadlikult andmete olukorda ei muuda. Kõige lihtsama näitena taolisest rakendusest võib tuua veebi külastusstatistika, mis võib olla kasutajale täiesti varjatud ja pole seotud kasutaja teadliku tegevusega. Siin on nüüd tõlgendamise küsimus, kas tegu on rakenduse või teenusega, kuid igatahes pole tegu veebilehega. Samamoodi võivad arendajad rakendada väga tõhusaid kasutaja tegevuse jälgimise meetmeid, et hinnata oma loodud süsteemi tõhusust või kaardistada kasutajate käitumismudeleid. Sellised lisarakendused omavad tähtsust vaid arendajatele või ka süsteemile, kui see on programmeeritud saadud andmete põhjal ise otsuseid vastu võtma.

Tehnoloogilisest vaatepunktist kujutavad mitmed veebilehestikud tänapäeval endast keerukat programmeeritud süsteemi, mis rakendab info esitamiseks päringuid erinevatesse andmekogudesse. Süsteemi arendajate ja tihti ka omanike ja haldajate seisukohast on tegu rakendusega, kuid lõppkasutaja seisukohast on tegu informatsioonikogumiga, veebilehestikuga. Samas võib olla olemuselt veebiteenus üles ehitatud tavalise staatilise veebilehestikuna. Näiteks võib tuua linkidega ühendatud küsimustiku, kus kasutaja saab jah/ei, vastusevariantide või muudele linkidele klõpsates edasi minna. Olen näinud selliseid inimese psühholoogilise tüübi määratlemise teste ja miljonimängu tüüpi viktoriine, mille taga on vaid staatilised HTML lehed. Ometi on tegu pigem teenusega, sest kasutaja annab endast märksa rohkem, kui tavalisel veebilehel huvi pakkuvale lingile klikkides.

Üheks oluliseks veebilehti ja rakendusi eristavaks tunnuseks on kasutaja seisukohalt veel individuaalse suhte aste. Kuigi see pole reegel, siis enamuse veebilehtede jaoks oleme me anonüümsed kasutajad. Teatud individuaalset lähenemist võib pakkuda vastavalt sellele, mida järeldatakse brauseri seadete ja IP järgi ja pakkuda kasutajale selle põhjal näiteks oletatavalt emakeelset lehte. Kuid multikultuurses ning mobiilses ühiskonnas me teame, et selline tehnika ei toimi.

Samal ajal autoriseerimist nõudvad süsteemid suudavad pakkuda individuaalset lähenemist sõltumata sellest, mis maailma otsast seda teenust või rakendust kasutatakse. Tõsi küll autoriseerimine pole ainult rakenduse tunnus, on infole orienteeritud lehti, mis nõuavad autoriseerimist nagu New York Times, või Google News, kuid tavaliselt sisaldub neis siis ka mingi rakenduslik element nagu seda on

kommenteerimisvõimalus esimeses ja enda valitud allikatest uudisteportaali loomise võimalus viimases.

Ja vastupidi, veebis on olemas täiesti anonüümseid rakendusi. Kuid see anonüümsuse ja isikliku suhte küsimus on üles tõstetud võrdlemaks veebirakendusi personaalarvuti rakendustega. On paradoksaalne, et kasutaja suhe personaalarvuti rakendustega on märksa impersonaalsem. Võimalus individuaalsuseks on veebirakendusel suurem kui lauaarvutis, kus võivad samuti olla kasutajad ja kasutaja individuaalsed seaded, kuid mis kehtivad ainult selle konkreetse arvuti juures. Minnes külla oma sõbrale ja avades tema arvutis tekstiredaktori, peab harjuma tema seadete ning keelega, kuid avades Google uudisteportaali on seal alati enda valitud uudised.

Kokkuvõtteks võiks esile tõsta kahte olulist tunnusjoont veebirakenduste juures:

- Võimalus permanentselt andmeid muuta – veebirakendused võimaldavad kasutajal luua, manipuleerida ja salvestada andmeid. Sellised andmed võivad võtta rahaliste ülekannete, ostu-müügitehingute või emaili saatmise vormi. Kontrastina võib tuua veebiteenuse Google.com, kus on küll võimalik sooritada päringuid, kuid mitte muuta andmete seisu. See, et süsteemi siseselt nende päringute üle arvet peetakse ja mingi andmete permanentne muutus siiski toimub, ei väljenda lõppkasutaja eesmärke.

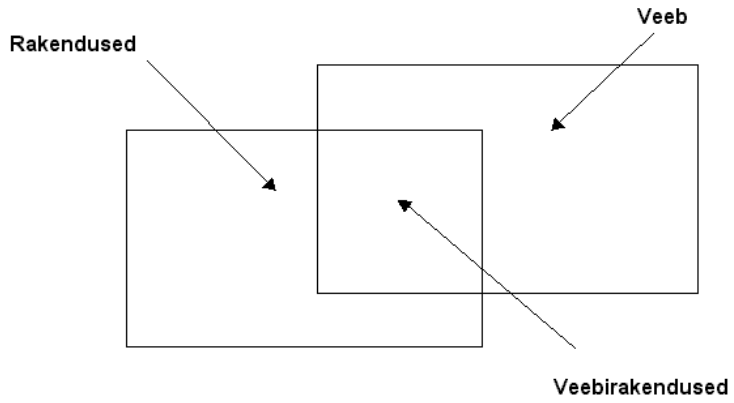
- Üks-ühele personaalne suhe rakendusega – veebirakendused võimaldavad luua unikaalse sessioonilaadse suhte iga kasutajaga. Selline võimalus puudub tavaliselt sisule orienteeritud veebilehtedel ning personaalarvuti rakendustes. Veebirakendus nagu Google veebimail teab, kes on tema kasutaja, samas kui tavaline veebileht või rakendus nagu Photoshop seda ei tea.

Käesolevas töös defineerin veebirakendust lõppkasutaja seisukohast, kui süsteemi, mis võimaldab kasutajal tahtlikult informatsiooni seisu muuta ja jagada. Konkreetse näitena käsitlen e-õppe süsteemi IVA testimiskeskkonda, kus lõppkasutajad jagunevad õpilasteks ja õpetajateks. Uurimuse jooksul pööran siiski suurt tähelepanu ka süsteemi

arendajate ja hooldajate kasutajagrupile, et väljatöötatavate tehnoloogiate rakendamine oleks nii lihtne kui võimalik.

3 Veebirakendus erinevate süsteemide ühisosana

Veebirakendus kujutab endast ühisosa rakendustest ja veebist, nagu näha järgneval joonisel (Joonis 1). On ilmne et teatud piirangud ja tingimused tulevad üle nii ühest kui teisest ülemhulgast.



Joonis 1: Veebirakenduste kuuluvus

Veebirakendus uue meediumina peab siin ühendama endas kaks täiesti erinevat interaktsioonimudelit – lehtedel baseeruva veebi interaktsioonimudeli ning ja akendel baseeruva lauaarvuti mudeli (Baxley 2002, 28). Kui kasutaja interaktsioonid piirduvad sellega, et ta liigub internetis klikkides lingilt lingile, siis on tegu veebilehestikuga. Tegu on raamatulaadse mudeliga, kus on miljoneid lehti ja igal lehel on unikaalne number. Kasutaja sõnavara, suhtlemisel serveriga oleks siis justkui piiratud väikelapse tasemele: “anna mulle seda, anna mulle teist”. Alguses HTTP ja esimesed veebibrauserid piirasidki kasutaja sõnavara sellisele tasemele. Kuid see, mis teeb veebi sobivaks hüpertekstile orienteeritud veebilehestike jaoks, ei ole pruugi olla hea toimingutele orienteeritud tarkvara rakenduste puhul (Garret, 2005).

Siiski on suur hulk veebirakendusi siiani rajatud samale “anna mulle seda, anna mulle teist” põhimõttele. Kui lokaalsesse arvutise installeeritud rakenduste puhul oleme harjunud kiire tagasiside ja vahetu kontrolliga rakenduse üle, siis veebirakenduste juures võib kohata venivaid lehekülgi, mille avanemise jooksul kasutaja juba unustab, mida ta tegema tuli (Nielsen 1997, Nielsen 1993, 136, Selvidge 1999).

On väidetud, et kiiruse probleemid lahendab püsiühendus, kuid see kehtib vaid siis, kui kasutaja ja serveri vahel mingeid pudelikaelu ei ole. Kiiruse kasvades lisandub ka kasutajaid, vahendatakse uut ja andmemahukamat meediumit ja kasvab kasutajate hulk, kes pole inimesed. Pealegi ei ole kiirus veebilehestiku interaktsioonimudeli ainus puudus. Ülikiirete ühenduste korral võib saavutada küll lehtede kiire laadimise, kuid interkatsioonide arv, keskkonna reageerimine kasutaja vahetule tegevusele jääks ikkagi piiratuks HTTP protokollis defineeritud lingile klikkimise ja vormi postitamisega. Kasutaja eelmainitud tegevuse tulemuseks on alati uue lehekülje laadimine serverist.

Lehekülgede mudelile üles ehitatud veebirakenduste peamine puudus on vaene interaktsioonimudel ja suutmatus reageerida sündmustele. HTML ja CSS keelte abil loodud kasutajakeskkonnal puudub intelligentsus. Lauaarvuti rakendustega võrreldavat käitumist, vahetut reageerimist sündmustele ja kasutaja vahetut kontrolli keskkonna elementide üle saab saavutada vaid kliendipoolsete intelligentsete tehnoloogiate rakendamisega.

Järgnevas alapeatükis positsioneerin veebirakendust natuke väiksemas rakenduste alamhulgas - klient-server süsteemides ja näitan kuidas kliendipoolsete tehnoloogiatega rikastatud veebirakendus kuulub hetkel optimaalseimasse klient-server süsteemi kategooriasse.

3.1 Veebirakendus klient-server süsteemina

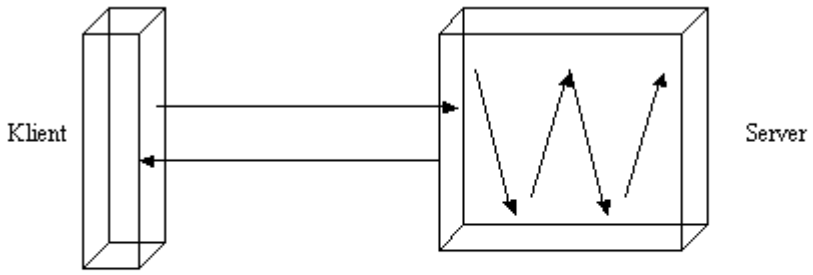
Aastal 1999 pakuti veebiarenduse teemalises uudisgrupis ee.arvutid.www.webmasters välja klient-server süsteemi kohta uussõna sirver-server süsteem (Mänd 1999). Sõna sirver sobib hästi tähistamaks brauserit, kuid klient-server süsteemides ei sobiks see tähistama kõiki kliente, mis ei pruugi olla brauserid. Seetõttu võib seda vaimukat sõna lugeda õnnestunuks kitsamalt vaid veebisüsteemide tähistamiseks. Siiski, sõna sirver tuletab meelde lehtede sirvimise mudelit, mis pole kõige parem loogika rakenduste jaoks.

Klient-server süsteem, nagu nimigi ütleb, koosneb kliendist ja serverist. Server ei pea tähendama eemalolevas arvutis paiknevat süsteemi, samamoodi võivad klient ja server eksisteerida koos ühes arvutis. Kuid süsteemi idee peaks vähemalt võimaldama serveri

paigaldamist ka eemalolevasse arvutisse. Erinevaid klient-server süsteeme, liigitatakse tavaliselt kliendi paksuse järgi. On peene ehk õhukese kliendiga süsteemid, paksu kliendiga süsteemid ning vahepealsed ehk parajad. Paksus sõltub sellest, kuidas on jagatud rakenduse loogika. Järgnevalt esitan lühiülevaate kolmest klient-server süsteemi liigist, ning seejärel analüüsin iga grupi eeliseid ja puuduseid.

3.1.1 Peen klient

Peene või õhukese kliendi (*thin client*) puhul asub enamik rakenduse loogikast serveris. Peenikesed kliendid on olnud kasutusel juba üsna kaua terminali rakenduste näol. Veebirakenduste sünniga on ilmunud uus liik terminalirakendusi, mille kliendiks on veebibrauser. Kui kliendi poole saadetakse staatilised vaated HTML lehtede näol, ja selle üle mida saadetakse ja mida tehakse vastuvõetud andmetega, otsustab server, siis on tegu kahtlemata õhukese või peene kliendiga.



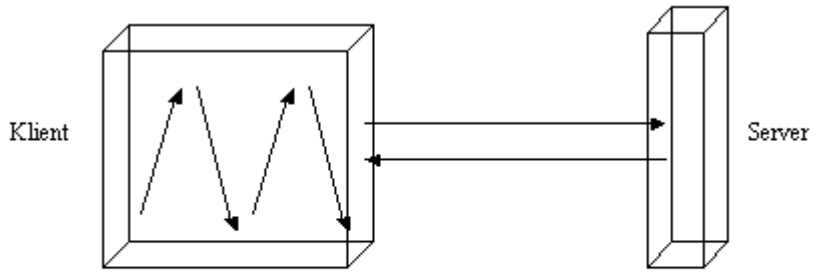
Õhukese kliendi puhul asub kogu rakenduse loogika serveris ja kliendile saadetakse vaid vajalikke vaateid (lehti). Tagasiside kliendi tegevusele saab toimuda vaid läbi serveri.

Joonis 2

Veebirakendusi on realiseeritud peamiselt õhukese kliendina, kuna serveri keskkond on arendusplatvormina selgelt piiritletud ja serveri üle on arendajatel kontroll, samas kui kliendipoolsed interpreteerimisvõimalused on varieeruvad ja nende üle kontroll puudub, kui tegu pole just kinnise intranetiga.

3.1.2 Paks klient

Paksu kliendi (*fat client*) puhul asub enamus rakenduse loogikast kliendis. Tegu on tihti mingi graafilise kasutajakeskkonnaga, mis suhtleb samas masinas või kaugemal asuva serveriga. Serverist päritakse vaid hädavajalikke andmeid, nende esitamine töötlemine ja otsesed interaktsioonid kasutajaga on realiseeritud kliendis. Apstest (Apstest 2004) on näide paksust kliendist. 90-date keskel olid paksu klindiga rakendused üsna populaarsed, Eestis olid kasutusel paksu kliendi tehnoloogial rajanevad riiklikud ja omavalitsuste registrid, muud andmebaasisüsteemid ja raamatupidamissüsteemid.



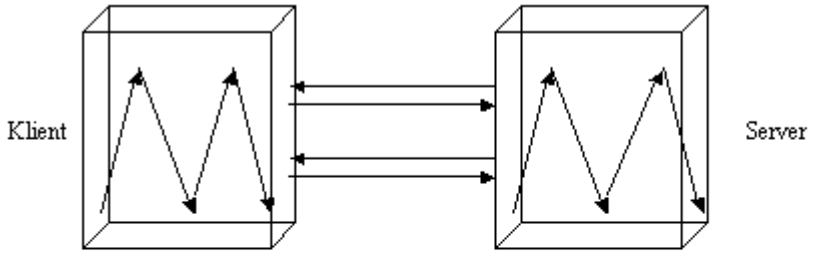
Paksus kliendis asub kogu programmi loogika ja serverist võetakse vaid aegajalt andmeid

Joonis 3

Reeglina tuleb paks klient installeerida tööjaamadesse. Veebitehnoloogiaid kasutades võib teha ka paksu klindiga rakendusi, mida installeerima ei pea, kuid neid võiks juba pidada iseseisvaks, järgnevalt käsitletavaks paraja kliendi liigiks.

3.1.3 Paras klient

Paras klient eriliigina on uuem, tavaliselt veebiga seotud nähtus (Strahl 1999, 22). Nagu nimestki võib järeldada, kujutab paraja kliendiga süsteem kombinatsiooni paksu ja peene kliendiga süsteemide omadustest. Selle liigi juures on rakenduse loogika jagatud kliendi ja serveri vahel.



Paraja kliendi puhul on loogika kliendi ja serveri vahel jagatud. Klientrakendus reageerib koheselt kasutaja tegevusele ja ei pea selleks ära ootama vastust serverist.
Andmevahetus serveriga võib toimuda taustal asünkroonselt, katkestamata muid tegevusi klientrakenduses.

Joonis 4

Vajadusest ühendada veebirakenduses nii kliendi kui serveri paremaid omadusi on olnud juttu juba varem (Bosworth 1998). Viimasel ajal ongi toimunud tuntav nihe veebirakendustes parajate klientide lahenduste poole, mis kasutavad ära võimsuse, mida kliendi operatsioonisüsteem pakub, jättes samal ajal siiski suure osa andmetöötlusest ja loogikast serverile. Parajad kliendid võivad keskenduda tööriistadele, mida pakub kliendi brauser nagu DHTML või tänapäeval uue moesõnana leviv AJAX. Teatud mööndustega kuuluvad tööriistade komplekti ka brauseri lisad Java, Flash ja ActiveX tehnoloogia.

Mis tegelikult eraldab parajat klienti paksust kliendist, on see, et paras klient keskendub standardsetele tehnoloogiatele, mille interpretaatorite olemasolu on üldine (veebibrauser) või milliste interpretaatorite olemasolu on laialt levinud (Java, Flash). Sellises rakenduses vastutab kliendi pool serveri kontrollimise eest, ja tegeleb samal ajal enamikuga kasutajakeskkonna interaktsioonidest. Samas jäetakse kõik suuremat arvutusvõimsust nõudvad toimingud serverile.

Muidugi pole nende kolme vahel piirjooned alati selged, kuid igal grupil on kahtlemata omad eelised ja puudused, mida järgnevalt põhjalikumalt analüüsin.

3.1.4 Paksu, peene ja paraja kliendi võrdlus

3.1.4.1 Paks klient - plussid

Alustades paksust kliendist, võib suurima eelisena välja tuua võimaluse saavutada suurim sarnasuse tavaliste lauaarvuti rakendustega. Keskkonna käitumine on harjumuspärane, reageerimine kasutaja tegevusele on kiire ja ei katkesta töökulgu. Need ilmsed plussid on tagatud sellega, et tegelikult ongi klient lokaalne rakendus, mille üks ülesannetest on vahendada andmeid serveriga. Erinevalt peenest kliendist, vahetatakse serveriga vaid vajalikke andmeid. Server ei dikteeri ega loo kasutaja loogikat.

3.1.4.2 Paks klient – miinused

Paksu kliendi suurim miinus on süsteemi kasutuselevõtmise ja täiustamise ning hooldamise suur hind. See tuleneb vajadusest kliendi installeerimiseks töökohaarvutitesse ning edasiseks täiendamiseks ja hooldamiseks kohapeal. Isegi väikses riigis nagu Eesti on mõeldamatu, et selliseid süsteeme käiakse kohapeal hooldamas ja vajadusel muutmas. Mul on olnud mitmeid kogemusi selliste paksudele lokaalselt installeeritud klientidele rajatud registritega, mille puhul üks hooldaja peab sõitma mööda paarisaja kilomeetri raadiuses asuvaid kliente ja tegema rakenduses muudatusi, mis tingitud seadusmuudatustest, vigadest või kliendi soovist riistvara uuendada.

Ka paksu installeeritava kliendi tarkvaraarendus on pikem protsess, sest tuleb väga palju ette näha, süsteemi väga hoolikalt testida ning lisada palju funktsionaalsust nii öelda igaks juhuks. See on nagu lennuk, mida õhus on juba väga raske parandama hakata. Samuti kulub palju aega selleks, et luua kergelt installeeritav süsteem ning valmistada ette kasutus- ja installeerimisjuhendid.

3.1.4.3 Peen klient –plussid

Kui varem peeti terminaliklientide suureks eeliseks kokkuhoidu kalli riistvara pealt, siis nüüd on erinevused riistavahindades kahanenud. Esiplaanile on tulnud süsteemi kasutuselvõtmise ja hooldamisega seotud kulud. Peene kliendi puhul pole vaja midagi töökohaarvutitesse installeerida, hooldus ja täiendamine toimuvad keskselt,

levitamiskulud puuduvad. Veebiklientide puhul võib rääkida platvormist sõltumatusest ning võimalusest kasutada süsteemi igal poolt, kus on ligipääs veebile.

Erinevalt paksust kliendist, saab peene kliendiga süsteemi varem käiku anda ja hiljem vajadusel jooksvalt funktsionaalsust lisada või muuta. Tarkvara kalli testimise asemel laboris ja piiratud hulga kasutajatega, saab rakendada pidevat reaalsete kasutajate jälgimist.

3.1.4.4 Peen klient –miinused

Peene kliendi suurimad miinused seisnevad süsteemi jäikuses. Kuna kliendi ressursse kasutatakse vaid minimaalselt serverist saadetud andmete töötlemisel ekraanipildiks, siis toimub ka reaktsioon kasutaja tegevusele vaid uue pildi saatmise tulemusel. See tähendab ühest küljest piiratud arvu sisendeid, millega kasutaja saab opereerida ja teisest küljest tagasiside sõltuvust ühenduse kiirusest. Veebis järgneb igale tegevusele päring serverisse, mis katkestab kasutaja töö seniks, kuni klienti saadetakse uus ekraanitäis. Isegi kiire ühenduse korral kulub aega kogu ekraanipildi taaskuvamiseks ja iga tegevuse peale kasutajakeskkond lihtsalt kaob mõneks ajaks.

Vahest on kliendil vaja serverist saada vaid ühte kindlat numbrilist väärtust, mis kujutab endast vaid paari baiti informatsiooni, kuid selle asemel saadetakse talle kogu kasutajaliidese kuvamiseks vajalik kood, mis kümnetes või sadades tuhandetes kordades ületab tegelikult vajaliku info hulka.

Põhimõtted rakenduse reageerimisaja kohta toodi esile juba aastal 1968 (Miller 1968) ja selle järgi peeti soovitavaks reageerimisaega 0,1 sekundit, et reaktsioon tunduks kohene. Kui reageerimisaeg on 1,0 sekundit, siis kasutaja märkab viivitust, tekib ootamise tunne, kuid see veel olulisel määral ei häiri kasutajat. Kui reageerimisaeg on 10 sekundit, siis on see viimane piir, et kasutaja tähelepanu veel mingil dialoogil hoida. Seejärel tahavad nad juba midagi muud teha.

Lehekülgedel rajanev veebirakendus ootab tänapäeval kasutajalt suuremat kannatust kui 1968 aasta omalt. Ometi viitavad mitmed märgid sellele, et inimeste tähelepanu ulatus (*Attention Span*) on hoopis vähenenud, isegi filmide kaadrivahetused on palju sagedasemad, kui nelikümmend aastat tagasi.

HTML pakub kasutajaliidese loomiseks küll suure valiku vormielemente ja võimaluse neid sobival moel paigutada, kuid puudub igasugune kohese reageerimise võime sündmustele, ehk kasutaja tegevusele. Arvestades ajastut, kus rakendus tähendab reeglina väga interaktiivset keskkonda, siis õhukese kliendi rakendused läinud justkui ajas tagasi 3270 terminalide[1] aega (Flanagan 1997). Viimasel ajal on teatud interaktiivsust võimalik lisada kasutades ära CSS pseudoklasse, kuid kuna CSS on kujunduskeel, siis jääb selle võimalustest keerukamate süsteemide arendamisel väheseks.

Peene kliendiga veebirakenduste mudel on hoopis erinev sellest, millega ollakse harjunud lauaarvuti rakendustes. Järjepidevus (*consistency*) on sagedamini loetletud kasutusmugavuse atribuute (Nielsen 1993, 132), kiire reageerimise (Nielsen 1993, 135) kõrval. Järjepidevus tähendab harjumuspärasust, seda, et erinevates programmides toimuvad sarnased tegevused samamoodi ja näevad välja samamoodi. Hiljem turule tulevad rakendused peavad järgima sarnaseid printsiipe nagu varasemad, et vastata kasutajate harjumustele. Peene kliendi veebirakendus rajaneb lehtede mudelile, mis nõuab kasutajalt ümberõppimist ja harjumist.

3.1.4.5 Paras klient – plussid

Paraja kliendi eesmärk on siduda endas eelnevate liikide plussid. Kasutades efektiivsemalt brauseri poolt pakutavaid tehnoloogiaid nagu HTML, JavaScript ja CSS, saavutatakse sarnasem kasutajakogemus paksu kliendi rakendustele. Rakendusele luuakse kliendi pool töötav kasutajaliidese käitumist ja serverit kontrolliv kiht. Kasutajaliides hakkab reageerima enamatele sündmustele ja pöördub serveri poole vaid siis, kui see on vajalik ning võib seda teha ka taustal katkestamata kasutaja töökulgu. Kliendipoolsete tehnoloogiate rakendamine võimaldab järgida rakendustes väljakujunenud konventsioone ja see teeb paraja kliendi kasutajale arusaadavamaks.

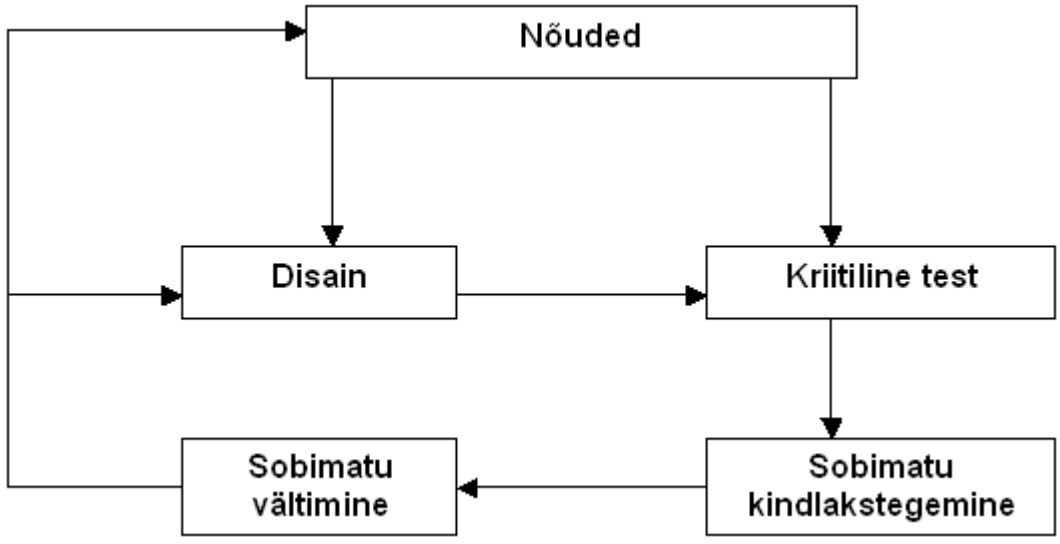
---

[1] Seitsmekümnendate keskel levinud IBM 3270 terminalid kuulusid juba intelligentsemate terminalide klassi, kuna varasemate videoprinteri tüüpi terminalidega võrreldes võimaldasid need suuremat interaktiivsust, näiteks kontrolli kursori üle ekraanil. Paralleel veebiga seisneb selles, et tavaliselt suhtles 3270 terminaliklient serveriga läbi vormide, millest andmete saatmise ja vastuvõtmise ajal oli kliendi klaviatuur lukustunud.

Kuna kliendipoolne süsteem rajaneb siiski standardsetele tehnoloogiatele, siis puudutavad seda ka samad plussid, mis peent klienti – need on platvormist sõltumatus, installeerimisvajaduse puudumine ning keskne haldus.

Paraja kliendiga tekib uus paradigma tarkvaraarenduses: kui paksu kliendi loojaks on kõike etteplaneeriv arhitekt, siis paraja kliendi arendaja on nagu aednik (Murugesan, Desphande 2001). Veebi keskkond ja brauser kui platvorm ei ole nii selgelt piiritletud ja selle areng ning vajadused on väga mitmekesised. Seetõttu on ka arendustöö pidev protsess, mitte ei keskendu konkreetsete, teineteisest selgelt eraldatavate versioonide turuletoomisele. Asju proovitakse kiiresti läbi ja parandatakse (Joonis 5). Tegu on pigem evolutsioonilise kui planeeritud protsessiga. Suured plaanid lihtsalt ei tööta enam. Nagu ütleb Google tarkvaraarenduse asedirektor (*vice president*) Adam Bosworth, tegu pole intelligentse disainiga[2], vaid intelligentse reaktsiooniga (Bosworth 2005).

Joonis 5: Disain evolutsioonilise protsessina, kohandatud DasGupta (1991, 79) järgi



[2] Intelligentne Disain on kreatsionistide evolutsiooniteooriale vastanduv teooria, mis keskendub lünkade otsimisele evolutsiooniteoorias ja püüab osutada sellele, et elu teke (ja laiemalt ka kogu universumi teke) on mõistuslikult planeeritud disaini tulemus. Samas kõnes võrdleb Adam Bosworth varasemat paradigmat ka nõukogudeaegse plaanimajandusega.

3.1.4.6 Paras klient – miinused

Paksu kliendi miinusteks olid peene kliendi plussid ja vastupidi, seetõttu parajale kliendile arusaadavalt mingeid suuri miinuseid jääda ei saagi. Päris probleemidevaba olukord siiski pole. Brauseri dokumendi objektimudel ning selle manipuleerimine JavaScripti abil ning mitmed lisavõimalused on küll oma põhiosas suhteliselt hästi standardiseeritud, kuid siiski tuleb pidevalt arvestada teatud erinevustega ja lähtuda W3C spetsifikatsioonide ja soovituste asemel reaalsete brauserite (teatud juhtudel nõrgalt dokumenteeritud) implementatsioonidest. Mitmed vajalikud, erinevate brauseritootjate poolt toetatud ja töötavad tehnoloogiad, ei ole üldse standardiseeritud (näiteks WYSIWYG redigeerimine, XMLHttpRequest objekt).

Kuna klienti korraga laetava koodi hulk, ei saa olla väga suur, siis tuleb jällegi rakendada hoopis uut lähenemist tarkvaraarenduse seisukohalt. Ka kogu rakenduse kliendiloogika tuleb väga kaalutletult osadeks jagada ja laadida klienti parajate osade kaupa. Kui klienti laaditavate andmete hulk on suur, siis rakendatakse ka ennustavaid tehnoloogiaid, mis kasutaja tegevuse järgi püüavad ära arvata, milliseid andmeid või funktsionaalsust järgmisena vaja läheb, et alustada nende laadimist juba enne, kui kasutaja neid küsib ja tagada sellega momentaalne tagasiside. Paras klient peab suhtlema kasutajaga teatud mõttes sarnasel tasemel nagu inimene suhtleb inimesega tehes järeldusi nii liigutuste ja miimika põhjal või mõistes teist poole sõna pealt. Paraja kliendi tarkvara intelligentsuse määr peab olema märksa suurem kui paksu kliendi puhul ja see nõuab paratamatult lisakulusid.

Tihti tuleb paraja kliendi loomisel arvestada ka nende kasutajatega, kelle brauserite interpreteerimisvõimalused ei vasta paraja kliendi tehnoloogilisele miinimumile. Sel juhul tuleb säilitada ka peene kliendi loogika. Mõnikord on võimalik rajada paraja kliendi loogika õhukese kliendi peale neid omavahel segamata, kuid alati pole see võimalik ja tuleb luua sisuliselt erineval loogikal töötavad rakendused.

Kokkuvõtteks võib eri klienditüüpide plussid ning miinused esitada järgmise tabelina:

Tabel 1

Kliendi liik	Kasutusmugavus	Hoolduskulud	Kokku
Paks	+++	----	-
Peen	--	+++	+
Paras	++	++	++++

Eri klienditüüpide analüüsi tulemused võib kokku võtta järgmiselt:

- **Paks klient** on oma aja ära elanud. Üha tihedam vajadus tarkvaras muudatusi teha, suurenev kasutajate arv ja kõrged hoolduskulud ei kaalu ülesse suuremaid võimalusi kasutajamugavuse tagamiseks.

- **Peen klient**, mis on siiani populaarseim veebirakenduse liik paistab silma sellega, et kõik plussid asuvad süsteemi valdaja, hooldaja ning arendaja poolel ning miinused kasutaja poolel.

- **Paras klient**, lisab tööd süsteemi arendajatele ja võib sellega vähendada plusside arvu, kui süsteemi kasutajaid on vähe. Võimalus tarkvara keskselt ning jooksvalt muuta, parandada ja arendada kaalub siiski üles enamuse miinuseid. Puuduvad kulud tarkvara levitamiseks. Kokkuvõttes on paraja kliendiga süsteem hetkel eelistatuim.

4 Veebilehtedele ja veebirakendustele esitatavate nõuete võrdlev analüüs

On publitseeritud hulk juhtnööre, kuidas peaks veebi disainima. Google otsing "*web usability guidelines*" pakkus poole aasta eest 1 300 000 vastet, nüüd juba 3 720 000. Tundub, et inimesed tahavad sellest rääkida. Kuid rääkides millestki nii subjektiivsest nagu kasutusmugavus, ei ole oodata ka suur konsensust. On oht, et väiteid rebitakse välja kontekstist ja kuulutatakse kui kõikjal kehtivat tõde. Ja veel suurem oht on selles, et mingi põhjendamatu või aegunud väide võetakse omaks ilma sellele mõtlemata.

Võtame näiteks tavalise dogmaatilise väite, et menüüs ei tohi olla üle 7 elemendi. Sellest räägitakse väga palju ja seostatakse seda inimese lühimälu võimega mitte mahutada seal rohkem elemente. Põhiliselt baseeruvad need kõik Georg Milleri (Miller, 1956, 81-97) 1956 aastal publitseeritud artiklis "*The magical number seven, plus or minus two*", milles ta diskuteerib numbri seitse laialdase esinemise üle kultuuris ja psühholoogias ning tõendusmaterjalist, et inimese lühimälu (*short-term memory*) piir on kuskil seitsme ühiku (*bit*)[3] ümber. Selles artiklis ei olnud loomulikult ühtegi viidet tarkvara kasutajaliidese disainile, kuid kaheksakümnendatel hakati seda artiklit laialdaselt tarkvara arendajate poolt tsiteerima, eriti seda osa, mis puudutas inimese lühimälu piiratust.

Lühimälu piiratust puudutav on muidugi tõsi, ja seda saab igaüks kontrollida, püüdes näiteks meelde jätta mingeid suvalisi numbreid, mida teine inimene või arvutiprogramm lühikese aja jooksul esitab, tehnika mis on kasutusel näiteks Wechsleri intelligentsustesti ühe alamtestina (Herrnstein, Murray 1996, 283). Kuid kas seda lühimälu puudutavat teavet saab üle kanda tarkvara disaini? Paljudel juhtudel saab, kuid menüüsid see kindlasti ei puuduta. Tegelikult, kui vaadata rakendustes nii laialdaselt kasutatavaid rippmenüüsid, siis nende eesmärk peitub just vastupidises – me

[3] Ehkki inimene võtab vastu väga palju infot, mille koguväärtuse juures tundub kohatu rääkida võimest käsitleda vaid ca seitset bitti samaaegselt, siis on Milleri käsitlus siiski üsna sarnane informaatikas kasutatavale (Shannon, Weaver). Miller ütleb, et bitt on valik kahe võrdväärse alternatiivi vahel nagu näiteks hinnang, kas mees on üle kuue jala pikk või lühem. Seitsme biti käsitlemine tähendab võimet hallata valida 2^7=128 valiku seast.

ei pea meeles pidama ühtegi valikut, mida see menüü meile pakub. Püüdke näiteks meelde tuletada, mis on teie brauseri või tekstiredaktori File menüü valikud. Võibolla meenub 5. Kuid kas see segab tööd.

Ajal, mis menüüvalikust, kui interaktsioonistiilist rääkima hakati, oli see lausa definitsiooni järgi valik, mida ei pea meelde jätma (Nielsen 1993, 69).

Teine valdkond, kus lühimälu probleemid ei kehti nii üks-üheses seoses, on mitmed organiseerivad tehnikad nagu tähendusrikas grupeerimine (*chunking*), mida ka Miller oma artiklis mainis. Põhimõte on selles, et andes sarnaste tunnustega objektide grupile ühise nime, saab vabastada lühimälu, mis on muidu kinni üksikute grupiliikmete all (Tulving 2002, 69).

Kolmandaks on teada, et mitmed korduvad tegevused võivad tunduvalt parandada inimese võimet adekvaatselt toime tulla väga inforikkas keskkonnas. Nii suudame täita mitmeid keerukaid funktsioone, näiteks juhtida autot, ilma sellele tähelepanu pööramata (Csikszentmihalyi 1991, 29).

Oht subjektiivseks hindamiseks või väidete dogmatiseerumiseks on medali üks pool. Teisel poolel on tõsiasi, et mingit ühtset veebi ei olegi, vaid on üks arenev ja avatud evolutsiooniline kooslus oma mitmekesisuses. Nagu käesolevas uurimuses on teemaks rakendused, mida eristan veebilehtedest, siis sisalduvad ka need hulgad väga mitmeid alamhulki, mille kohta kehtivad hoopis teised nõuded. Või kas üldse saabki rääkida nõuetest, kui see eeldab usku objektiivselt ideaalse lahenduse olemasolusse.

Mulle meeldib võrdlus evolutsiooniga ja selle sobivaima ellujäämise põhimõttega (*Survival of the Fittest*). Ainsa sobivuse objektiivse kriteeriumina on kasutatud järglaste arvu või ellujäämist reprodutseerimisvõimeni, kuid see muudab evolutsiooniteooria põhilise väite tautoloogiaks: ellu jääb sobivaim ja sobivaim on see, kes jääb ellu. Väide muutub samaväärseks püha Anselmi ontoloogilise jumalatõestusega. Kuid tsiteerides Richard Dawkinsit, kumbki neist on liiga suur selleks, et neid võiks sõnademänguga tõestada või ümber lükata (Dawkins 1999, 181).

Sobivust on mõistetud üksikute tunnuste sobivusena mingisse keskkonda, taimse toidu hankimisel osutusid oluliseks tugevad hambad ja mälumislihased, mingis teises kontekstis hea kuulmine, haistmine ning kiired refleksid. Looduslik valik põhjustas nende tunnuste paranemise ja põhjustas ka konteksti muutumisest tulenevad muudatused.

Veebi ajalugu on lühem kui bioloogiline evolutsioon, kuid ka siin on ellujääjad ja evolutsiooni surnud oksad. Ka siin on ressursid, mis on arenguks vajalikud nii kasutajate kui tehnoloogia näol. Ja nagu evolutsiooni puhul, ka siin ei ole suurt plaani, mis sest lõpuks saama peab. Seetõttu on järgmised väited pigem veebirakendusi kaitsva ja dogmaatilisi juhtnööre kummutava iseloomuga.

Kuigi mingeid universaalseid nõudeid kehtestada ei saa, siis teatud arusaamad on välja kujunenud selle kohta, mis on veebis halb tava ja mida tuleks vältida. Ma ei mõtle siin halvaks tavaks peetavate asjade all selliseid terve mõistuse vastaseid asju nagu kollane vilkuv kiri kollasetriibulisel taustal, vaid tehnoloogiaid, mis on saanud vahest täiesti teenimatut kriitikat ja on teatud kontekstis kui mitte asendamatud, siis vähemalt omal kohal. Järgnevalt toon tabeli kujul lühiülevaate neist nähtustest ja annan hinnangu veebilehtede ja veebirakenduste seisukohalt ning hiljem analüüsin igaühte neist põhjalikumalt.

Tabel 2

Veebilehed	**Veebirakendused**
Freimide kasutamine	
Freimide (*frames*) kasutamine on taunitud. Põhjuseks on probleemid lingi saatmise, printimise, salvestamisega ja leheküljele järjehoidja (*bookmark*) panemisega (Nielsen, 1996), ja *back* nupuga (Krug 2006, 58). Sama puudutab ka mitmeid uusi tehnoloogiaid, mis sisu näidates aadressirida ei muuda	Freimide kasutamise puudused ei laiene rakendustele. Kuna rakendus keskendub tegevustele, siis tegevus on lahutamatu protsess, millele ei saa viidata lingiga, ega panna tegevuste vahele järjehoidjat. *back* nupp tähendaks rakenduse loogika kohaselt *undo* funktsiooni. Kui ta sellena ei tööta, siis on parem kui ta ei tööta üldse.

Navigatsiooni säilitamine	
Üks oluline osa veebilehestikus on kasutaja orienteerumise hõlbustamine. Selleks peab hoidma kogu aeg nähtaval mingit navigatsiooni, näiteks põhijaotuste linke päises horisontaalsel kujul või vertikaalselt vasakul või paremal ning pikkade lehekülgede puhul ka lehekülje jaluses.	Veebirakenduses on lehekülgede asemel toimingud. Pigem on soovitatav oluliste toimingute juures navigatsioon hoopis ara kaotada ja jätta alles vaid nupud aktuaalse toimingu sooritamiseks, lõpetamiseks või katkestamiseks.
Pikad vs. lühikesed lehed	
Pikad vs. lühikesed lehed on olnud arutelu objektiks ammu. Samad autorid on siin esitanud vastakaid seisukohti. Kui 10 aastat tagasi tauniti pikki lehti, siis vahepeal on seda seisukohta revideeritud.	Rakendus peaks reeglina mahtuma ekraanile. Parem kui kerimisribasid ei tekiks. Erandiks on pikemad vormid, mille puhul ülevaatlikkus ja erinevas järjekorras täitmine on oluline. Testi, küsitlus-, registreerimis- või muu vormi mudeliks reaalsest maailmast on pigem selle paberkandja analoog ning seetõttu ei käsitletagi seda klassikalisele rakendusele esitavatest ootustest lähtuvalt.
Ühilduvus ja ligipääsetavus	
Informatsioon peab olema võimalikult paljudele kättesaadav, erinevate brauserite, erinevate seadmete ja füüsiliste puuete korral.	Rakenduste puhul on ühilduvus vähemoluline ja kesksel kohal on funktsionaalsus ning mugavus. Näiteks Google kaarditeenus (Google Maps) välistab teenuse kasutamise hinnanguliselt 5-10% interneti kasutajatest. Kui võrrelda lauaarvuti rakendustega, kus on nõuded riistvarale, ning operatsioonisüstemile, siis pole neist ühilduvuse osas veebirakendustele niikuinii konkurente.

Struktuur ja esitus	
Struktuuri ning esitust on kasulik hoida lahus tagamaks masinloetavust ja kättesaadavust eri seadmetele.	Struktuurile ja esitusele lisandub tihti veel käitumine läbi DOM skriptimise. Ei saa väita, et eraldamine pole vajalik, kuid kliendi pool genereeritud sisu korral on mõnikord õigustatud CSSi ja JavaScripti esitamine HTML argumentidena ja stiilide seadmine läbi JavaScripti. Masinloetavuse vajadus ei puuduta loetavust otsingusüsteemidele
Skriptide kasutamine	
Ei tohiks olla sisu, mille esitus sõltub üksnes JavaScripti või muu kliendi poolse intelligentse tehnoloogia toetusest	Mitmete rakenduste ja teenuste realiseerimine ilma JavaScripti või muu kliendipoolse intelligentse tehnoloogiata oleks võimatu.

Järgnevalt analüüsin eelolevas tabelis toodud väiteid põhjalikumalt.

4.1 Freimid ja veebilehed

Palju sellest, mida on räägitud freimide linkimise ja järjehoidjasse panemise kohta ei vasta enam tõele. Võib vaadata näiteks Microsofti (MSDN) `frame` elemente kasutavat lehekülge ja selle linkimist. Siiski peab nõustuma, et tehnoloogiad, mida tuleb rakendada freimide puhul, et ületada nende kasutamisega kaasnevaid probleeme, on tavaliselt liiga keerukad, et üles kaaluda eeliseid, mida freimide kasutamine annab. Mis puudutab järjehoidjate kasutamist, siis Internet Exploreri uuemad versioonid võimaldavad seada järjehoidja ka `frame` elemente kasutavale lehele, Firefox 1.5 seda veel ei võimalda.

Mis puutub *back* nupu mittetöötamisse siis, need väited baseeruvad 1995 aasta uuringutel, et 30-40% kõikidest veebi klikkidest moodustavad vajutused brauseri *back* nupule (Krug 2006, 58). Menüüde süsteemid ei olnud siis veel nii arenenud kui praegu ja vahest oli *back* nupp ainsaks navigeerimisvõimaluseks. Olukord on praeguseks muutunud mitmes mõttes, esiteks on loodud korralikumad navigatsioonisüsteemid ja

teiseks toetavad freimide puhul ka brauseri *back* nupu funktsionaalsust näiteks juba Netscape 3 ning Internet Explorer 3 ja loomulikult kaasaegsemad brauserid.

Põhimõtteliselt sama kehtib ka `iframe` elementide (*iframe –inline Frame*) kasutamise kohta.

Peale freimide on mitmeid uuemaid tehnoloogiaid, mis võimaldavad muuta lehekülje sisu, muutmata seejuures brauseri aadressiriba. Näiteks võib tuua DHTML tehnoloogiate abil lehekülje sisu valikulist esitamist või selle analooge, mis on saavutatud puhta CSS abil. Viimasel ajal võidavad üha enam populaarsust tehnikad, mis lubavad serverist andmeid laadida, ilma kogu lehekülge uuendamata. Kuid kui lehekülge ei uuendata, siis jääb ka aadressirida muutmata ja tekib viitamise ning järjehoidjasse seadmise probleem. Muutus brauseri aadressireal on seotud HTTP GET meetodiga, mille kutsub esile vajutus tavalisele lingile ja ka vormide postitamine, mille meetodiks on määratud GET.

Veebilehekülgedel peab olema aadress nagu raamatulehekülgedel on numbrid. Seetõttu freimid ja muud tehnoloogiad, mis aadressi samaks jätavad rikuvad seda raamatu mudelit. Ka mitmete veebiteenuste puhul, on linkimise võimalus ja vastavalt aadressi muutmine oluline. Toome näiteks kaarditeenused nagu seda pakub Maa-amet (Maa-amet 2005), Regio (Regio 2005) või Google (Google Maps). Kõikidel neil puudub võimalus järjehoidja seadmiseks ja linkimiseks. Teenuste puhul saab vabandada linkimise võimaluse puudumist sooviga tagada paremat kasutuskogemust. Näiteks selleks, et Google kaarti viidatavaks muuta, peaks igale kasutaja tegevusele järgnema kogu lehekülje uuendamine, mis oluliselt kahjustaks kasutusmugavust. Sisuliselt poleks kaardi nihutamine hiirega võimalik, kuna selle katsele järgneks kohe lehekülje laadimine, mis katkestab kasutaja tegevuse.

Alternatiivina, mille kasutamist ma kuskil kohanud pole, oleks aadressirea muutmine JavaScripti abil. Võibolla pole selle peale varem tuldud, kuna tegelikult ei saa JavaScripti abil lihtsalt aadressireale kirjutada, ilma et midagi ei juhtuks (vastasel korral kujutaks see endast turvaprobleemi, sest võimaldaks lehekülgede loomist, mis esinevad näiteks panga lehekülje nimel). Ainus võimalus aadressirea muutmiseks on objekti `location` kaudu, mille efekt on sama nagu lingivajutusel või aadressi

manuaalsel sisestamisel. See tähendab, et aadressirea muutmisega koos toimub alati ka lehekülje laadimine ja sellega on kasutaja tegevus katkestatud. Siiski ei vasta see päris tõele, sest juba varastes HTML spetsifikatsioonides on olemas aadressi lisa, mis kirjutatakse olemasoleva lõppu # märgi järele ja mis ei põhjusta lehekülje taaslaadimist. Kui nende linkide tegelik mõte on teha leheküljesiseseid viiteid, siis saab neid ära kasutada ka teatud protseduurilise viidana, kuhu kogunevad jäljed selle kohta, milliseid protseduure on kasutaja antud keskkonnas rakendanud. Sellise viite käivitamisel taastaks JavaScripti protseduur viidatud olukorra.

4.2 Freimid ja rakendused

Freimide kasutamisega loetletud puudustest mitte ükski ei laiene rakendustele, sest reeglina rakendust ei käsitleta lehtedena vaid tervikuna, milles “lehed” väljendavad erinevate toimingute etappe. Samuti võib linkimise välistada rakendustega tihti kaasnev vajadus autoriseerimiseks.

Ka järjehoidjasse paigutamine ei täida oma eesmärki, sest see on siis justkui programmi avamise nupp, nii nagu me oma arvuti töölaualt hiireklikiga avame programme, näiteks MS Wordi ja ei eelda, et see peaks avanema mingi spetsiifilise dialoogi koha pealt. Autoriseeritud rakenduse puhul muutub selline viitamine eriti mõtetuks. *Back* nupu problemaatika on samuti kadunud, rakendustes peaks back nupule vajutamist pigem vältima või tagama, et selle nupu vajutus tooks kaasa eelmise olukorra taastamise (*undo*).

Ka printimise probleem kaob ära, sest nagu desktop rakendustes ei soovi me reeglina välja printida dialoogide ja menüüde vaateid, nii on ka veebirakendustes, kus printimise funktsioon, kui sellist on vaja realiseerida, kujutab endast iseseisvat funktsiooni, mis loob, kujundab ning avab prinditava vaate.

4.3 Navigatsioon ja veebilehed

On lubamatu lasta kasutajal ekselda mööda linke kuskile lehekülje sügavusse jättes ainsaks tagasiteeks brauseri tagasi (*back*) nupu kasutamise võimaluse. Ekraaniresolutsioonide suurenedes on niikuinii tekkinud palju vaba ruumi, mida

efektiivseks sisu kuvamiseks ei saa kasutada. Ka loetav sisu ei tohi olla eriti lai, sest liiga pikad read raskendavad lugemist. Seetõttu on tavaline vertikaalse navigatsiooni paigutamine nendele tühjaksjäänud aladele. Serveripoolsete tehnoloogiate kasutamisega on see ka tehnoloogilistelt väga lihtne.

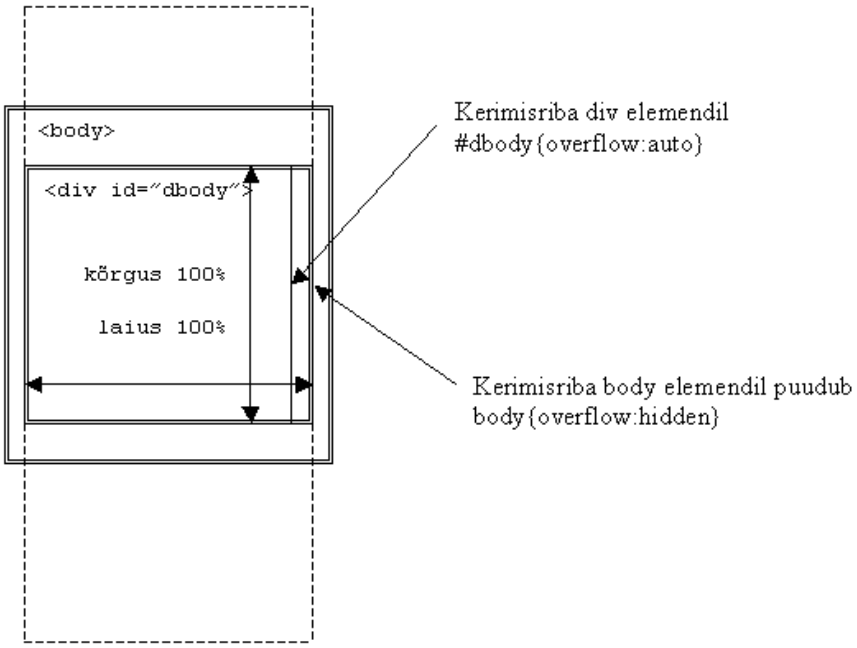
Pikemate lehekülgede puhul tekib küll olukord, kus enam ükski menüü ei paista, olukord, mille efektiivselt lahendas freimide kasutamine, kuid mille taunimine on üsna üldine ja ka teatud piirini põhjendatav veebilehtede puhul.

Freimidele sarnase, pidavalt kõrvale jääva menüü saab tekitada ka JavaScripti abil, mida jällegi ühilduvuse seisukohalt taunitakse. Antud juhul pole küll taunimiseks mingit põhjust, sest tegu oleks vaid lisaga, mis sõidutab menüüd kasutaja vaateväljas, kuid JavaScripti puudumisel jääb menüü siiski alles, lihtsalt oleks staatiline.

Tänapäeval toetavad kõik enamlevinud brauserid peale Internet Exploreri 4-6 versioonide (versioon 7 juba toetab) CSS fikseeritud positsioone (`position:fixed`), mis tähendab, et elemendi asukoht jääb vaatevälja suhtes fikseerituks ka siis, kui lehekülge keritakse. Nii saab ka JavaScripti kasutamata pakkuda freimidega samaväärset kasutajakogemust, kus pika lehe kerimisel jääb menüü siiski paigale.

Teatud kõrvalteid kasutades saab ka Internet Exploreri 5 ja 6 versioonidel CSS abil sellist fikseeritud menüüd emuleerida nagu järgneva lihtsustatud näite varal demonstreerin (Koodinäide 1), kuid see seab teatud piirangud muude elementide stiili määratlemisel. Põhimõte oleks selline, et lehekülje sisse tekitatakse konteiner, mille kõrgus ja laius vastavad akna sisemistele mõõtudele (100% ja 100%), ning millele on määratud CSS abil omadus, tekitada kerimisriba siis, kui selles konteineris on sisu on rohkem, kui ekraanitäie peale mahub (`overflow:auto`). Samal ajal kaotatakse `body` elemendi kerimisribad ja loodud konteiner esineb edaspidi justkui `body` nimel (Joonis 6).

Joonis 6



Koodinäide 1

```
<style>
body{margin:0;}
#menyy{position:fixed;left:10px;top:10px;padding:10px;width:20%;
height:200px;border:2px solid black;}
#sisu{padding-left:26%;width:74%;height:100%}
</style>
<!--[if IE]><!--[if !IE 7]>
<style>
body{overflow:hidden;}
#sisu{overflow:auto;width:100%;}
#menyy{position:absolute;}
</style>
<![endif]--><![endif]-->
<body>
<div id="menyy">menüü</div>
<div id="sisu">Et näha soovitud tulemust, peab siin olema rohkem sisu
kui ekraanitäiele mahub</div>
</body>
```

Tulemuseks on leht, milles üks aknasuurune element emuleerib brauseri kerimisribasid ning kõik elemendid, mis on paigutatud absoluutsete positsioonidega jäävad siis keritava lehe suhtes paigale justkui oleks nende positsioon määratud nii nagu muude brauserite `fixed` positsiooni puhul. Sellise tehnika puhul on efekt paigalolevast menüüst märksa realistlikum, kui JavaScripti abil saavutatud kergelt hüpleva lahenduse puhul.

4.4 Navigatsioon ja rakendused

Navigatsioon rakendustes koosneb menüüdest, dialoogidest, nuppudest ja on oluliselt mitmepalgelisem, kui veebilehel. Kui rakendus on mingi veebilehe osa, siis on väga soovitatav üldised lehekülje menüüd üldse vaateväljast ära kaotada. Selleks on kaks põhjust:

- **Ruumipuudus:** erinevalt veebilehest, kus reeglina on ruumi, vaevab rakendusi tihti ruumipuudus. Raske on suruda, mingit keerukamat rakendust leheküljele, mille nähtavast osast 30% võtab enda alla lehekülje päis, allesjäänud osast on reserveeritud 20% vasakus tulbas paiknevale menüüle ja teine 20% paremale tulbale.

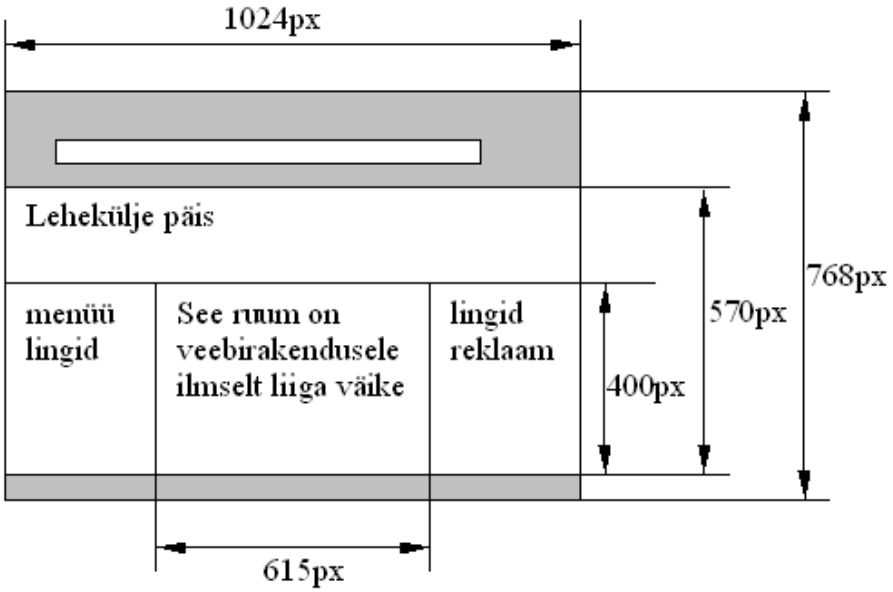
- **Ebaolulise kõrvaldamise vajadus**: rakenduses on hea keskendada kasutaja vajalikele tegevustele. Kõik nupud ja lingid, mis otseselt selle tegevusega seotud ei ole, võivad kasutajat eksitada ja nendele klikkimine võib tühistada tehtud töö. Kui veebilehtedel on domineeriv peamenüü, siis rakendustes peavad domineerima vaid käesoleva tegevusega seotud valikud.

4.4.1 Ruumipuudus

Seoses ruumipuudusega näeme, et eeltoodud põhimõtete järgi liigendatud leht, millest päis ja menüüd võtavad olulise osa, jätab rakendusele 40% lehest, mida on loomulikult vähe. Hetke seisuga jaotuvad kasutajate ekraaniresolutsioonid järgmiselt: 55% kasutab 1024*768, 14% kõrgemat, 25% 800*600 ning 6% kasutab teadmata resolutsiooni (W3Schools 2005). Efektiivne pind on sellest aga veelgi väiksem, kui laius jääb enamvähem kasutusse, siis kõrgusest läheb tegumiriba, menüüde, olekurea ja tiitliriba peale suhteliselt palju kaduma. Minu testide põhjal erinevate brauseritega oli brauseri

kasuliku piirkonna kõrgus ca 200 pikselit väiksem kui tegelik ekraanikõrgus. Portaalilaadse kujunduse säilitamisel jääks seejuures eelnevalt käsitletud vahekordade puhul kasutusse resolutsiooni 800*600 puhul tegelikult vaid pind 480*280 ja resolutsiooni 1024*768 puhul 615*400.

Joonis 7



Veebirakendustes on reeglina kasutuses vormid ja vaated, kus vaade kujutab endast navigatsiooni või mingi tegevuse oleku tulemuse kuvamist, kui vorm on ette nähtud informatsiooni oleku muutmiseks. Siinjuures ei pea veebirakenduses vormiga seostama vaid HTML vormielemente, sest neid võib kasutada ka navigatsioonis ja andmete esituses. Ja samamoodi võib andmete olukorra muutmiseks kasutada elemente, mis pole vormielemendid.

Graafilistes kasutajakeskkondades baseeruvad rakendused akendel, mida jagatakse tavaliselt neljaks (Apple HIG 2005, 172).

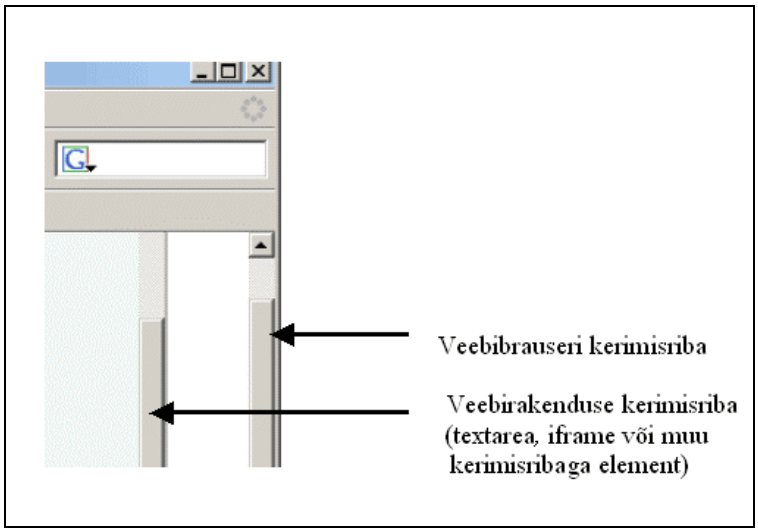
- Dokumendi aken
- Rakenduse aken
- Tööriistade aken

- Teateaken (*alert*)

Apple eristab dokumendiakent tavalisest rakenduse aknast ja mõtleb selle all näiteks tekstiredaktorit, või mingit graafika redigeerimise programmi. Erinevalt traditsioonilisest rakenduse aknast on neis suur osa akna pinnast ühtselt redigeeritav, tihti WYSIWYG piirkond, mis võib olla suurem, kui nähtav osa ja mida saab vastavalt vajadusele kerida või nihutada. Sellise akna tööriistade nupud, menüüd, tööriistaaknad ning dialoogid võtavad vähem ruumi ja seisavad keritava redigeeritava osa suhtes paigal.

Võib tekkida kiusatus liigitada veebirakendused ka selliste rakenduste alla, sest veebileht on ju brauseriaknas keritav, kuid see ei ole enamasti õige. Kuna veebirakenduse kliendipoolse osa platvormiks on brauser, mis peab samal ajal teenima kasutajat ka dokumentide näitajana, siis ei tee see platvormi eripära kõiki veebirakendusi automaatselt dokumendiakendeks. Vaid siis, kui veebirakendus sisaldab põhiosas mingit redaktori funktsionaalsust, võib liigitada teda dokumendiakna tüüpi rakenduseks. Kuid ka siis on sellel redigeeritaval piirkonnal enamasti omad kerimisribad ja brauseri enda kerimisfunktsioon selle kõrval (Pilt 1) tekitab vaid asjatut segadust.

Pilt 1



Seega oleks hea kui kogu veebirakendus mahuks lehele ära ja brauseri kerimisriba ei tekikski.

Tuleb tähele panna, et nii dokumendi akna puhul, kui rakenduse akna puhul, peaks rakenduse aken ise ekraanile ära mahtuma. Kui tõmmata paralleel veebirakendusega, siis peaks ka see mahtuma ekraanile ilma, et seda peaks edasi kerima. Muidugi on siin erandeid, millest ühena võib mainida mingi elektroonilise testi lehte, kus kasutaja saab anda vastused erinevatele küsimustele valikvastuste või muude vastusetüüpide kaudu, kuid on oluline, et tal oleks võimalik liikuda kogu testi või küsitluse piires enne kui ta oma vastused postitab.

Kuid reeglina on keritav rakendus taunitav ja kasutajale tuleb serveerida vaid seda, mis on antud tegevuse seisukohast oluline ja see peaks olema kompaktselt nähtav, mitte ilmnema lehekülje edasi kerides. Üldine tava kasutada kodulehele omast ülesehitust veebirakenduste juures teeb rakenduse ruumi väiksemaks ja leheküljed tahes tahtmata pikemaks ning tekitab suurema vajaduse kerimiseks.

4.4.2 Keskendumine vajalikule

Teine oluline põhjus peale ruumipuuduse, on kasutaja valikute minimeerimine ja olulise esiletõstmine. Selleks on rakendustes välja töötatud erinevad dialoogi tüübid. Apple eristab kolme liiki dialooge (Apple HIG 2005, 201-202):

- Dokumendi modaalne dialoog
- Rakenduse modaalne dialoog
- Mittemodaalne dialoog

Siin esinevad mõisted modaalne ja mittemodaalne väärivad erilist tähelepanu. Modaalsus on dialoogi omadus, mis ei lase kasutajal teha antud rakenduses midagi muud, kui viia lõpule toimingud, mida peab tegema selles dialoogis. Modaalsest dialoogist väljumiseks ei ole tavaliselt muid võimalusi kui *ok* ja *cancel* nupud. Tekstiredaktori MS Word failisalvestamise dialoog on näiteks modaalne dialoog. Seda ei saa kõrvale panna, et dokumendiga edasi tegeleda. Modaalsuse mõte on välistada olukordi, kus mingi kasutaja tegevus võiks põhjustada andmete kadu. Seal kus seda ohtu ei ole, ei soovitata modaalset dialoogi kasutada (Gnome HIG 2.0 2004).

Mittemodaalne dialoog on MS Wordi näidet jätkates menüüst Edit -> Find valikuga avanev otsingudialoog. Selle avades jääb dialoog küll aktiivselt dokumendi ette, minimeerimisnupp dialoogil puudub, kuid samal ajal saame me muuta dokumendi sisu, saame dokumenti kerida ja minimeerida või isegi sulgeda. Dialoogist vabanemiseks on modaaldialoogiga sarnaselt kaks võimalust, sulgeda või sooritada tegevus. Mittemodaalse dialoogi puhul ei pruugi tegevuse sooritamine alati dialoogi sulgemisele viia nagu me näeme otsingudialoogi näitel.

Veebirakenduste puuduseks on platvormi toetuse puudumine nii modaalsuseks kui mittemodaalsuseks. Tõsi Internet Explorer implementeeris juba neljandas versioonis aastal 1997 modaaldialoogi ning viiendas versioonis ka mittemodaalse dialoogi, kuid teistel brauseritel midagi sellist ei ole kui mitte arvestada Netscape 8 brauseri võimalust töötada ka Internet Exploreri renderdusmootoril. Arvestades brauserite turuosa praegust jagunemist (W3Schools 2005), võiks julgelt nendele dialoogidele ka rakenduse üles ehitada ja kasutada muude brauserite puhul `window.open` meetodil avatud aknaid. Kuid tänapäeval ei saa enam ei akende ega modaaldialoogide peale kindel olla, sest neid on võimalik blokeerida brauseri vahendite või kolmandate tootjate programmide abil, ning selliseid blokeerijaid tõesti ka kasutatakse. Netscape 8 aga paistab silma sellega, et `window.open` meetodil avatud uus aken ei avane akna määratletud mõõtudes akna vaid uue *tab*-ina. Seetõttu tuleb modaalsuse ja mittemodaalsuse saavutamiseks kasutada mitmeid kõrvalteid. Lähemalt leiavad need probleemid käsitlemist peatükis 6.2.

Modaalsuse põhimõtted on risti vastupidised navigatsiooni säilitamise põhimõtetega. Kui infole orienteeritud keskkonnas peab tagama inimesele igalt poolt juurdepääsu ükskõik kuhu, siis tegevusele, ülesannete täitmisele suunatud keskkonnas, peab hoolitsema selle eest, et ülesande täitmine ei oleks häiritud, et kasutajal poleks mingit võimalust oma tegevust kogemata katkestada, vaid dialoogist väljumiseks oleks vaid kaks võimalust, ühemõtteline tegevuse katkestamine või selle sooritamine.

Kui pakkuda rakenduse vorme lehekülje sees (mis on tegelikult veebirakenduste puhul üsna sage), kus säilib navigatsioon, siis jääb kasutajale alati võimalus poole tegevuse pealt linke mööda edasi navigeerida arvates, et tagasi pöördudes on tema pooleliolev töö teda ootamas. Navigatsiooni säilimine tekitab võimaluse, et kasutaja käsitleb seda

rakenduse sisese menüüna nagu näiteks Microsoft Wordi menüüriba (File, Edit, View jne), mille valikuid ja sealt avanevaid dialooge me võime avada, säilitades seejuures oma tehtava töö. Segadust tekitab seegi, et lehekülje navigatsiooni peamenüüd on reeglina kõige enam esile tõstetud (Pilt 8), kuid ei ole mingil moel seotud käsiloleva tegevusega.

Joonis 8: Elioni veebimail

Küsimus on selles, kas tulu, mis saadakse sellest, et pakutakse veebirakendust lehekülje sees ja lubades kasutajal otse rakendusest navigeerida mujale, kas sisule või rakendustele orienteeritud osadesse, on suurem, kui kahju, mis kasutaja eksitamisega võib sündida. Selle tulude ja kahjude arvestuse juurde tuleb kindlasti kaasata ka ruumipuudus, mida menüüde ja lehekülje päise säilitamine tekitab ja täiendav laadimisaeg, mis selle esitamiseks kulub.

Ainus tõsiseltvõetav argument võiks olla lehekülje üldise ilme säilitamine. Kuid selleks piisaks väga kitsast päiseribast ja vähendatud asutuse või organisatsiooni logost kuskil nurgas, mis jätaksid suurema osa kasulikust pinnast rakendusele ja minimeeriksid valede valikute arvu.

4.5 Navigatsiooni harjumuspärasus veebilehtedes

Veebilehtede navigatsiooni eeskujud osaliselt pärit trükimeediast, kuid need on arenenud iseseisvalt ja välja kujunenud suuremate infoportaalide eeskujul. Harjumuspärasus sisaldab endas eelkõige seda, kus navigatsioonimenüüd paiknevad ning kuidas on väljendatud nende hierarhia. Põhiliselt otsitakse peamenüüd horisontaalsest piirkonnast lehekülje päises all või vasakust vertikaaltulbast. Paremalt vasakule kirjutavate rahvaste puhul on peamenüüd tihti paremal vertikaaltulbas.

Peale asukoha oodatakse teatud tagasisidet selle kohta, kus kasutaja parasjagu asub ning tähiseid selle kohta, kus kasutaja on juba käinud. Kui aadressirida on arvutile orienteerumiseks, siis peab lehekülje disain tagama võimaluse ka visuaalselt oma asukohta lehe piires positsioneerida.

Viimase omaduse realiseerimise lihtsaim võimalus muuta külastatud linkide värvi, milline omadus on brauseritel ka vaikimisi defineeritud. Teiseks tihti rakendatavaks tehnikaks on peamenüü käesoleva valiku esiletõstmine ning komandaks tehnikaks leivakoorukesteks (*breadcrumbs*) kutsutava teekonna kuvamine, mis väljendab asukohta veebilehestiku üldises struktuuris.

4.6 Navigatsiooni harjumuspärasus veebirakendustes

Veebirakendustes tegelikult navigatsiooni polegi. Tinglikult võib nimetada navigatsiooniks (ja seda tehakse ka praktikas) mingi suurema rakenduse vaateid, mille kaudu tagatakse ligipääs erinevat funktsionaalsust pakkuvatele rakenduse osadele. Rakenduste tegelik navigatsioon seisneb aga interaktsioonides, mis kasutaja tegevust toetavad. Need on defineeritud interkatsiooni stiiliga, milleks võivad olla:

- käsurea keeled,
- ja/ei tüüpi dialoogikeskkonnad,
- funktsiooniklahvidega juhitavad keskkonnad,
- vormide täitmise keskkonnad,
- menüüvaliku keskkonnad
- otsese manipuleerimise keskkonnad

- või mitmed tulevikustiilid

(Nielsen 1993, 69).

Peene kliendiga veebirakenduse stiil on võrreldav seguga vormide täitmisest ja käsurea keelest. Vormielement `select`, lisab tükikese menüüvaliku keskkonnast, kuid sellega asi piirdubki. Teatud ülesannete puhul on vormide stiil piisav, kuid kasutajaharjumusi mõjutavad tänapäeval otsese manipuleerimise keskkonnad. Otsese manipuleerimise mõiste võttis kasutusele Ben Shneiderman (1982) ja ning omal ajal peeti selleks paljusid graafilise keskkonna ja akendega programme, mida sai manipuleerida hiirega nuppudele klikkides (Nielsen 1993, 61), kuid mis on tänapäeval pigem vormielementide lahutamatu omadus. Praegu seostatakse otsese manipuleerimisega põhiliselt lohistamisetehnikas (Drag & Drop) realiseeritud interaktsioonistiile (Apple HIG, 42). Mõned autorid (Gordon 2003, 86) liigitavad otsese manipuleerimise alla ka WYSIWYG keskkonna, teised (Foley 1997, 398) käsitlevad seda eriliigina. Põhimõte on mõlemal siiski üks: tagada kohene tagasiside ja vahetu kontroll toimuva üle.

Tänapäeva rakendused kujutavad reeglina segu kõikidest loetletud interaktsioonistiilidest ja see kujundab ka ootused, mis rakendustele seatakse. Ka paraja kliendi rakendused võimaldavad teostada kõiki loetletud interaktsiooniliike ja sellega vastata paremini kasutaja ootustele.

4.7 Pikad vs lühikesed lehed veebilehtedes

Varasemad uuringud (Nielsen 1997) näitasid, et 90% inimestest ei külastanud linke, mis jäid lehekülje nähtavast osast välja. Alguses järeldati sellest, et inimesed ei armasta kerida, kuid hiljem leiti, et inimesed ei tulnud veel selle peale, et veebilehti saab kerida. Veebilehti vaadeldi kui dialooge, kus kõik vajalik on asetatud nähtavale. Kahjuks levis see esialgne järeldus, et inimesed ei armasta kerida ja kajastus ka mitmete lugemiseks mõeldud veebidokumentide struktureerimises. Kalli modemiühenduse aegadel eelistasin alati pikemat elektroonilist manuaali, mille laadimisaeg oli küll vastavalt pikem, kuid selle lugemise ajaks võis ühenduse katkestada. Samuti tundusid väga mugavad dokumendisisesed lingid, mis erinevalt veebi linkidest toimisid momentaalselt. Viimane omadus on väärtuslik ka nüüd, kui (vähemalt käesoleva töö autori jaoks) aeglane ja kallis ühendus on minevik.

Ootamine on ebameeldiv (Morkes et Nielsen, 1997). See on kindlasti üks kasutusmugavust puudutavatest väidetest, millega tuleb nõustuda praktiliselt igas olukorras. Meile meeldib, kui meie tegevust ei katkestata ja kui meie tegevusele reageeritakse koheselt. Muidugi veebilehtedel ja rakendustel on selles osas erinevad nõuded ja on suur vahe kas oodata paar sekundit mingi lehekülje ilmumist või oodata sama kaua, ilma vähimagi tagasisideta mingi nupuvajutuse tulemust.

Kuid see üleüldine konsensus ootamise ebameeldivusest, ei sobi kokku väitega, et inimesed ei armasta kerida ning ka mitte väitega lugemisstiili kohta, et inimesed ei loe vaid skaneerivad lehekülgi (Nielsen, 1997), kus skaneerimise all peetakse silmas seda, et inimesed, selle asemel, et kogu teksti lugeda, nopivad sealt üksikuid fraase. Kui kõik need väited kokku panna, saame me sellise vastuolulise olukorra, kus kasutaja rahulolu on igatpidi välistatud.

Kuna kasutajale meeldivad lühikesed lehed ja ta loeb neid väga kiiresti, siis peab ta kogu aeg laadima uusi lehekülgi, mis on omakorda ebameeldiv, sest sunnib ootama. Vastuväiteks võib öelda, et pika lehekülje laadimine sunnib küll üks kord ootama kuid siis vastavalt rohkem. Tegelikult ei ole see päris tõsi.
Kuna freimide kasutamine oli juba eelnevalt taunitud, siis tähendab see seda, et iga uue lehekülje laadimiseks tuleb laadida ka kogu lehekülje kujundus ja menüüd. Piiritledes sisu osa ühele lehele mahtuva mahuga ja arvestades, et mugavaks lugemiseks ei tohiks selle laius olla suurem kui 600 pikselit, siis saame enamusel juhtudest tulemuseks, et iga laadimise korral laetakse palju vähemal määral sisu kui vormi.

Juhul kui menüüd ja kujundus on tõesti väga vähenõudlikud, oletame, et sisu ja esitus on väga hästi lahutatud, kujunduses on kasutaja arvuti mälus või kõvaketta puhvris talletunud elemente nagu CSS ja pildid või JS failid, ja tegeliku vormi ja menüüde jaoks kuluv koodimaht osutub tõesti väiksemaks sisust, siis erilist vahet ei tohiks olla.

Pikkadel lehekülgedel on oma koht kindlasti teatud liiki info vahendamiseks ja järjest enam lisandub kasutajaid, kes mitte üksnes ei skaneeri lehti vaid võivad ka süvenenult lugeda pikki artikleid. Minu subjektiivne arvamuse põhjal, millel tean olevat palju toetajaid, on hästi vormistatud HTML lehte elektrooniliselt palju kergem lugeda kui

sama kirjastiili ja vormistusega PDF formaadis faili või Zinio lugeja failiformaati, mida kasutab ajakiri Science oma elektroonilise versiooni jaoks.

4.8 Pikad vs lühikesed lehed veebirakendustes

Lokaalses arvutis töötavate rakenduste dialoogid on kiired ja sisaldavad harva dialooge, mida peaks kerima. Erandiks on siin sellised dialoogi osad, milles inimene kujundab mingit suur pilti või teksti, mis ei saagi ekraanile ära mahtuda. Kuid kui tegu on näiteks suure hulga sisestusväljadega, mis tuleb täita, siis rakenduse loogiline lahendus oleks need grupeerida ja esitada eri dialoogides. Veebirakendused on sellisel juhul raskes olukorras kui iga uus dialoog tähendaks ebameeldivalt pikka laadimisaega ning uue vormi laadimise ajal tuleb salvestada eelmisest vormist saadetud andmed.

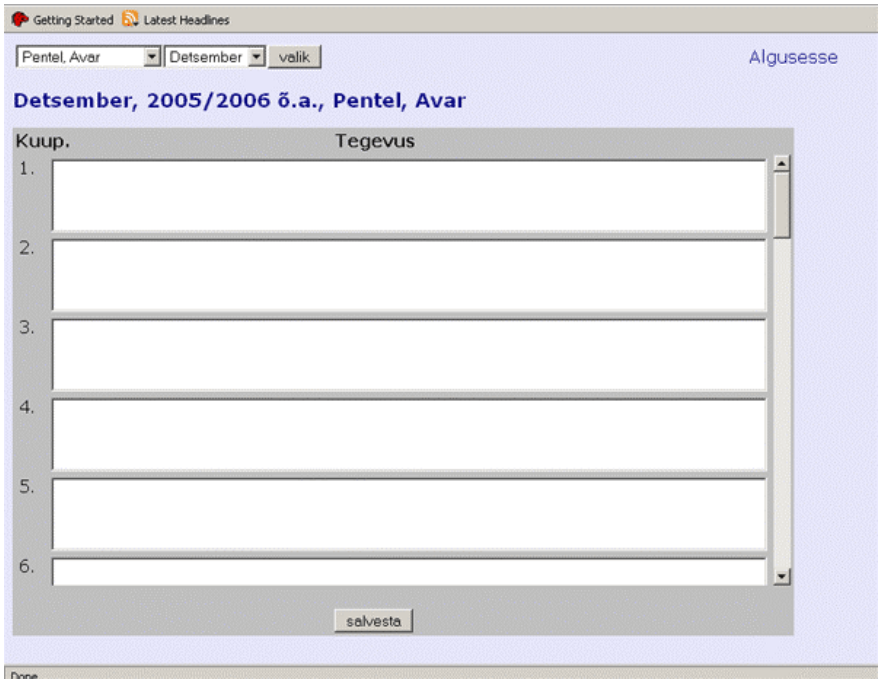
Kui vormi tahetakse hoida mingi lehe üldise kujunduse sees, siis suureneb laadimisaeg veelgi. Kokkuvõttes tekitab see kasutajale palju harjumatuid katkestusi. Kui tegu on aeglase ühendusega, siis võib selline katkestus muutuda eriti häirivaks. Aga rakendus on mingi ülesande sooritamiseks. Rõhk on sellel ülesandel, rakendus ise peaks jääma nähtamatuks. Rakendus, mis kasutaja tegevust kogu aeg katkestab, ei saa jääda nähtamatuks, see võib sobida mingi vormi jaoks, mille täitmine on kohustuslik, st. ebameeldivuste talumine on tagatud administratiivsete vahenditega, kuid kahtlemata ei saa see sobida loova töö tegemiseks.

Kuidas meeldiks kirjanikule kirjutamine pastakaga, mis iga natukese aja tagant, lõpetab töö, näiteks iga lause tagant 10 sekundiks, või iga sõna või tähe tagant. Me ei saaks sellise pastakaga töötada, meie tegevuse loomulik rütm on häiritud, me unustame üldse ära, mis täht, sõna või lause pidi tulema järgmisena.

Seetõttu tuleks vormid mõistliku kogumina klienti laadida ja esitada etapiviisiliselt. Oma olemuselt oleks tegu pika lehega, sest vormielemente oleks rohkem kui ühele ekraanitäiele mahub, kuid reaalselt esitatakse kliendile vorme grupeeritult. Lihtsam on oodata üks kord natuke kauem rakenduse avanemist, kui kannatada pidevaid katkestusi töö ajal.

Kui rakendus peab olema kättesaadav ka ilma JavaScriptita klientidele, siis oleks mõistlik rakendada pika lehe loogikat nagu järgmisel joonisel

Pilt 2: Salvestusprobleemide tõttu ümberdisainitud vorm Toila Gümnaasiumi ÕHSis



Nagu näha on seal tegu rakenduse sisese kerimisribaga ja brauseriakna kerimisriba ei tekigi. Näites toodud vorm oli alguses vormistatud traditsioonilise pika lehena, mida tuli brauseri kerimisribast kerida, kuid mitmetel kasutajatel tekkis probleeme salvestamisega. Peamise põhjusena kahtlustasin lõpus asuva salvestamisnupuni mittejõudmist ja paigutasin ühe nupu ka ülesse. See tõepoolest vähendas probleeme, kuid mitte kõigi kasutajatega.

Seejärel märkasin et probleemid ilmnesid ajal mil oldi tõenäoliselt nende lahtrite täitmise juures, mis jätavad kasutaja vaateväljast välja ülemise salvestamisnupu, kuid alumine pole veel vaatevälja jõudnud. Mitme töökohaarvuti töölaualt leitud HTML fail selle sama lehe salvestusega viitas sellele, et ka brauseri File->Save As dialoogi üritati kasutada. Peale seda, kui olin pika lehe kujundanud eelpool toodud ekraanipildi järgi, enam salvestamisprobleeme ei esinenud.

4.9 Ühilduvus ja kättesaadavus veebilehtede puhul

Ühilduvuse all mõeldakse seda, et lehekülg oleks võimalikult paljude brauseritega nende eri konfiguratsioonides kättesaadav. See puudutab näiteks lehekülje kujunduse selliseid aspekte nagu mitte lootmajäämist piltide peale, sest kasutaja võib olla tekstibrauseriga, või on piltide näitamise ära keelanud või on tegu inimesega kes üldse ei näe ja kasutab veebi lugemiseks assisteerivaid tehnoloogiaid. Sama masinloetavuse aspekt puudutab ka kättesaadavust otsingurobotitele.

Piltidele tuleb luua alternatiivsed tekstid, mis kirjeldavad pildid kujutatud. Kõik pildid, mis teenivad kujunduslikke eesmärke, tuleb lisada CSS `background` omadust kasutades. Leht peab olema nähtav, ka siis kui kasutaja brauser ei toeta CSSi, tuleb jälgida, et lehekülje loogiline ülesehitus säiliks ka siis, kui CSSi toetus puudub.

Tuleb kindlustada, et värvid oleks nähtavad ka madalama värvisügavusega, kindlustada, et lehekülg mahuks ka väiksema ekraanilahutusega ekraanidele. Tuleb jälgida, et lehekülje kasutamine oleks mugav ka ilma hiireta. Tuleb arvestada, et laadimisaeg oleks normaalne ka modemiühenduse korral.

Neid tingimusi võib lõpmatuseni jätkata. Küsimus on selles, kes on meie kasutajad. Ühest küljest võime väita, et meie arvutid on edasi arenenud, värvisügavused, ekraaniresolutsioonid on suurenenud, ühendused on kiiremad ja tekstibrausereid ei kasuta enam keegi. Kuid isegi kui see nii oleks, on tehnoloogia areng toonud meile uued seadmeid, PDAd mobiiltelefonid ja muud seadmed, mille piiratus ekraani, ühenduse ja muude aspektide osas, viib meid mõnes mõttes veel kaugemale tagasi kui internet oli 10 aasta eest. Isegi GPRSi ja GSM kaudu kaasaegset laptopi kasutavad inimesed keeravad oma brauseril kinni kõik, mis võimalik kuni piltideni välja, et vähendada allalaaditava ja maksulise info mahtu. Loomulikult on need kasutajad üsna vihased, kui neile hetkel hädavajalik infoteenus on kasutav ainult brauseris, mis näitab pilte ja kui info nägemiseks peaks veel alla laadima Flash Playeri ja hulgaliselt reklaambännereid. Siis tõenäoliselt otsitakse muid alternatiive.

Seega tähendab ühilduvus tänapäeval väga mitmekihilist lähenemist. Meie kiired ühendused suured ekraanid ja võimalusterohked brauserid nõuavad, et nende

võimalused oleks maksimaalselt meie huvides realiseeritud. Me ei lepiks sellega, kui kogu veeb oleks optimeeritud, mingi mobiiltelefoni järgi. Kuid samas me tahame sedasama veebi kasutada ka selle mobiiltelefoni abil

4.10 Ühilduvus ja kättesaadavus veebirakenduste puhul

Kui juba veebilehekülgede ühilduvuse puhul me mõistame, et kogu veebi ei saa optimeerida mobiiltelefoni jaoks, siis seda selgem on see rakenduste puhul. Veebirakendused pole kunagi nii laialdase kasutajaskonnaga, kui leheküljed. Muidugi on siin eranditeks suured veebimaili teenusepakkujad nagu Hotmail, Yahoo ja Gmail, kui need ongi ehk parimaks näiteks, kuidas ühilduvus tuuakse ohvriks funktsionaalsusele ja mugavusele. Siiski on nende teenuste puhul järgitud põhimõtet, et kellel on vähem see saab ka vähem, mitte ei jää päris ilma. Näiteks võib tuua, et kõik nimetatud emailiteenuse pakkujatest toetavad sõnumi reaalkuva redigeerimist (WYSIWYG), mis ei tähenda, et seda peaks emailivahetuse puhul propageerima.

Gmail paistab siin silma eriti innovatiivsete lahendustega, mis pakuvad näiteks reaalajas töötavat õigekirjakontrolli, kus automaatselt pakutakse oletusi õigete sõnavariantide kohta sõna kohal avanevas kontekstmenüüs. Ja seda muidugi mitmes keeles, kaasaarvatud Eesti keeles. Samuti on liikumine erinevata dialoogide vahel kiire ja ei toimu uue lehe laadimise tulemusel.

Kuid kui antud teenuste puhul pakuti ka nn vaese kasutaja varianti, siis Google poolt arendatav teenus Google Map on hea näide sellest, et isegi teatud teenuse puhul võib kasutajaskonda julgelt piirata. Kui ei ole JavaScript sisselülitatud, ei näe ka kaarte. Samuti ei saa teenust kasutada liiga vananenud brauseriversiooni korral.

Loogika peaks olema arusaadav. Keegi ei oota, et Regio Atlas (Regio Eesti 2005) töötaks 286 arvuti MSDos 6.22 operatsioonisüsteemil. Keegi ei pane pahaks, et Regio pole teinud selle jaoks miinimumversiooni. Selle asemel oleme harjunud lugema erineva tarkvara pakendilt minimaalseid nõudeid riistvarale ja operatsioonisüsteemile.
Nüüd, kui mitmed teenused ja rakendused on liikunud veebi, siis oleks sama alusetu oodata, et minimaalset funktsionaalsust toetav brauser võiks olla piisav platvorm rakendusele, mille eesmärk on asendada mingit lauaarvuti rakendust. Brauser on

veebirakenduste arendusplatvorm. Ja see keskkond, mis veebipõhises klient server süsteemis vaatab vastu kasutajale, on kirjutatud just selle arendusplatvormi jaoks. Kui kliendi keskkond optimeerida tööks minimaalsetes tingimustes, siis ei saa sellega eriti midagi teha, vähemalt pole see tegevus mugavam, kui näiteks proovida mingis joonistusprogrammis tegutseda üksnes klaviatuuri abil.

Kasutaja saab tavaliselt aru kui veebirakenduse kasutamiseks on brauserile esitatud teatud nõuded. Kui seejuures pakub rakendus ka mingit mitmekihilist lähenemist, mis pakub funktsionaalsust vastavalt sellele, mida kasutaja keskkond võimaldab, siis on tegu lausa suurepärase rakendusega, millist lähenemist me reeglina ei kohta desktop rakenduste hulgas.

Tihti on nõuded, et rakendus peab töötama ka nii iga mobiiltelefoni või LYNX brauseriga, kantud pigem religioossest fanatismist, kui tervest mõistusest. Püüdes kõigile sobida, on meil oht langetada rakenduse tase madalaimale ühisele nimetajale, mis lõpuks kedagi ei rahulda. See toob meelde loo rabi Hilleli kahest naisest, kellest üks oli noor ja teine vana. Noor häbenedes oma mehe halle juukseid kiskus neid salaja öösel välja, kuid vana naine, kes kadestas mehe musti juukseid, kiskus musti karvu välja. Sellepärast jäi rabi Hillel varakult kiilaks.

5 Hinnang kliendipoolsetele tehnoloogiatele

Käesolevas alapeatükis puudutan ühte olemuslikku probleemi antud uurimuse seisukohalt. Küsimus on selles, et kasutaja poolt soovitavat lõpptulemust saab saavutada eri tehnoloogiaid kasutades. Näiteks võib kogu rakenduse kirjutada Java appletitena samuti saab kasutada mitmeid lisasid (*plugin*). Tulemus on mitmes mõttes arendaja poolt vaadatuna kindlam, sest ükskord ettevalmistatud programmi stsenaariumi ei pea kohandama eri brauserite iseärasustele.

Teisest küljest, lisade kasutamine juhul, kui valdava osa brauserite puhul saab saavutada sama tulemuse ka brauserile sisseehitatud skriptimistehnoloogiate abil, sarnaneb autole lisavarustuse soetamisega, mis on juba standardvarustuses olemas. Ressurss, mis on kasutajal niikuinii internetibrauseri näol kasutuses, jääb siis enamuses kasutamata ja on seega raisatud. Argument lisade kasutamise poolt saab siis olla vaid arenduse odavam hind.

Ka platvormist sõltumatusest rääkides on lisade ja brauseri vahel kvalitatiivne vahe. Brauseri sisemistele võimalustele rajatud tehnoloogia on tõesti praktiliselt platvormist sõltumatu, sest platvormiks ongi siis brauser ise ja tuleb vaid eeldada selle olemasolu. Lisade puhul on vaja siiski tõmmata alla ja installeerida selle lisa vastav mahamängija (*player*), näitaja (*viewer*) või virtuaalmasin (*virtual machine*). Tänapäeval pole sugugi tavaline Java virtuaalmasina automaatne installeerimine koos brauseriga. Ka Macromedia Flashi lisa tuleb installeerida eraldi. Vabanduseks pole ka see, et virtuaalmasina ja Flashi lisa saab kasutaja enda arvutisse tasuta laadida. Selleks ei pruugi kasutajal olla oskusi ega õigusi. Seetõttu kasutab valdav enamus veebirakendustest kliendipoolse intelligentse tehnoloogiana JavaScripti (Baxley 2002, 28).

Brauserid on arendanud välja väga rikkaks keskkonnaks ja kolmandate tootjate lisade kasutamine pole enamusel juhtudel enam vajalik. Näiteks Google kaarditeenuse (Google Maps) puhul saab kogu maailma kaarti ja satelliitfotot nihutada ja suurendada või vähendada hiirega, seejuures lehte kordagi uuesti laadimata. Selleks pole kasutataud ühtegi lisa ehk pluginat, kõik on saavutatud standardseid tehnoloogiaid

kasutades. Google mailiteenus (Google Mail) baseerub samuti puhtalt kliendipoolsel JavaScriptil ning pakub samasugust lehel olemise kogemust, kus üksik nupuvajutus, ei jäta kasutajat maha ega põhjusta uue lehe laadimist ometi struktureerub keskkond ringi vastavalt kasutaja valikutele, kirju saab redigeerida WYSIWYG vaates ning manuste lisamisel luuakse uusi üleslaadimisvälju dünaamiliselt juurde Internet Exploreril neid kasutajale isegi mitte näidates. Väga hästi on lahendatud sisestatud mailiaadresside meeldejätmine, mis esimese tähe sisestamise peale varemsisestatud valikutsest menüü tekitab. Ja muidugi on mailikliendil õigekeelsuskontroll ja seda lugematul hulgal keeltes, kaasa arvatud Eesti keeles.

Teenused nagu Google kaarditeenus naing rakendused nagu Google mailiteenus on näidanud ka suuremale auditooriumile, et brauseri poolt interpreteeritav JavaScript on märksa võimsam vahend, kui temast tavaliselt on arvatud.

Järgnevalt analüüsime brauserite lähiajalugu, et teha sellest järeldusi praeguse olukorra ja tuleviku kohta.

5.1 Brauserite evolutsioon platvormiks

Valides olemasolevate tehnoloogiate vahel, on üheks oluliseks valikukriteeriumiks hinnang tehnoloogia küpsusele ning perspektiivile. Viimane on küll tarkvaraarenduses aina väiksema tähtsusega, eriti kui arendada keskselt hallatavaid paraja kliendi süsteeme. Perspektiivi tuleb küll hinnata, aga see ei saa olla mitmetes aastates mõõdetav. Pigem tuleb mõelda sellele, et süsteem oleks lihtsalt muudetav ja edasiarendatav, kui selleks vajadus või võimalus tekib. Kuid küpsuse ja praktilisuse hindamine on kindlasti omal kohal.

Iga tehnoloogilise toote või leiutise areng sisaldab endas sarnaseid tunnusjooni. Uue leiutise sünni juures oskab näha selle perspektiivi vaid leiutaja. Näiteks võib tuua Karl Drais von Sauerbonni[4] jalgratta leiutamise aastal 1817. See oli kaherattaline kohmakas

[4] Ka enne von Sauerbonni, kellele mitmetes allikates on viidatud kui von Drais, on analoogseid leiutisi dokumenteeritud, viidatud on isegi Leonardo da Vincile (McGurn 1987, 12). Põhiliselt on jalgratta

jooksumasin, mida kutsuti hobihobuseks (Lindstedt, Burenius 2003). Alguses ei peetud seda kasulikuks leiutiseks, pigem mingiks viguriks, mis sobiks tsirkusesse. Kahtlemata oli sellel jalgrattal veel palju puudusi, puudus korralik juhtimismehhanism, ei olnud pedaale ja ratast tuli edasi tõugata jalgadega. Kuid isegi märksa täiuslikuma mudeliga väljatulemisel oleks olnud raske inimestele selgeks teha, et ka kahel rattal on võimalik püsida.

Kui meenutada arvuti hiire ajalugu, siis ka sellel ei nähtud alguses mingit kasulikku rakendust. Ja uuemast ajast meenub võibolla paljudele, et valik mustvalge ja värvilise monitori vahel ei olnud sugugi nii enesestmõistetav. Teatud rolli mängis värvimonitori kallim hind, kuid samas oli suur kaal ka nende kasutajate argumentidel, kes leidsid, et ilu pärast pole neil värve vaja ja kõik vajaliku saavad nad tehtud ka ilma värvideta.

Tõmmates paralleeli brauseritega, siis veeb on üks tehnoloogilise evolutsiooni eranditest selles mõttes, et see võeti suhteliselt kiiresti omaks. Kuid kui vaadata edasisi arenguid, siis sarnanevad need iga teise tehnoloogilise toote, olgu selleks näiteks jalgratas või lennuk, arengule. Võibolla mainitud lennukist ja jalgrattast eristab brauseri arengut veel see, et kui esimeste arendajatel oli algusest peale selge, mida nad tahavad saavutada ja põhimõtteliselt selleks need tooted ka arenesid, siis vaevalt Tim Berners-Lee oskas ette näha seda, et esialgsest elektroonsete dokumentide vaatamise vahendist võiks saada tarkvara arendamise platvorm. Sellist suurt sisulist muutust on isegi raske märgata ja vahest on sellega raske leppida. Veebi puhul on tegu nagu lapsega, kes on suureks saanud ja tal tuleb lasta minna oma teed, ükskõik kui väga me teda ei tahaks enda juures hoida.

Kui üks tehnoloogia on justkui teise traditsioonides sündinud, siis löövad need vanema-lapse suhted välja. Kunagi oli nii televisiooni sünni juures, et esimesed telesaated kopeerisid justkui raadiosaateid. Kogu rõhk oli sõnalisel osal, helidel ja muusikal ning visuaalset poolt ei osatud kasutada. Veebil, mis oli algselt dokumentide kogumikuks mõeldud, on selles mõttes suur side tüpograafiaga ja paljud veebiväljaanded kopeerivad paberkandjale mõeldud dokumente.

---

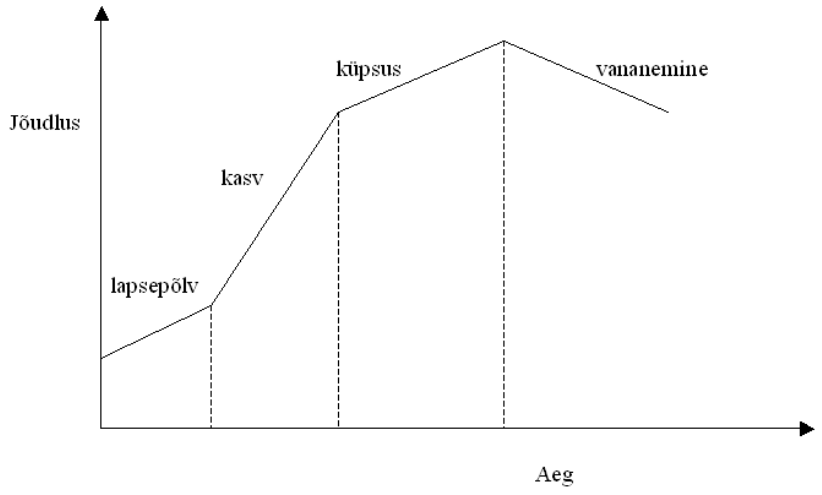
leiutajana mainitud siiski Draisi, kes oma nime jäädvustas oma aja kohta praktilisema leiutise, dresiini kaudu.

Seaduspärasused, mis iseloomustavad tehnoloogilise innovatsiooni arengut on Genrich Altshulleri poolt loodud TRIZ[5] teooria kohaselt sarnased ja ennustatavad. Iga tehnoloogia areng läbib järgmised kindlad staadiumid:

- sünd
- lapsepõlv
- kasv
- küpsus
- vananemine

Neile arenguetappidele on omased sarnased näitajad nii tehnoloogia üldise jõudluse (Joonis 9), innovatiivsuse taseme (Joonis 12), üksikute leiutiste hulga (Joonis 10) kui ka praktilisuse ja kasumlikkuse (Joonis 13) poolest (Yang, El-Haik 2003, 249). Üldist jõudlust iseloomustab S-kõveraks (*S-curve*) nimetatud graafik, mis oma nime on saanud sellest, et väljendades x teljel aja kulgu läbi nimetatud arenguetappide ning y-teljel süsteemi üldist jõudlust, siis moodustuv graafik meenutab ettepoole kallutatud S tähte (nagu normaaljaotuse vasak pool).

Joonis 9



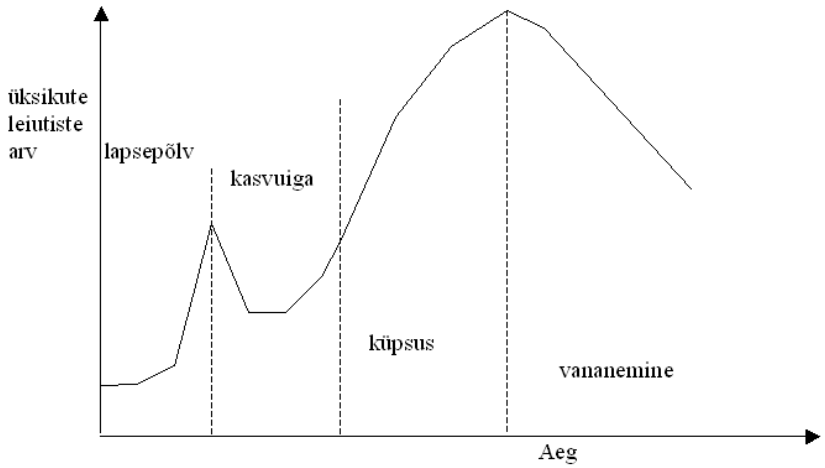
Süsteemi jõudluse areng toimub teatud S-kõverat järgides

[5] Vahest refereeritud ka kui TIPS - Theory of Inventive Problem Solving. TRIZ on algupärase venekeelse lühendi transkriptsioon: ТРИЗ – Теория Решения Изобретательских Задач

Suur kõver moodustub omakorda paljudest väiksematest S-kõveratest, mis iseloomustavad väiksemate uuenduste arenguid (Betz 2003, 151).

Paigutades sellele graafikule näiteks vendade Wrightide lennuki arendamise, siis sünnile ja lapsepõlvele eelnes pöördelise kinnitus sellest, et õhust raskem sõiduk saab lennata. Alguses kulus põhiline energia juhtimissüsteemi täiustamisele ning arengut lennumasina kiiruses eriti ei toimunud. Jalgratta puhul kehtisid samad arengud, juhtimissüsteemi täiustamine, pedaalide kasutuselevõtt. Tehnoloogia lapsepõlves leiab aset hulk väikeseid leiutisi. Brauseri puhul kulusid algusaastad HTTP protokolli ja HTML keele täiustamisele, mis lõi aluse edasiseks kiiremaks kasvueaks.

Joonis 10



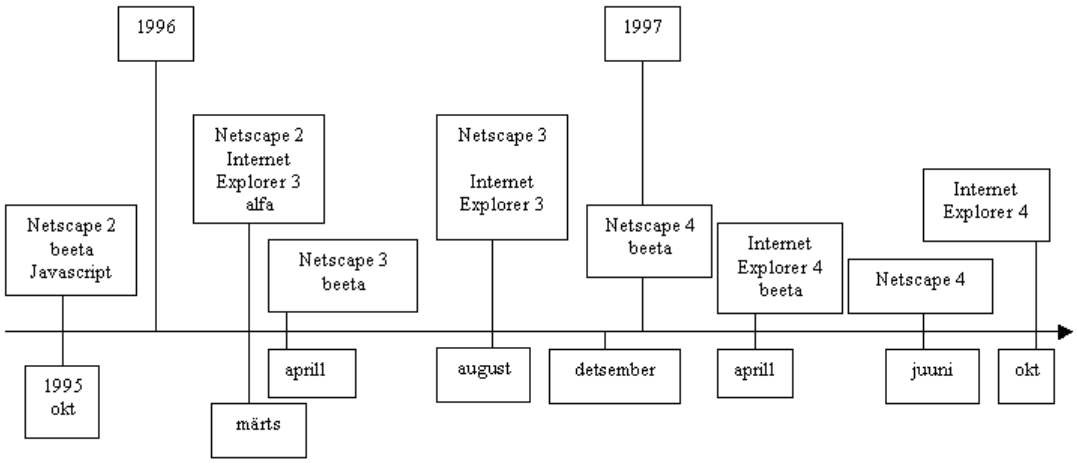
Üksikute leiutiste arv tehnoloogia arenguetappides

Üksikute leiutiste arv kasvab kiiresti kuni tehnoloogia lapsea lõpuni, sellega on tehnoloogia arenenud staadiumisse, kus vajalike ressursside olemasolul on juba kiirem kasvuea areng võimalik. Vendade Wrightide puhul oli selleks lapsepõlve lõpu rohkete pisileiutiste ja järgneva üldise jõudluse arendamise ressursiks alanud I Maailmasõda ja sõjatööstuse vahendid. Brauserite puhul dateeriksin lapsepõlve lõpu rohkete leiutiste lisandumise ajavahemikku 1995-1997, kus aasta 1997 lõpp või 1998 tähistaks kasvuperioodi algust. Nagu näha, jääb ka see periood kuulsa brauserisõja aegadesse, mis sundisid konkurente Netscape-i ja Microsofti oma brauseritesse rohkem investeerima.

1995 aastal tuli Brendan Eich Netscape tööle ja arendas seal välja brauseri skriptimiskeele LiveScript, mida esitleti Netscape 2 brauseri beetaversioonis 1995 aasta lõpus. Varsti ristiti uus keel ümber JavaScriptiks kasutades ära tolle aja moesõna Java (Eich 1998). Netscape 2 paistis ka muidu silma mitmete uuendustega ja lõi juba üsna reaalse võimaluse mõelda brauserist kui platvormist. Nagu Eich hiljem on öelnud (Wayner 2005) nägid nemad koos Marc Andreesseniga[6] JavaScriptis alati arendajatele pakutavat võimalust luua kliendikeskseid veebirakendusi, mis ei peaks kogu aeg uusi lehekülgi laadima.

Analüüsides üksikute tehnoloogiliste uuenduste arvu perioodil 1995-1997, siis oli arengutempo tõesti hämmastav, selle iseloomustamiseks toon järgnevalt kaheaastast perioodi iseloomustava ajatelje (Joonis 11) alates oktoobrist 1995 kuni oktoobrini 1997.

Joonis 11



See periood iseloomustab kiiret JavaScripti ja dokumendi objektimudeli[7] arengut. Eriti kiire oli areng Internet Exploreril, mille teist versiooni ma ajateljele ei märkinudki,

[6] Marc Andreessen (s. 1971) on enim tuntud Mosaic brauseri kaasautorina ja Netscape Communications korporatsiooni ühe asutajana.

[7] Kasutan sõna dokumendi objektimudel tinglikult brauserisispetsiifilise süsteemina, sest W3C kinnitatud DOM sel ajal veel puudus.

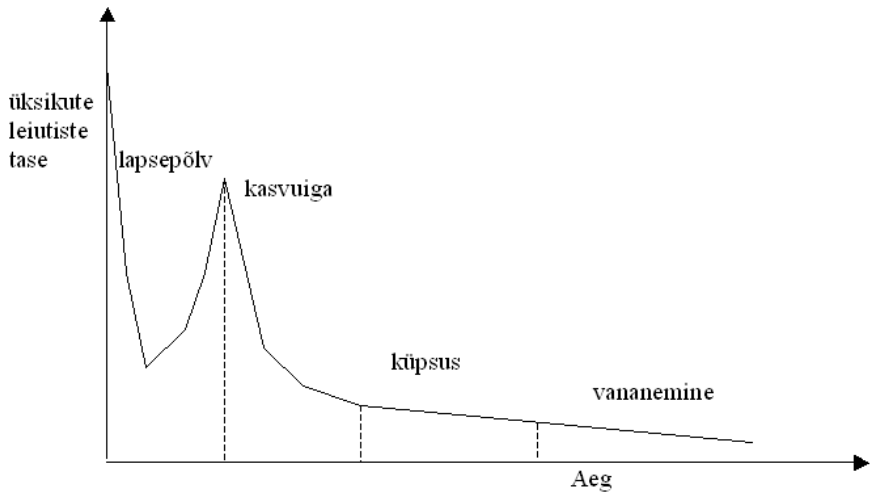
kuna see polnud oma omadustelt ligilähedanegi Netscape 2 brauserile. Internet Explorer 2 beeta anti välja samuti oktoobris 1995 ning lõppversioon sama aasta novembris. Kuid vaatamata Internet Exploreri väikse venna staatusele brauserisõja alguses, jõudis ta neljandas versioonis tasemele, mis võimaldas teha juba üsna palju sellest, mida mitmed kaasaegsed veebirakendused alles nüüd avastavad.

Üheks vähemtuntud võimaluseks Internet Explorer 4 juures oli näiteks WYSIWYG redaktori akna tegemise võimalus veebilehel, mis leidis rakendust mitmetes sisuhaldussüsteemides. Teiseks vähemtuntud ja kasutatud võimaluseks oli teatud kõrvalteid pidi saavutada rakenduse suhtlemine serveriga ilma lehekülge uuendamata. Kolmandaks vähetuntud ja kasutatud võimaluseks oli kliendi poolse vektorgraafika tegemise ja skriptimise võimalus.

Kiiresti sai tuntuks võimalus kõikide veebilehe elementide skriptimiseks. Kui varem sai JavaScriptiga muuta vaid vormiväljade sisu ja vahetada pilte ja muu sisu loomine JavaScripti abil sai toimuda vaid `write` meetodiga raami või akna sisu ülekirjutamisel, siis nüüd oli võimalik muuta iga elemendi sisu, asukohta ja väljanägemist. Sellised võimalused olid ka Netscape 4 brauseril, kuid natuke piiratumad.

Üksikleiutised mingi tehnoloogia piires ei ületa oma tasemelt tehnoloogia sünnile aluse pannud innovatsiooni taset. Brauserite areng aastatel 1995-1998 viitab järgneval skeemil (Joonis 12) lapsepõlve viimastele aastatele. Peale seda tehnoloogiliste uuenduste tase langeb. Ka uuenduste arv langeb alguses, kuid hakkab reeglina kuskil kasvuea keskel taas tõusma. Brauseri neljanda versiooniga saabus teatud pikem vaikus, Netscape parandas küll pidevalt oma lõputuid probleeme, aga seda tegevust ei saa nimetada uuendamiseks.

Joonis 12



Üksikute leiutiste tase tehnoloogia arenguetappides

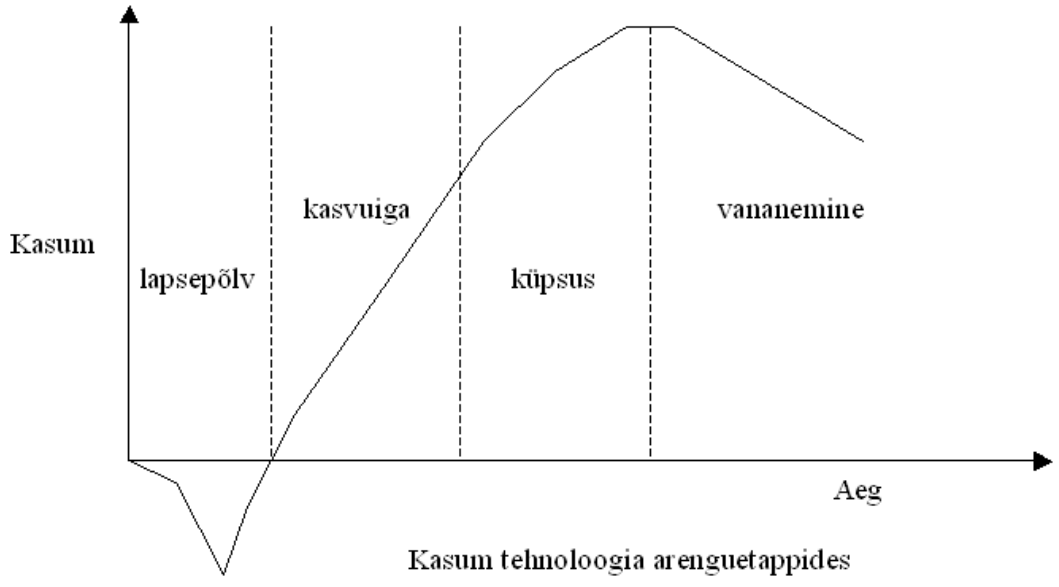
Juunis 1998 tuli Internet Explorer beetaversiooniga 5, mille lõppversioon saadi valmis märtsis 1999. Seda võib kindlasti käsitleda uuendusena ja see kajastub tehnoloogia uuenduste arvu kujutaval skeemil (Joonis 10). Oma tasemelt midagi nii pöördelist enam ei saavutatud. Oktoobris 1998 oli W3C välja töötanud dokumendi objektimudeli standardi (DOM 1998) ning uus brauseriversioon püüdis seda võimalikult palju järgida, kuid viienda versiooni varasem väljatulemine ei võimaldanud seda täiel määral teha. Enesestmõistetavalt tagati ka tagasiulatuv ühilduvus kõikidele veebilehtedele ja rakendustele, mis olid kodeeritud neljanda versiooni mudelist lähtuvalt. Seetõttu puudus arendajate seas reaalne vajadus kodeerida lehti veel vastavalt uuele DOM-ile. Niikuinii erinesid dokumendi objektimudelid Internet Explorer 4 ja Netscape 4 vahel olulisel määral ja neljanda versiooni kasutajate hulk oli veel märkimisväärselt suur. Kolmanda lisastsenaariumi kodeerimine üksnes selleks, et olla kooskõlas W3C soovitustega ei tundunud mõistlik.

Peale standarditega ühildumise tõi viies versioon veel mitmeid kasulikke uuendusi, nagu `XMLHTTP` objekt, millel rajaneb tänapäeval AJAX nime all tuntud tehnoloogia. Skriptimise seisukohalt olid uued elementide loomise ja manipuleerimise meetodid nagu `createElement`, `appendChild` ja teised, peale selle oli toetus uuele vektorgraafika keelele VML. Kuid midagi põhimõtteliselt uut, mida poleks neljanda versiooni mudeliga kuidagi teha saanud enam ei tehtud.

Kui siiani on jutt olnud vaid brauseritootjatest ja brauserite arengust, siis ei saa tähelepanuta jätta ka veebi sisuteenuste arengut. Võib tunduda, et tehnoloogilise arengu kirjeldus läheb jalgrattaga võrreldes väga keerukaks, sest jalgrattal sisuteenuse pakkuja puudub, aga pole mõtet asja näha keerulisemana, kui see tegelikult on. Veebirakenduse või teenuse arendaja ja veebibrauseri suhe on nagu jalgratturi ja jalgratta suhe. Brauseritootjate põhilised kliendid on siiski veebi sisu arendajad, mitte lõppkasutajad. Ja lõppkasutajad ei tarbi brauseritootja produkti, vaid teenuseid, mida pakuvad veebi sisu arendajad. Selles mõttes on parem ära unustada, et brauseritootjal ja brauserikasutajal on mingi suhe. Kui jalgratta võrdlust kasutada, siis sõidab lõppkasutaja sisu arendaja seljas. Pisivõimalused nagu *tabbed bowsing*, ja muu selline, mida vahest brauseri uue omadusena rõhutatakse, moodustavad brauseri arendustööst väga tühise osa.

Kuid nagu öeldud on brauseri tootja kliendiks veebi arendajad ja alati on teatud ajaline nihe tehnilise uuenduse tekkest kuni selle realiseerimiseni kasuliku teenusena. Esimene arvutihiir oli küll natuke kohmakas, aga see ei olnud põhjus, miks seda kohe kasutama ei hakatud nagu praegu. Seda suhet väljendab kõige paremini tehnoloogia arenguetappide kasumlikkus (Joonis 13):

Joonis 13



Lapsepõlv nõuab hoolt ja ülalpidamist. Ükski uus tehnoloogia ei muutu kohe kasutoovaks. Kulud ja tulud võivad nulli jõuda kuskil lapseea lõpus. Aasta 1997 lõpus ei olnud arendajatel erilist motivatsiooni rakendada viimaseid tehnoloogilisi uuendusi, mis oma arvu ja taseme poolest olid just äsja haripunkti jõudnud. Kuid varsti peale seda siiski esimesed veebilehed, teenused ning rakendused ilmusid, kus uut tehnoloogiat ära kasutada püüti. Aga ega sellele väga kiiresti praktilist väljundit ei leitud, pigem paistis silma ebapraktiline pool. Arendajate jaoks väga tähtis on brauserite toetus, ja tavaliselt võtab see aega paar aastat, et uuele brauseriversioonile on üle läinud 65-70% selle kasutajatest. Eri brauserid ei toeta samasid omadusi või tulevad nendega turule eri aegadel ja seetõttu pikeneb ooteaeg veelgi. Teatud rakendusi võib teha ühele või mõnele kindlale brauseriversioonile, kuid suuremate teenuste jaoks oodatakse ka suuremat toetust ja küpsust.

Ma arvan, et see soovitav küpsuse aste tähendab, et 90-95% kasutajatest on vajaliku tehnoloogiaga varustatud. Oletus rajaneb brauseriversioonide statistikale aasta 2004 lõpus (W3School, OneStat) ja Google uute teenuste väljatulemise faktile samal ajal. Kuna Google uued teenused kasutavad väga julgelt JavaScripti ja töötavad vaid teatud brauseritega kindlatest versioonidest alates, siis võib tolleaegse statistika põhjal järeldada, et nemad pidasid sellist protsenti piisavaks. Google valdab muidugi täpsemat statistikat ja võimalik, et see ei lange üldse kokku siinkasutatuga.

Kui see oletus vastab tõele, siis saame veel teisegi seaduspärasuse, mis näitab, et lapsea ja kasvuea perioodid on enamvähem võrdsete pikkustega ja ees võib olla veel üks sama pikk periood tõusvat arengut. S-kõver ei ennusta kindlalt seda, et peale küpsuse perioodi on tehnoloogia vältimatul vananemisele ja hukule määratud. Tõenäolisem stsenaarium on see, et mingi iseseisev S-kõver hakkab arenema selle tehnoloogia sees või kõrval (Addison 2003, 47) ja viib vana tehnoloogia uuele tasemele. Lennukite arengus oli selleks näiteks puust ja trossidega toestatud biplaanilt üleminek alumiiniumisulamist monoplaanile ja hiljem reaktiivmootoritele, või jalgratta puhul, õhukummide leiutamine. Brauserite puhul võib olla selleks mõni tehnoloogia, mis on juba praegu siin, võibolla vektorgraafika või Canvas elemendi skriptimis võimalused, mis on uuemate brauserite varustusse ilmunud. Teisest küljest võib selleks osutuda mingi tehnoloogia, mida me praegu brauseriga veel seostada ei oskagi.

Igatahes viitab kõik sellele, et brauserite poolt pakutav platvorm on väljunud oma lapseast ja kasvuraskustest ning on saavutanud teatud stabiilsuse, küpsuse ning ühtsuse.

5.2 Õige tehnoloogia valik, ehk kes mäletab enam piiparit?

Kes mäletab enam piiparit? Kes kasutab praegu piiparit. Me võime tuua palju näiteid tehnoloogiast, mis on osutunud evolutsioonipuu surnud oksaks. Kiiresti arenevates valdkondades nagu seda on veebitehnoloogia, peame kindlasti jälgima, et meie poolt arendatav tehnoloogia ei osutuks üheks selliseks piipariks. Käesolves alalõigus püüan kaaluda erinevaid tehnoloogilisi alternatiive püstitatud eesmärkide saavutamiseks ja põhjendada oma valikut.

Põhimõtteliselt on valikuid kliendi rikastamiseks kaks:

- kasutada brauseri enda skriptitavust, mis tähendaks siis ECMAScripti abil DOM objektide ja CSS omadustega manipuleerimist, mille abil anda veebirakendustele desktop rakendustega samaväärset kasutajakogemust.

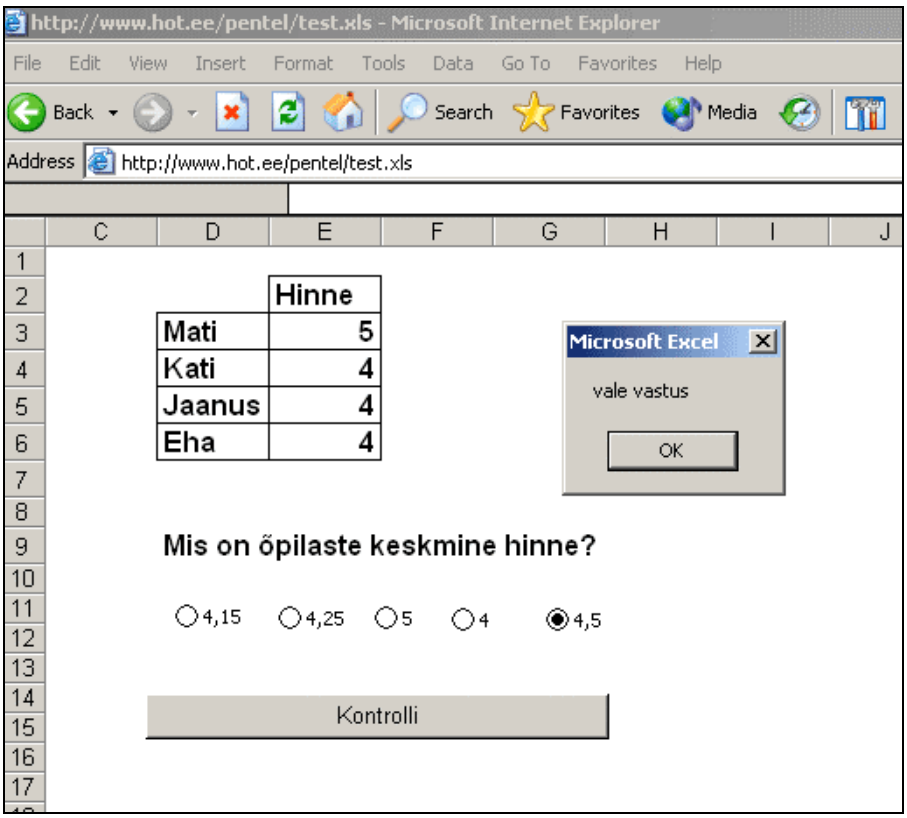
- kasutada lisasid ehk pluginaid (ing. *plug-in*) nagu Macromedia Flash (Macromedia), Java applet või mingeid *control* elemente.

Lisade puuduseks on nende aeglus ja see, et nad moodustavad brauseris teatud esiletükkiva võõrkeha, mis katab kinni kõik html elemendid (*Windowed Control*). See välistab Flashi või Java appleti integreerimise DHTML tehnikaid kasutavasse süsteemi, või tuleb selleks näha palju vaeva, et vältida võimalikke ebamugavusi kui mingi element jääb osaliselt kinnikaetuks. Lisad ei kombineeru HTML elementidega, nende sees ja vahel, ei saa olla fragmente HTML-ist ja seetõttu jäävad nad alati teatud võõrkehaks brauseris.

Seega võiks öelda, et lisade kasutamine õigustab ennast siis, kui kogu rakendus ongi tehtud Javas või Flashis või muud lisa kasutades. See püstitab aga küsimuse, kas me ei degradeeri brauserit vaid raamistikuks, mis võimaldab sisestatud aadressil paiknevat lisa enda arvutisse tõmmata. Mõnes mõttes on olukord sarnane ju iga allalaaditava programmiga, mis siis serveriga suhtlema hakkab.

Veel parem näide on teatud formaatides dokumendid, mida näiteks Internet Explorer vaikimisi avab brauseriaknas. Näiteks võib tuua MSOffice paketi abil loodud dokumendid, presentatsioonid või Exceli tabelid. Kõik need võivad sisaldada makrosid ja nende abil ühendust võtta serveriga. Põhimõtteliselt võib sel moel luua väga rikka kliendikeskkonna, kuid kas seda saab nimetada veebipõhiseks keskkonnaks.

Pilt 3



Joonis 14: Kas brauseriaknas avanenud Exceli rakendus on veebirakendus?

Kas fakt, et valdav enamus maailmast kasutab operatsioonisüsteemina Microsoft Windowsit, brauserina Internet Explorerit ja kontoripaketina MSOffice't, teeb veel veebis publitseeritud Exceli dokumendist veebirakenduse? Sõltub definitsioonist, kuid antud uurimuses kitsendan veebirakenduse tähendust sellega, et välistan taolised allalaaditavad kliendi keskkonnad, mis oma tööks peavad tegelikult kasutama muid interpretaatoreid.

Kui veebirakenduse peamiseks eeliseks muude klient-server süsteemide ees on selle kasutuselevõtmise lihtsus, kuna ei pea kõikide kasutajate jaoks installeerima eraldi

kliente, siis teatud mõttes see eelis kaob, kui tuleb siiski igale kliendile eraldi installeerida vastav plugin. Arvutivõrgus, kus tavakasutaja õigused on piiratud, võib see osutuda oluliseks takistuseks.

Probleemina võib välja tuua ka selle, et brauseri JavaScripti toetus võib samuti olla välja lülitatud. Sellekohast statistikat on tehtud (W3Schools), aga statistika tegemine pole sugugi lihtne. Kui see baseerub tavalistel serveri logidel ja võrreldakse lehele tehtud päringute arvu ja selle lehe kaudu laetud eradiseisvale JavaScripti failile tehtud päringute arvu, siis suur osa JavaScripti failide päriguid jääb toimumata lokaalse vahemälu või proxy serveri kasutamise tõttu. Teatud märkimisväärne osa statistikas on ka otsingumootorite robotitel. Kui JavaScript on välja lülitatud, siis eeldan, et selle sisselülitamine rakenduse mugavamaks kasutamiseks on lihtsam, kui uue interpretaatori (Flashi pleieri, Java virtuaalmasina) installeerimine. Siiski võib ka see nõuda administraatoriõigustega inimese sekkumist. Seega on plussid ja miinused praeguses arutelus 50/50. Tuleb kaaluda seda, mida võimaldab teha Flash, Java ja Javascript.

Java on kahtlemata suuremate võimalustega omades head tuge võrgurakenduste loomiseks. Java abil saab klient pöörduda otse andmebaaside poole, tehes näiteks SQL päringu. Kuid Javat brauserite standardvarustusse enam ei panda ja see kahtlemata vähendab temale tehtavaid panuseid brauseris kasutatava klientrakenduse osana.

Pärssivat mõju lisade kasutamise poliitikale avaldab otseselt Microsofti puudutav kohtuasi aastast 1999 ühes patendi küsimuses (Patent 5838906 1998), kus Microsoft on kostjaks ning hagejaks on California Ülikool ning üks väikefirma Eolas Technologies. Nimelt väidavad hagejad, et lisade brauseris kasutamise tehnoloogia on patenteeritud nende poolt ja Microsoftil pole õigust kasutada lisasid sel moel nagu seda seni on tehtud (Eolas). Aastal 2003, 11 augustil Chicago föderaalkohus Microsoftilt hagejatele 520,6 miljonit dollarit kahjutasu (Davis 2003). Summa moodustus sellest et iga vahemikus november 1998 – september 2001 müüdud koopia pealt arvestati 1,47$. Microsoft apelleeris selle otsuse. 29. oktoobril 2003 esitas W3C direktor ja veebi leiutaja Tim Berners-Lee taotluse selle patendi ülevaatamiseks (Berners-Lee 2003). Aasta 2005 kevadel apellatsioonikohus osaliselt tühistas eelmise astme kohtu otsuse, kuid olulistes punktides jäi Miscrosoft endiselt süüdi. Kuigi kahel patendi

ülevaatamisel leiti tõendeid sellest, et patent pole valiidne, siis sel sügisel tunnistati patent taas valiidseks ja Microsofti apellatsiooni ei rahuldatud.

Vaatamata sellel, kuidas kohtuasi lõpeb, on Microsoft asunud revideerima oma brauseritehnoloogiat, eriti selles osas, mis puudutab ActiveX tehnoloogiat. Teatavasti on Microsofti brauserite lisade nagu Java ja Flash playeri kasutamine seotud ActiveX tehnoloogiaga. Võimalik, et tulevikus tuleb ümber teha paljud lehed, mis seda tehnoloogiat kasutavad ja lisade integreerituse tase jääb veel väiksemaks, kui see on praegu (Fried 2005).

Sellise kitsenduse tulemusel ei jäägi muud üle kui brauseri poolt interpreteeritav keskkond, (X)HTML, CSS ja JavaScript.

JavaScripti see patendiküsimus ei puuduta, kui välja arvata mõned JavaScriptiga koos kasutatud süsteemseid ActiveX objektid, mis vaikimisi olid liigitatud kategooriasse turvalised ja nende töö toimus ilma hoiatusteta nagu tavalise JavaScripti puhul. Üheks selliseks viimasel ajal väga palju populaarsust võitnud objektiks on XMLHTTP object. Uuest Internet Explorer 7 versioonis ongi see objekt asendatud ActiveXist sõltumatu loomuliku objektiga.

Väga paljut sellest mida võimaldab Java või Macromedia Flash saab tänappäeval korda saata ka puhta JavaScripti, dokumendi objektimudeli, CSSi abil. Mitmeid asju saab ka paremini. Brauseri arenduskeskkonnana kasutamise võimalused on oluliselt avardunud, brauserid on ühtlustanud oma dokumendi objektimudelite realisatsioonid W3C soovitustest lähtuvalt ja platvormivahelised erinevused pole enam nii suured, kui need olid näiteks 5 aastat tagasi, kui pidi arvestama vähemalt kolme eri tüüpi DHTMLi toetava käsitlusega ja teatud kolmanda põlvkonna brauserite olemasoluga. Kui me praegu jätame brauseri sisseehitatud võimalused kasutamata, siis me raiskame sellega kliendi ressursse.

Suhtumine selles osas on siiski muutumas. Ja huvitaval kombel ei ole seda suhtumist esilekutsunud mitte uute tehnoloogiliste võimaluste tekkimine, st nende pakkumine brauseritootjate poolt, vaid suur tegija Google, kes olemasolevad võimalused on julgelt

ja jõuliselt kasutusse võtnud ja muutnud sellega üldist arusaamist veebist ja veebirakendustest.

Aasta 2005 moesõnaks veebitehnoloogias sai AJAX (Garret 2005). See on termin, mis iseenesest ei tähenda mingit uut tehnoloogiat vaid teatud tehnoloogiate kooskasutamist, milleks on siis XMLHttpRequest, DOM, CSS ja ECMAScript. Tehnoloogia kasutamise eesmärk on saavutada kasutajakogemust asünkroonsest andmevahetusest, mis asendaks seni levinud “kliki ja oota” tüüpi andmevahetust. Põhimõtteliselt võimaldab see teha serverile päringuid taustal nõudmata kogu lehekülje uuesti laadimist. Viimasel ajal on terminit AJAX kasutatud ka tehnoloogia kohta, mis XMLHttpRequest objekti üldse ei kasuta, ega ka muid alternatiive serveriga taustal ühenduse pidamiseks. Nagu varasema moeõna DHTML puhul omistatakse ka siin ühele nimetusele erinevat sisu.

Kuna nimetatud tehnoloogiad üksikuna ja ka koos ei ole sugugi uued, veelgi enam, osaliselt samaväärseid tulemusi on saavutatud ka teiste vahenditega, siis tekib õigustatud küsimus, miks sellest nii palju räägitakse. Vastus sellele on lihtne: sellepärast, et üks Google võttis need tehnoloogiad kasutusele ja arendas üsna samal ajal ja massiivselt kliendipoolsetele skriptimistehnoloogiatele panustades välja kaks teenust (Google Maps), (Google Suggest) ja ühe rakenduse (Gmail). Nüüd kui see Googlel on, siis tahavad seda kõik.

Kui üks selline tegija nagu Google usaldab oma teenused mingi kindla tehnoloogia hooleks, siis tähendab see esiteks selle tehnoloogia küpsust ja laiaulatuslikku toetust veebiklientide ehk brauserite poolt ja teiseks seda, et toetus nendele tehnoloogiatele lisatakse ka brauseritele, kus see veel puudub. Viimase näiteks on internetibrauser Opera, kes alates 7,64 versioonis samuti XMLHttpRequest toetuse osaliselt realiseeris (Opera Changelog 2004) ja hiljem veel mitmete uute klienti rikastavate tehnoloogiatega välja tuli. Mingis mõttes on vaja ümber hinnata kõik see, mida me oleme seni arvanud veebikliendi skriptimisega kaasnevatest piirangutest. On oodata suur hulka täiendavaid tehnoloogiaid mida veebikliendid toetama hakkavad. Selle näiteks on Opera 8 versioonis realiseeritud loomulik tugi skaleeritavale vektorgraafikale, mis defineeritakse XML tüüpi koodiga (W3C SVG). Selles osas käisid arendustööd ka Mozillal, mis on nüüdseks realiseerunud. Samuti on oodata

olulisi uuendusi ECMAScriptis, mis on seotud loomuliku XML toe loomisega (ECMA 2003), et vähendada kulu, mis hetkel seotud DOM kohmaka parsimisega. On oodata mitmete uute analoogsete meetodite implementeerimist.

Peab siiski mainima, et kasvõi see sama XMLHttpRequest meetod pole veel standardiseeritud, ehkki W3C töös on sarnane tehnika - DOM 3 Load and Save specification (W3C Load). Standardiseerimata on ka WYSIWYG redigeerimine, mis Internet Explorer 4 eeskuju järgides on aasta 2005 lõpuks muutunud de facto standardiks kõikide tuntud brauseritootjate seas. Kiired arengud tingivad selle, et standardid ei jõua neile arengutele järgi, kuid piisava huvi korral leitakse muid ka koostöövorme, mille tunnistuseks on Mozilla, Apple ja Opera WHATWG grupp.

Veebirakenduste võimalused on ümberhindamisel, mitte ainult arendajate poolt vaid ka kasutajate poolt. Uus ja meeldiv kasutajakogemus hakkab üha suuremat rolli mängima ja tekitab sellega lumepalliefekti.

Kui vaadata sellest seisukohast IVA õpisüsteemi, siis on see suhteliselt õhukese kliendi tüüpi rakendus. Õpisüsteemi võib käsitleda teatud seguna sisuhaldussüsteemist, dokumendihaldussüsteemist ja autorsüsteemidest ning paljudes seda tüüpi rakendustes, eriti sisuhaldussüsteemides on kliendipoolsete tehnoloogiate rohke kasutamine juba mõnda aega aukohal. Omades sellealast varasemat kogemust, tegin ka IVA õpisüsteemi suhtes vastavad ettepanekud ja kirjutasin aasta 2004 suvel ühe osana käesolevast tööst IVA testimissüsteemi küsimuste sisestamiseks WYSIWYG redaktori. Uurimus, mis sellele tööle eelnes oli leidis väljundi TPÜ Research Methods kursuse lõppraporti näol aasta 2004 jaanuaris. Mõistes, et see on vaid piisk meres, olen käesoleva töö käigus loonud sellele mitmed edasiarendused ning parandused ja loonud töötavad prototüübid täiesti uute küsimustetüüpide loomiseks ja esitamiseks kui ka olemasolevate rikastamiseks.

6 Rakendusespetsiifiliste komponentide alternatiivid veebirakendustes

Rakendusliku uurimuse põhiküsimuseks on kuidas midagi teha (Järvinen 2001, 88). Olles määratlenud puudused olemasolevas süsteemis ja soovitatava muutuse, peab keskenduma sellele kuidas soovitud muudatust realiseerida. Seetõttu on järgnevate peatükkide keel suhteliselt tehniline. Üks võimalus tehnilise osa vältimiseks oleks vormistada see töö lisadena, kuid antud uurimuse seisukohalt ei loe ma seda õigeks. Vastamine küsimusele “kuidas” ja uue loomine on siiski selle uurimuse peamine sisu.

Iga järgnev alapeatükk käsitleb mingit konkreetset kliendi interaktsiooni liiki või tarkvara omadust, mis on kaasaegsetes lauaarvuti rakendustes harjumuspärane, kuid mille realiseerimine veebirakendustes on võimalik vaid kliendipoolse skriptimise tulemusel. Lähenemine probleemile algab tehnoloogia ajaloolise tausta ja kasutusvaldkondade tutvustamisest ning seejärel asun praktilise realiseerimise juurde. Praktilised näited algavad lihtsustatud näidetest, mille eesmärk on tutvustada käsitletava tehnoloogia üldist tööpõhimõtet. Käsitlemist leiavad sama tulemuse saavutamiseks olemasolevad erinevad alternatiivid. Seejärel püüan lahendada tutvustatud tehnoloogia abil konkreetseid ülesandeid, mis on peamiselt seotud IVA testimissüsteemi täiendamisega. Kui lahendatavaid ülesandeid on mitu, siis olen ülevaatlikkuse huvides jaganud need alapeatükkidesse.

Iga peatüki teine poolt on pühendatud üksikutele probleemidele, mida käsitletud tehnoloogia kasutamine võib kaasa tuua ja nende probleemide võimalikele lahendustele ja nende vältimisele. Samuti puudutan võimalikke tulevikustsenaariume, mis lähtuvad brauseritootjate ja standardite tööversioonidest.

6.1 AJAX ja sellega seotud tehnoloogiad

Tänapäeval on omandanud sõna AJAX väga erinevaid tähendusi ja nagu omal ajal DHTML, ei tähistagi see mingit konkreetset tehnoloogiat vaid üsna hajusalt piiritletud segu erinevatest tehnoloogilistest võimalustes. Tuues näiteks DHTML tehnoloogia, siis selle all on tavaliselt mõistetud JavaScripti abil dokumendi objektimudeli ja CSS'i manipuleerimist saavutamaks interaktiivset ja vahetult reageerivat kasutajakeskkonda. Kuigi DHTML on vanem, kui W3C poolt kinnitatud DOM, siis tavaliselt on selle tehnoloogia all mõeldud siiski manipulatsioone ühe dokumendi piires. See tähendab, et uued aknad ja raamide ülekirjutamine väljuvad traditsioonilisest DHTML definitsiooni ulatusest. Aga tegelikult polegi mingit definitsiooni, on vaid ühe ajastu moesõna mida erinevad huvigrupid on oma äranägemisel tõlgendanud. Näites Microsoft nimetas DHTML dialoogideks oma modaal- ja mittemodaaldialooge, mis on olemuselt pigem aknad.

Sama puudutab ka termini AJAX kasutust. Algselt defineeriti see kui *Asynchronous JavaScript and XML*. Ja tehnika seostati põhiliselt Internet Explorer 5 poolt kasutusele võetud `XMLHTTP` objektiga, mis võimaldas JavaScripti abil teha otse HTTP päringuid (näiteks GET, POST, HEAD) ilma selleks lehekülge laadimata. Mozilla implementeeris samasuguse objekti versioonis 1 ja hiljem on nende eeskuju järginud ka Safari alates versioonist 1,2 ning Opera alates versioonist 7,6. Selle objektiga seotud asünkroonsus tähendab, et tehes päringu, ei pea ära ootama päringu lõppu vaid selleks on spetsiaalne sündmusehaldur, mis päringu eduka vastuse lõpu suudab vastu võtta ja seepeale mingi funktsiooni käivitada.

Ka AJAX'i X, mis tähistab XML'i, sisaldub juba objekti nimes, mis praeguseks on kõikide uute brauseriversioonide puhul `XMLHttpRequest`. Tegelikult ei määra see meetod sugugi vahendatavate andmete formaadina XML'i ja on selles mõttes natuke eksitav. Vahendatavad andmed võivad olla ükskõik millises tekstiformaadis ja neid saab vastu võtta tavalise string tüüpi muutujana. Kuid enne selle tehnoloogia üksikasjade käsitlemist ja konkreetseid näited, teen väikese tagasivaate ajas, et näha millised arengud on toimunud enne seda ja milliseid alternatiive on AJAX-iga sarnaste tulemuste saavutamiseks varem kasutatud.

6.1.1 Varasem ajalugu

Juba üsna varsti peale JavaScripti sündi tekkis teatud katsetusi muuta lehekülje sisu ilma kogu lehekülge taaslaadimata. Üks võimalus momentaalseks tagasisideks oli kasutada selleks vormiväljade sisu muutmist või kirjutamist olekureale. Kuid viimast ei pruukinud kasutaja üldse märgata ja vormiväljadesse sai sisestada vaid ilma vorminguta teksti. Teine võimalus oli pildilise informatsiooni muutmine. Pildi `src` atribuudi muutmine JavaScripti abil võimaldas laadida vana pildi asemele uue pildi. Kuid HTML vormingus fragmentide muutmine ühe lehekülje piires oli veel kolmandate brauserite ajal võimatu. Siiski pakkus teatud võimaluse selleks `frame` element, mille puhul võis laadida sisu vaid ühe `frame` elemendi kaupa. Laadimine ei andnud siiski päris dünaamilist efekti, sest jättis kasutaja ootele kuni selle ühe raami sisu laetakse. Märksa dünaamilisema efekti andis `frame` elementide ülekirjutamine teisest raamist `document.write` meetodiga. Järgnev näide on testitud Netscape 3 brauseriga. Näiteks järgmise `frameset` faili korral:

Koodinäide 2

```
<html><head>
<frameset rows="*" cols="200,*">
<frame src="vasak.htm" name="vasak">
<frame src="parem.htm" name="parem">
</frameset>
</head></html>
```

Kus parem.htm faili sisu puudub ja vasak.htm sisu on järgmine:

Koodinäide 3

```
<html><head>
<script>
function kirjuta(midagi){
   parent.parem.document.open();
   parent.parem.document.write(midagi);
   parent.parem.document.close();
}
</script>
</head><body>
<a href="#"
```

```
onmouseover="kirjuta('<html><body>tere maa!<\/body><\/html>')">
kirjuta teise raami
</a><br><a href="#"
onclick="kirjuta('<html><body>tere jälle!<\/body><\/html>')">
kirjuta veel
</a></body></html>
```

Siis vasakul asuvad kaks linki, millest esimene reageerib sündmusele `mouseover` ja teine sündmusele `click` ning mõlemad neist kirjutavad parempoolse raami sisu täielikult üle funktsiooni `kirjuta()` argumendiks määratud stringiga.

Selle võimaluse ärakasutamine sisuarendajate poolt oli signaaliks brauseriarendajatele, et sellised ülekirjutatavad elemendid võiksid olla ka ühe lehekülje sees. Nagu teame need võimalused realiseeriti Netscape 4 ja Internet Explorer 4 brauserite puhul. Kuid veel tagasi tulles kolmanda brauserite võimaluste juurde, siis vaatamata vahetule tagasisidele, mida raami ülekirjutamine teisest raamist suutis pakkuda, seadis see tehnika arendajatele omad piirid, mis seisnesid korraga laetavate failide mahus. Et ehitada keerukamat süsteemi, selleks oli vaja laadida andmeid järkjärguliselt.

Selleks oli võimalus samuti enne brauserite neljandaid versioone sellesama raamistiku kasutamise abil. Nimelt võis luua raamistikus ühe raami laiuse või kõrgusega 0, mis tähendab, et raam jäi nähtamatuks:

Koodinäide 4

```
<html><head>
<frameset rows="*" cols="0,200,*">
<frame src="parem.htm" name="varjatud">
<frame src="vasak.htm" name="vasak">
<frame src="parem.htm" name="parem">
</frameset>
</head></html>
```

Nähtavast osast sai siis lingi `target` atribuudiga selle raami nimele viidates suunata laetavaid lehti. Esmapilgul võib tunduda imelik, miks peaks nähtamatusse raami suunama lehekülgi, kuid selle mõte on kasutajale nähtava osa hoidmine pidevas

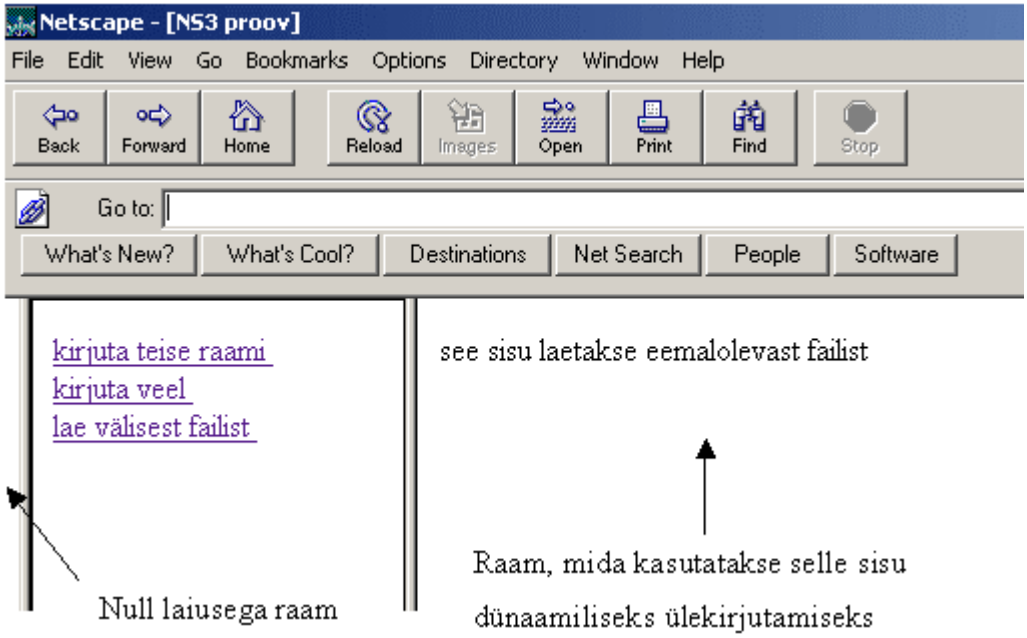
vaateväljas. Varjatud raami laetud lehel asus skript, mis käivitati selle lehe *load* sündmusega ja mis kutsus esile näiteks mingi nähtava osa ülekirjutamise:

Koodinäide 5

```
<html><head>
<script>
  function kirjuta(midagi){
  parent.parem.document.open();
  parent.parem.document.write(midagi);
  parent.parem.document.close();
}
function laemesisu(){
sisu='<html><body> see sisu laeti eemalolevast failist '
+'varjatud.htm<\/body><\/html>';
kirjuta(sisu);
}
</script>
</head>
<body onload="laemesisu()"></body>
</html>
```

Netscape 3 brauseril produtseerivad eeltoodud katsefailid järgmise tulemuse:

Pilt 4



Sellega olid loodud esimesed alged tehnikale, mida hiljem tunti nime all *remote scripting* ja mida alternatiivina ka tänapäeval üsna tihti kasutatakse, tõsi küll arhailise 0 laiusega `frame` elemendi asemel suunatakse päringud nüüd tavaliselt peidetud `iframe` elementi.

6.1.2 Neljanda põlvkonna brauserid

Alates neljanda põlvkonna brauseritest tõi Netscape 4 uuendusena kasutusele elemendid `layer` ja `ilayer`, mis võimaldasid peale manipuleerimise nähtavuse, positsioonide ja stiilidega veel välisest failist sisu laadida `src` atribuudi abil. Internet Explorer 4 seevastu tõi uuendusliku elemendina `iframe` elemendi, mis käesoleval hetkel on kõikide enamlevinud brauserite poolt toetatud. Erinevus `ilayer` ja `layer` elemendist seisnes selles, et `iframe` käitus nagu tavaline `frame` element, olles `frames` kollektsiooni liige dokumendi objektimudelis, kui `layer` oli Netscape 4 spetsiifilise `layers` kollektsiooni liige. `Iframe` tavalise `frame` elemendina võimaldas sellesse sisu laadida ka ilma JavaScripti abita tavalise lingi abil, mille `target` atribuut viitas `iframe` elemendi nimele. Peale selle oli `iframe` elemendile omane kerimisriba, mis `layer` ja `ilayer` elemendil puudus ja mida sai vaid suhteliselt keeruka protseduuriga emuleerida. Kuid oluliseks puuduseks `layer` elementide ees oli `iframe` elemendi omadus katta kinni kõik elemendid, mis absoluutsete positsioonidega on sattunud, selle kohale. Sellist omadust evivaid objekte nimetati *windowed control* ja peale `iframe` elemendi kuulusid Internet Explorer 4 puhul sellesse rühma Java Appletid ja kõik lisad (*plugins*) nagu Macromedia Flash ja teised. HTML elementidest, oli *windowed control* veel `select` element, mis püsis sellisena veel versioonini 6 (parandatud versioonis 7).

Mis puutub Netscape 4 brauserisse, siis paistsid `layer` elementidest läbi peale `select` elemendi kõik muudki vormi sisestuselemendid nagu `input`t ja `textarea` ja samuti appletid ning lisad. Kuid rääkides eelistest, mis olid `layer` elemendil `iframe` elemendi eest, siis üks nende seast oli see, et `layer` kuhu oli laetud välisest failist pärinev sisu, ei olnud *windowed control* tüüpi ja seda oli võimalik vajadusel

osaliselt või täielikult teise *layer* elemendiga kinni katta (näiteks mingi menüüga). Kuna DHTML rippmenüüd oli üks esimesi neljandate brauserite poolt pakutud võimaluste populaarseid kasutusvaldkondi, siis oli *layer* elemendil teatud eelis *iframe* elemendi ees.

Kuid Internet Exploreri juures oli käepärane võimalus elementides sisalduva HTML koodi lugemiseks *innerHTML* omaduse abil. Seda võimalust sai ära kasutada kombinatsioonis kolmandate brauserite ajastust tuntud varjatud raami laadimisega. Selleks tuli luua varjatud *iframe* element, ning sellesse laetavasse faili lisada *load* sündmusega käivituv funktsioon, mis peamisel leheküljel muudatused esile kutsub. Kuid seoses *innerHTML* omadusega ei olnud nüüd vaja enam kõiki uuendusi defineerida JavaScripti koodis kirjeldatud stringina, vaid oli võimalus laadida ka lihtsalt varatud raami laetud lehekülje kogu HTML sisu. Lihtsustatult oleks peamine leht järgmine:

Koodinäide 6

```
<html><body>
<a href="varjatud_ie4.htm" target="varjatud">
laeme varjatud sisu</a><br>
<iframe src="about:blank" name="varjatud" width="0" height="0">
</iframe>
<div id="sisuKuvaja">Siia laetakse uus sisu välisest failist</div>
</body></html>
```

Ning laetava faili varjatud_ie4.htm sisu näeks välja järgmine:

Koodinäide 7

```
<html><body
onload="parent.sisuKuvaja.innerHTML=document.body.innerHTML;">
<div style="color:red">
See kiri laetakse välisest failist varjatud_ie4.htm
</div>
</body></html>
```

Tulemuseks oli võimalus ammutada sisu välistest failidest, siis kui seda parasjagu vaja oli. Sama tehnika töötab ka tänaste brauseritega, kuid siis tuleb kasutada sisu näitava elemendi *sisuKuvaja* poole pöördumiseks *getElementById()* meetodit:

Koodinäide 8

```
<body onload="parent.document.getElementById('sisuKuvaja').innerHTML=
document.body.innerHTML;">
```

Üheks suuremaks puuduseks selle tehnika puhul on vajadus niiöelda *callback* funktsiooni järele laetaval lehel, sest peamiselt lehelt oli küll võimalik varjatud raami laetud lehe sisu lugemine, kuid puudus info selle kohta, millal leht tegelikult laetud on. Kuna seda informatsiooni valdab vaid laetava lehe enda `onload` sündmusehaldur, siis peab sisu muutmist või muud toimingut käivitav protseduur asuma sellel laetaval lehel.

Teine puudus on seotud erinevatest domeenidest pärit lehtedega. Kui varjatud raami laetav leht ei asu samas domeenis peamise ehk laadiva lehega, siis põhjustab see brauseri tavaseadete korral veateate, sest turvalisuse kaalutlustel on skriptide ligipääs teises domeenis asuvale lehe sisule keelatud. Turvaintsidendina võib näiteks ette kujutada raamis avatud internetipanga lehekülge, mille teises varjatud raamis asuv skript loeb kasutaja poolt sisestatud infot ning postitab need mingile aadressile.

Kui eelnevalt leidsid käsitlemist vaid andmete uuendamisega seotud küsimused, siis andmete postitamine peidetud raami töötab samuti ja ei vaja isegi JavaScripti abi. Nagu linkidel, nii ka `form` elemendil on juba Netscape 2 aegadest `target` atribuut, mille väärtus määrab raami nime, kuhu postitus suunatakse. Kui tegu on varjatud raamiga, siis toimub postitus ilma lehekülge taaslaadimata.

6.1.3 Välise JavaScript faili dünaamiline laadimine

Teine tehnika, väliste failide dünaamiliseks laadimiseks seisneb `script` elemendi dünaamilises genereerimises või selle `src` atribuudi väärtuse muutmises. Dünaamiline genereerimine on seotud `createElement` meetodiga, mis tekkis alles Internet Explorer 5 versioonist alates, Dünaamiline HTML sisu genereerimine leheküljel oli Internet Explorer 4 puhul võimalik lehekülje või mingi konteinerelemendi `innerHTML` omadust muutes, kuid sel moel loodud koodiga ei saa luua `script` elementi, mis ka käivituks. See tähendab, et järgnev kood (Koodinäide 9), ei tööta:

Koodinäide 9

```
<script>
function lae(js){
document.body.innerHTML+='<scr'+'ipt src="'+js+'"></scr'+'ipt>';
}
</script>
<button type="button" onclick="lae('tere.js')">
Laeme välise faili
</button>
```

Seletuseks eeltoodud näite kohta veel nii palju, et funktsioon `lae()` kirjutab `body` elemendi lõppu juurde stringi, milleks on `script` element viitega välisele tere.js failile, mis on defineeritud funktsiooni argumendi `js` väärtusega. String on poolitatud pluss märkidega `script` elementi alustava ja lõpetava märgendi kohalt, et vältida konflikti, mis tekiks stringis sisalduva lõpetava `script` elemendi tõlgendamisega funktsiooni `lae()` defineeriva skript ploki lõpuna.

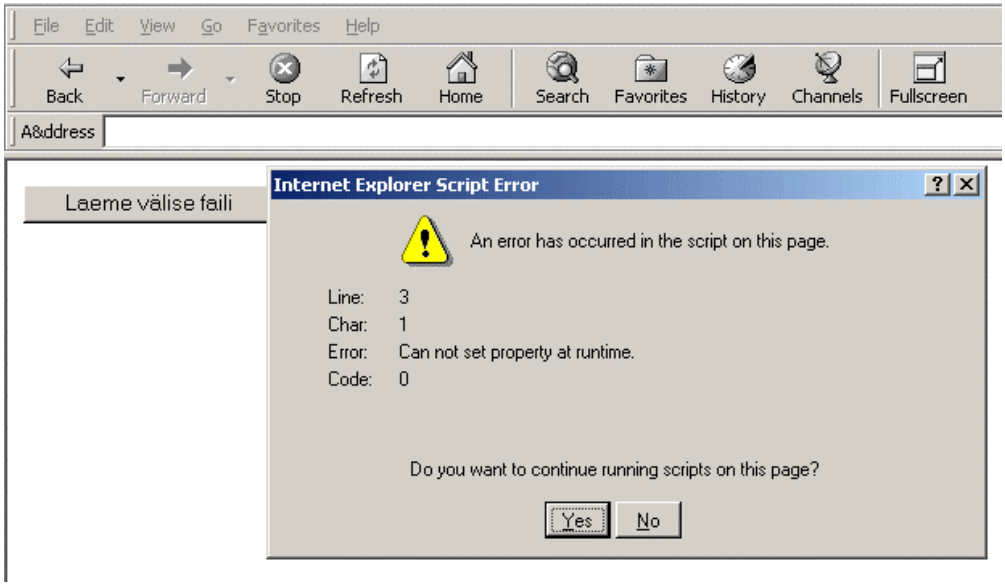
Kui kirjutada protseduur, mis väljastab meile lehe HTML koodi, siis võib veenduda, et peale nupuvajutust tõepoolest moodustus lehekülje lõppu rida <script src="tere.js"></script>, kuid see ei käivita protseduuri, mis on defineeritud failis tere.js. Nagu mainisin on võimalik leheküljel oleva unikaalse identifikaatoriga varustatud `script` elemendi poole pöörduda ning selle kaudu erinevaid skripte laadida. Sama näite modifikatsioon oleks järgmine:

Koodinäide 10

```
<script>
function lae(js){
jsLaadija.src=js;
}
</script>
<script id="jsLaadija"></script>
<button type="button" onclick="lae('tere.js')">
Laeme välise faili</button>
```

Internet Explorer 4 puhul on tulemuseks siiski veateade “Can not set property at runtime”:

Pilt 5



See tähendab, et Internet Explorer 4 ei luba dünaamiliselt `script` elemendil muuta `src` atribuudi väärtust. Sama tehnika töötab Internet Exploreri versioonidel 5 kuni 7 ja Opera versioonidel 7 ning uuematel. Viimase puhul tuleb muidugi muuta otsepöördumine `script` elemendi `id` kaudu `getElementById()` meetodiga. Kuid Opera 6 ja samuti Gecko mootoril rajanevad brauserid Mozilla, Firefox ja Netscape 6 ja uuemad sellise laadimisega hakkama ei saa. Kuid samalaadse tehnika realiseerimiseks on efektiivsem võimalus, mis on ka Gecko brauserite poolt toetatud. Nimelt tuleb `script` element luua `createElement` meetodiga:

Koodinäide 11

```
<script>
function lae(js){
  skr=document.createElement('script');
  skr.src=js;
  document.body.appendChild(skr);
}
</script>
<body><button type="button" onclick="lae('tere.js')">
Laeme välise faili</button>
```

Selle tehnika suurim eelis on võimalus laadida faile ka siis kui laadiv leht ja laetav fail kuuluvad eri domeenidesse. Puuduseks on juba *iframe* elemendi tehnikat käsitledes mainitud vajadus spetsiaalse funktsiooni järele laetavas failis, mis informeerib süsteemi, sellest, et laadimine oma lõpule jõudnud. Puuduseks võib nimetada ka laetava faili puhul vajadust spetsiaalse JavaScripti formaadi järele.

Kui vaadelda seda tehnikat andmete postitamise seisukohast, siis on olukord keerulisem. Kasutada saab vaid GET meetodit ning postitada andmed serverisse osana JavaScripti faili aadressist. Näiteks võib JavaScripti genereerida mingi serveripoolne skript, mis väljastab kõigepealt päise *content-type: text/javascript* ja samal ajal võtab vastu ka mingid muutujad. Eelmise näite modifikatsioon võiks olla siis järgmine:

Koodinäide 12

```
skr.src= 'JS.php?muutuja1= ' +muutuja1+' &muutuja2='+muutuja2;
```

Samalaadsel andmete postitamisel on kasutatud tihti ka tavalist pilti, kuna pildi *src* atribuudi muutmine oli võimalik juba varem ning pildi asemel võis olla mingi serveripoolne skript, mis pildi lõi ja samal ajal ka andmeid talletas. Eriti levinud oli see meetod omal ajal külalisteloendurite juures, mille aadressiga anti edasi muutujaid kasutaja ekraaniresolutsiooni ning värvisügavuse kohta ja muid andmeid, mille tuvastamine on võimalik vaid JavaScripti abil.

6.1.4 XMLHttpRequest – AJAX tehnoloogia aluskivi

Alates versioonist 5 implementeeris Internet Explorer tehnika eemalolevate failide poole poordumiseks JavaScripti abil *XMLHTTP* objekti kaudu. Mozilla võttis selle tehnika kasutusele alates versioonist 1 ja praegu toetavad seda kõik enamlevinud brauserid. Kui eelmised tehnikad kujutasid endast teatud ümbernurgalahendusi, siis *XMLHttpRequest* oli just nende eesmärkide saavutamiseks loodud tööriist.

Alates Internet Exploreri versioonist 7 on *XMLHttpRequest* objekti loomine kõikidel brauseritel samasugune, Internet Exploreri varasemad versioonid kasutasid ActiveX objekti, mis töötab brauseri tavaseadete korral ilma turvadialooge esitamata. Viimaste turvaparandustega on tekitanud palju segadust see, et Internet Exploreril ilmuvad turvadialoogid hoopis lokaalselt käivitatud failide puhul ja ka lokaalsesse arvutisse installeeritud veebiserveri korral, samas kui eemalolevas veebiserveris töötavad samad failid korrektselt.

XMLHttpRequest objekti loomine käib järgmiselt:

Koodinäide 13

```
var httpObj = null;
 if(window.XMLHttpRequest){ /*IE7, Mozilla 1+, Safari 1.2+, Opera
7,65+ */
     httpObj = new XMLHttpRequest();
  } else if(window.ActiveXObject){ /* IE5, IE5.5, IE6 */
     httpObj = new ActiveXObject("Microsoft.XMLHTTP");
  }
```

Tegelikult on neid ActiveX objekte, mida selleks otstarbeks kasutada võib, palju rohkem. Näites mainitud *Microsoft.XMLHTTP* on kõige vanem, hiljem on lisandunud veel uusi objekte, kuid vana objekt on endiselt toetatud.

Selle objektiga on võimalik kasutada meetodeid GET, POST ja HEAD, kuid põhimõtteliselt ka teisi nagu PUT, DELETE. Samuti on võimalus seada ise päringu päiseid. GET meetodil esitatud päringud, ei salvestu brauseri külastatud aadresside ajaloos (*history*).

Olles loonud objekti võime järgnevalt kasutada seda objekti erinevate päringute tegemiseks. Näiteks kui eesmärgiks on postitada mingid andmed POST meetodiga, siis tuleb loodud objekti kasutada järgmiselt:

Koodinäide 14

```
kuhu='faili_nimi_mis_päringuga_tegeleb_või_mida_päritakse';
sisu='andmed mida saadetakse';
kuidas=true;
httpObj.open('POST', kuhu, kuidas);
httpObj.send(sisu);
```

Nagu näeme meetodi `open` atribuutidest, tähistab esimene neist HTTP päringu tüüpi, teine serveris asuva ja päringut vastu võtva programmi aadressi ja kolmas argument on boolea tüüpi `true` või `false` vastavalt sellele, kas soovitakse asünkroonset või sünkroonset andmevahetust. Meetod `send` saadab argumendiks määratud muutuja `sisu` väärtuse teele. Tuleb tähele panna, et POST meetodi korral, ei saa postitavale muutujale kaasa anda mingit identifikaatorit, mida serveris saaks muutujana käsitleda nii nagu ollakse harjunud näiteks tegema PHP-d kasutades. Antud juhul läheb POST meetodiga saadetu, otse standardsisendisse *standard input,* ja sealt tuleb seda ka lugeda. Perli puhul näiteks:

Koodinäide 15

```
$saadetis=join('',<STDIN>);
```

Php puhul, kui skript on pandud tööle CGI-na, saab kasutada standardsisendit järgmiselt `php://stdin` ning edasi lugeda sellest juba nagu failist avades selle `fopen` meetodiga.

Mis puutub GET meetodisse, siis selle puhul ühildub lahendus igasuguste serveripoolsete traditsioonilisi vorme protsessivate lahendustega, sest meetodi `send` argument jääb sel juhul hoopis tühjaks või seatakse selle väärtuseks `false` ning kogu saadetav informatsioon esitatakse meetodi `open` teise argumendina, kus defineeritakse serveri vastuvõtva faili aadress:

Koodinäide 16

```
httpObj.open('GET', kuhu+'?muutuja1=väärtus1', kuidas);
httpObj.send('');
```

Peale andmete saatmise on olemas veel käepärane tehnika saabunud andmete kohalejõudmise üle valvamiseks. Selleks on spetsiaalne sündmusehaldur `onreadystatechange`, mis käivitub siis kui objekti omadus `readyState` omandab uue numbrilise väärtuse. Need väärtused annavad märku päringu käekäigust ja number 4 tähendab seda, et vastu serverist on kohale jõudnud. Objekti omadus `responseText` aga tagastab stringi kujul andmed selle kohta, mis serverist vastuseks saabus:

Koodinäide 17

```
httpObj.onreadystatechange=function() {
  if (httpObj.readyState==4) {
   alert(httpObj.responseText)
  }
 }
```

Nagu näha on see lihtne tehnika palju paindlikum, kui eelpool käsitletud alternatiivid. Ka kasutajakogemuses on suur erinevus võrreldes näiteks varjatud raami lahendusega, kus päringute toimimisest andsid märku mitmed välised indikaatorid, kuid antud juhul need puuduvad.

Suurimaks puuduseks on võimalus kasutada seda tehnoloogiat vaid ühe domeeni piires. Sellest üle saamiseks tuleb kasutada serveris asuvat vahendavat skripti, mida kliendis olev skript juhib ja mis tema eest päringuid esitab ja vastused tagasi kliendile edastab. Teise võimalusena võib mainida veel veebiserveri seadistamise võimalusi, näiteks Apache serveri `proxypass` direktiiviga (Apache mod_proxy) saab suunata mingi teises domeenis asuva lehe oma serveri alamkataloogiks.

6.1.5 Erinevate kaugskriptimistehnoloogiate plussid ning miinused

Tabel 3

Tehnika	**Plussid**	**Miinused**
Sisu laadimine peidetud raami kaudu	Ühildub iga brauseriga, mis toetab `frame` elementi ja JavaScripti load sündmust. Võimaldab POST meetodiga andmeid saata. Andmete saatmine on võimalik ka erinevasse domeeni.	Probleemid tagasisidega sellest, millal leht on tegelikult laetud. Andmete vastuvõtmine erinevas domeenis asuvast serverist ei ole ilma turvaseadeid pehmendamata võimalik. Päringud serverisse põhjustavad Internet Exploreril plõksumise.
Script elemendi dünaamiline loomine	Ainus tehnika mis tegelikult võimaldab eri domeenide serverite vahel andmeid vastu võtta ja saata.	Sisu peab vastama JavaScripti süntaksile. Sobib rohkem andmete vastuvõtuks kui saatmiseks. Laetava faili lõpus peab olema funktsioon, mis faili kohalejõudmisest teatab
XMLHttpRequest	Suurimate võimalustega vahend. Lubab saata ja vastu võtta päiseinfot ja omab spetsiaalset sündmust, mis registreerib päringu olukorda (kas on kohale jõudnud etc). Päringu vorm ei ole tähtis, kuigi mõeldud on meetod tööks tekstifailidega. Saab kasutada erinevaid meetodeid GET, POST, HEAD (põhimõtteliselt ka PUT jm)	Ei ole standard. Internet Explorer 5-6 nõuab ActiveX kasutamist XMLHttpRequest objekti loomiseks, mis karmimate turvaseadete korral lakkab töötamast. Ei võimalda andmevahetus eri serverite vahel

Eelpooltoodust näeme, et igal tehnikal on oma miinused ja plussid ja nende kasutamine võib olla kombineeritud ja/või sõltuda sellest, millist ülesannet tuleb lahendada.

XMLHttpRequest on spetsiaalselt selleks otstarbeks loodud tehnoloogia ja brauserite toetuse poolest ei erine see teistest tehnikatest enam olulisel määral. Samas evib ta võimalusi, mida teiste tehnikatega saavutada ei saa. Mõistlik tundub valida tööriistad sõltuvalt lahendusest ja jätta alati ka ilma skripti toeta brauseritele vähendatud funktsionaalsusega võimalused.

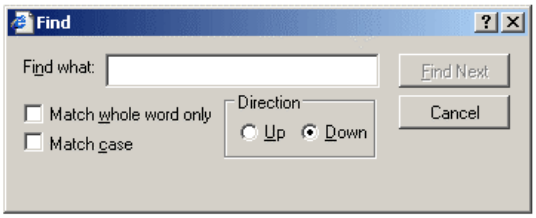
6.2 Dialoogid

Kuni kolmanda põlvkonna brauseriteni oli vaid kolm võimalust dialoogide tegemiseks:

- Disainida selleks lihtsalt spetsiaalne leht
- Kasutada JavaScripti dialooge `prompt`, `alert` ja `confirm`
- Kasutada aknaid

Neljanda põlvkonna brauserite advendiga lisandus dialoogide loomiseks uusi võimalusi. Kõige rakendusekesksemat suhtumist väljendas Internet Explorer oma modaalse ja mittemodaalse dialoogi loomisega. Modaaldialoog võeti kasutusse juba alates versioonist 4. Alates versioonist 5 võeti kasutusse ka mittemodaalne dialoog. Microsoft kutsub neid DHTML dialoogideks (Massy 2000), kuigi tihti mõistetakse DHTMLi all JavaScripti manipulatsioone kihtide ja CSS väärtustega ja dokumendi objektimudeliga ühe dokumendi piires. Internet Exploreri uued dialoogid on aga võrreldavad pigem `window.open` meetodil avatavate akendega. Sõltumata DHTMLi eri käsitlustest, tundus see Internet Exploreri muudatus igati asjakohane, veebirakenduste arendamise seisukohast.

Muuhulgas on huvitav ning paljudele teadmata asjaolu, et Internet Explorer kasutab ka ise dialooge, mis on ehitatud DHTML tehnikaid kasutades. Näiteks brauseri *Find* dialoog, mille leiab *Edit* menüüst. See dialoog on disainitud väljanägemiselt tavalise Windowsi dialoogiakna moodi, tegelikult on see aga DHTML dialoog, millest on kasutatud tavalisi HTML elemente kuvamaks nuppe ja sisestusvälja (Massy 2000).



Pilt 6: Internet Exploreri süsteemne find dialoog on tegelikult DHTML dialoog

Kahjuks langevad modaaldialoogid ka nende akende kategooriasse, mida tänapäeval brauseri enda seadete või väliste popup blokeerijate abil tihti ära keelata saab. Tehnoloogia väärkasutusel on omad viljad, millega tuleb arvestada ka edaspidi.

Võib mõelda, et head tehnoloogiat ei tohi lasta raisku minna, seetõttu, et seda kuskil vääriti kasutatud on ja meie rakenduse kasutajat võib teavitada, et ta oma popup blokeerija välja lülitaks. Ning mõistlikult seadistatud popup blokeerija ei takista nende akende avanemist, mille algatajaks on kasutaja ise. Aga see ei ole alati lahendus. Kasutaja ei pruugi seda osata või puudub tal selleks õigus. Näiteks võib asutuse arvutivõrgus olla popup akende blokeerija vaikimisi aktiveeritud ning kasutaja ei saa ka brauseriseadeid muuta. Mitmed teenused, mis on üles ehitatud selliste dialoogide peale on sellega halvatud.

Peale blokeerivate programmide ohu, ei ole aken kui selline spetsifitseeritud dokumendi objektimudeliga vaid on osa brauseri objektimudelist ja seega võib olla täiesti brauserispetsiifiline. Teatud ühtsetest traditsioonidest on brauserid kinni pidanud ja erandina võib mainida ehk Opera vanemaid versioone, kuid üllataval kombel Netscape 8 eirab traditsioonilisi aknaid ja avab need hoopis eraldi *tab*ina.

Väiksemate või dünaamiliste dialoogide mõte on kiirendada interaktsioone. Terve lehekülje taaslaadimine võtab aega ja justkui katkestab tegevuse. Raske oleks ette kujutada samasuguseid ebamugavaid katkestusi desktop programmi puhul, näiteks kui peaksime tavalise tekstiredaktori abil midagi vormindades iga nupuvajutuse järel 15 sekundit ootama ja ooteajal kaoks vahepeal pilt sootuks ja siis aeglaselt laeks uuesti. Kuid just nii näeb välja veebiaplikatsioon, mis panustab kõikide interaktsioonide puhul ainult HTTP meetoditele, mida juhitakse läbi HTML vormi elementide ning linkide.

Ühesõnaga veebiaplikatsioon peab sisaldama dünaamilisi dialooge, mis jätavad kasutaja ikka leheküljele edasi. Kuna popup blokeerijate buumi ajastul ei saa enam loota meetoditele, mis on blokeeritavad, tuleb kasutada teisi võimalusi, milledest lihtsamate dialoogide puhul on olemas vanad `alert`, `confirm` ja `prompt` dialoogid, kuid keerukamate puhul tuleb kõne alla DHTML tehnikatel baseeruv dialoog. Kuid erinevalt akendest, jõuame DHTML puhul taas olukorda, kus põrkame kokku mõnede elementide omadusega teistest läbi paista.

Internet Explorer 4 aegadel püüdsin lahendada keerulist olukorda: genereerida menüüd, mis oleksid üle tavalise freimi, nii, et mingi osa menüüst ulatuks freimi nn raami peale. Tolleaegsete tehnoloogiatega oli ainsaks võimaluseks kasutada selleks, window.open meetodiga avatud akent, mille viimaseks argumendiks on lisatud *fullscreen* ja siis meetodiga *reSize*, see aken parajasse suurusesse viia ning *moveTo* meetodiga õigesse kohta paigutada. Kuna *fullscreen* argument ei tekitanud aknale ühtegi aknale omast serva, nägi tulemus välja tõesti nagu tavaline lame kiht, ainsaks probleemiks oli akna genereerimisel toimuv välgatus, mis akna korraks siiski pidi üle ekraani laotama.

Toonane katsetus oli eksperimentaalne ja vajus unustusse, kuna akna positsioneerimine ekraani nurga suhtes ei olnud kerge, sest peaakna asukoht võis olla erinev. Eksperimenteerimise tuhinas proovisin, mis juhtub, kui selleks kasutada `iframe` elementi, aga see käitus siis (aastal 1998, Internet Explorer 4) väga kummaliselt, pressides ennast osaliselt üle `frameset` elemendi struktuuri. Nii nagu soovisin, seda siiski raami peale ei õnnestunud ajada, kuid katsetustest oli kasu hoopis teises kontekstis: kuna kasutasin menüüde tegemiseks kahte ifreimi, siis nende absoluutsete positsioonide korral tekkis tihti olukordi, kus nad olid natuke teineteise peal. Sellest ajast peale olen esiletükkivate objektide katmiseks nagu `iframe` ise, `select` element, Java applet või muu objekt `iframe` elementi kasutanud. Dialoog või menüü tuleb avada `iframe` elemendis ning `iframe` element liigutada ja teha nähtavaks sobivas kohas.

See on lahendus, mida ei saa kuidagi pidada rahuldavaks dialoogide puhul, aga IE 5 puhul paistab see olevat ainus lahendus peale spetsiaalsete dialoogide, millest eelpool juttu oli. Alates versioonist 5.5 võib kasutada ära `iframe` elemendi omadust olla nii *windowless* kui *windowed* element ja kasutada seda teatud aluspuhvrina nende elementide kohal, mida on vaja parasjagu kinni katta. Tehnika iseenesest pole eriti elegantne, kuid võimaldab mingit koodi väheste kuludega ühilduvamaks muuta. Muidu alates IE 5.5-st on olemas juba hoopis paremad tehnikad menüüde tegemiseks, mis lubavad menüül isegi üle akna serva avaneda.

Ühesõnaga tegu on menüüdega, mis vastavad kasutaja varasematele kogemustele ja sellest lähtuvale ootustele, mis neil on seoses menüüdega. Need ei kao nurga taha, ega paista neist teised elemendid läbi. Tegu on *createPopup* meetodil loodud elemendiga, milles saab kuvada dünaamiliselt loodud sisu. Element pole uus aken nagu `window.open` meetodil avatud aken ja on seetõttu palju kiirem. Samas seob teda aknaga kaks omadust: väliselt saab ta paigutuda üle brauseriakna äärte ja sisemiselt, mis puudutab tema asukohta brauseri objektimudelis, ei kuulu ta samasse dokumenti, kust ta avati. See tähendab, et *createPopup* meetodil loodud menüüst ei pääse otse ligi dokumendi funktsioonidele ja elementidele vaid neile tuleb viidata läbi *parent* objekti nagu `frame` elementide puhul. Tegu on *createPopup* meetodi korral kahjuks vaid IE spetsiifilise täiendusega ja selle eesmärk ei ole ka päris dialoogide loomine vaid pigem, mingite informatiivsete, siltide tekitamine võo kontekstmenüüde loomine. Siiski on analoogse leheküljesisese (`inline`) dialoogi vajadusest saanud aru ka teised brauseritootjad ja see kajastub ka nende veebirakenduste alase spetsifikatsiooni tööversioonis (WHATWG Web Apps 1.0, 2005).

IVA testimissüsteemis on väiksemate dialoogide vajadus näiteks WYSIWYG redigeerimise juures, millest tuleb juttu peatükis 6.4. 2004 aasta versioonis realiseerisin kaks diloogi traditsiooniliste `window.open` meetodil avatud dialoogidega ja ühe Internet Explorerile ainuomase dialoogi, mille alternatiiviks teistel brauseritel jäi `window.prompt` meetodil avatud aken.

6.3 Drag & Drop funktsionaalsuse kasutamine

Üks rakendustega seotud tehnika, mis vastab Apple kasutusmugavuse juhtnööridele (Apple HIG 2005) ja usutavasti on intuitiivne ning paljude kasutajate ootustele vastav, on lohistamistehnika (*drag & drop)*, mille abil saab elemente hiirega ühest kohast teise paigutada. Kasutusvaldkonnad on seotud näiteks failioperatsioonidega, failide redigeerimisega, kuni tavapärase dialoogiakna omaduseni, mis võimaldab selle tiitliribast hiirega "kinni võtta" ja seda ümber paigutada. Lohistamistehnika on väga kujukas näide otsese manipuleerimise (*direct manipulation*) kontseptsiooni rakendamisest (Sheiderman 1982; Sheiderman 1993; Hutchins, Hollan, Norman 1985)

Mõeldes õpisüsteemide ja testimissüsteemi peale, siis tuleb kõne alla mugavusfunktsioonide rakendamine, mis seotud näiteks testide koostamisel testide valikuga ülesannete basseinist või õpimapi kataloogide järjekorra muutmisega. Võib mainida mitmeid testiliike, kus ilma kõnealuse tehnikata pole praktiliselt võimalik rahuldavaid tulemusi saavutada.

Näiteks IMS testide interoperaabluse standard loetleb 9 testitüüpi (IMS QTI), mis on seotud mingite punktide määramisega kaardil. Seda tüüpi teste on küll võimalik realiseerida ka ilma lohistamistehnikat kasutamata, näiteks kasutades HTML *image map* elemendil määratud piirkondi, luues kaardi lingitavatest pilditükkidest, mis on paigutatud tabelisse, või kasutades vormielementi *input* tüübitunnusega *image*, millel on omadus saata serverisse päringustring, mis sisaldab klikkimise punktile osutavaid x ja y koordinaate. Kuid ükski loetletud lahendustest ei paku kasutajale sellist vahetut tagasisidet ja kontrolli nagu lohistamistehnika.

Teine testiliik, kus on raske kujutada alternatiivi lohistamistehnikale on õigesse järjekorda seadmine. Kui kasutaja ees on segipaisatud sõnad, tähed või pildid ja ta peab need õigesse järjekorda seadma, siis oleks ainsaks alternatiiviks lohistamise kõrval, iga elemendi juures olev sisestusväli, millesse tuleb kirjutada õige järjekorranumber.

Kolmas testiliik on vastavusse seadmise test, milles on tavaliselt esitatud kaks tulpa, mille elementide vahel valitsevate seoste järgi tuleb ühe tulba elemendid seada

vastavusse teise tulba elementidega. Probleemid, mis selle lahendusega kaasnevad on kasutaja ülevaate kadumine, kui vastavusse seatavaid elemente on väga palju. Ülevaade võib kaduda ka lohistamistehnikat kasutades, kuid siis on kõik segatud elemendid kasutajale korraga nähtavad. Teine probleem seisneb selles, et kasutaja võib seada sama asja vastavusse mitmel korral (kui seda pole spetsiaalselt lubatud). Tulemuse kontroll serveri pool ei ole piisav lahendus, kuna kasutaja võib oma testi postitada alles viimasel hetkel ja vea ilmnemisel ületada ajalist piirangut. Kliendipoolse sisestusväljade kontrollimise võimaluse korral tuleks juba eelistada lohistamistehnikat. Kolmas probleem on seotud piltidega. *Select* element ei võimalda seada vastavusse pilte. See pole küll 100% tõsi, kuna näiteks Mozilla toetab *select* elemendi *option* elemendile taustapildi omistamist CSS abil:

Koodinäide 18

```
<select name="test">
<option style="background-image:url(pilt.gif);
height:pildikõrgus;" value="0">Valik 1</option>
<option value="1">Valik 2</option>
</select>
```

Kahjuks on Mozilla kasutajaskond veel liiga väike, et sellele omadusele kogu rakendust rajada (Web Analyst 2005, OneStat 2005, W3School 2005). Pealegi ei parandaks see lahendus ülevaatlikust, ega välistaks sama elemendi korduvat valikut.

IVA-s on kirjeldatud kolmest testitüübist realiseeritud üksnes viimane ja lahendus kasutabki viimases näites kirjeldatud vormi *select* elementi, millest kasutaja peab valima õige vaste.

6.3.1 Lohistamistehnika realiseerimine

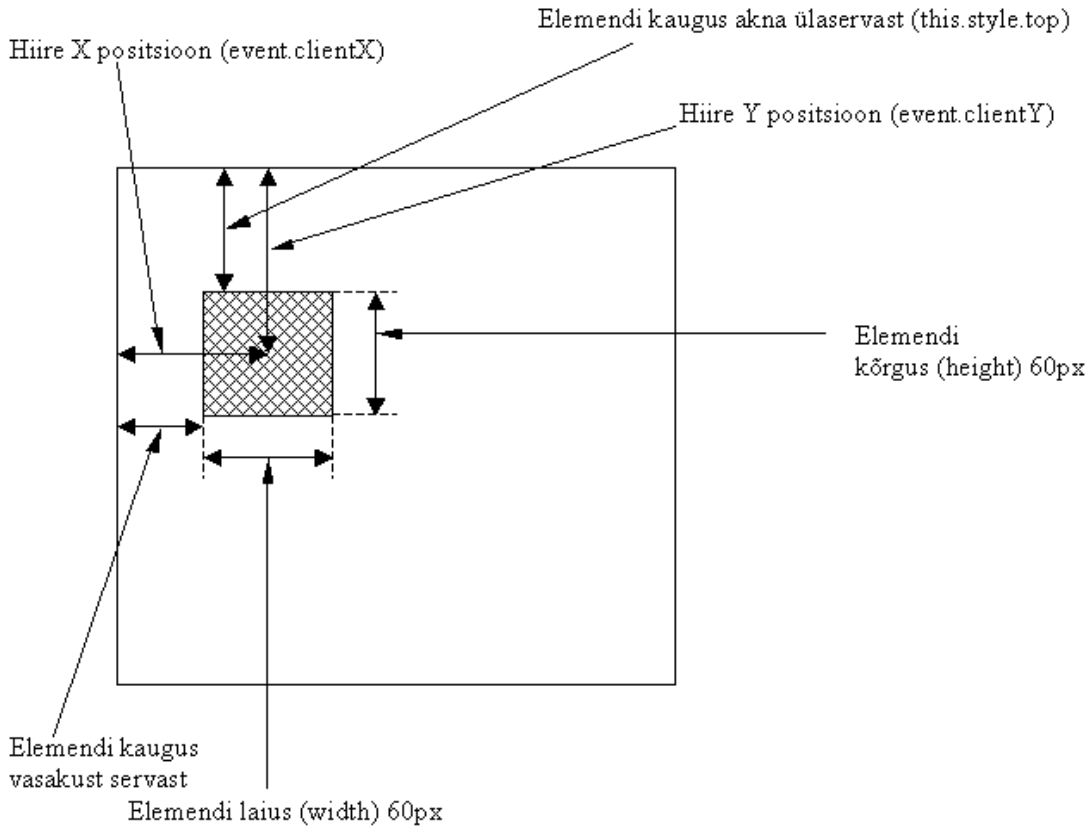
Alates brauserite neljandast põlvkonnast on lehekülje elemente võimalik positsioneerida absoluutsete koordinaatidega sõltumata teistest elementidest. Samuti on võimalik neid elemente liigutada JavaScripti abil. Ka hiirekursori positsioon on tuvastatav. Kombineerides neid kolme omadust, saame tehnika, elemendi liigutamiseks hiire abil. Väga lihtsustatud näide võiks olla järgmine:

Koodinäide 19

```
<div style="position:absolute;background-color:blue;
width:60px;height:60px"
onmousemove="this.style.left=event.clientX-30;
this.style.top=event.clientY-30"></div>
```

Eeltoodud näites (Koodinäide 19) on lisatud sündmust haldav atribuut *onmousemove* elemendi sisse. Kui elemendi kohal toimub hiire liikumine, siis käivitub JavaScripti stsenaarium, mis kirjeldatud jutumärkide vahel. Hiirekursori positsiooni tagastab objekti *event* omadus *clientX* ja *clientY*, mis viitavad vastavalt hiirekursori kaugusele brauseri vasakust servast ja ülaservast. *this.style.left* määrab elemendi vasaku serva kauguse brauseri vasakust servast ning *this.style.top* elemendi ülaserva kauguse brauseri ülaservast. Kui elemendile omistada samad koordinaadid, mis on hiirekursoril, siis paigutub elemendi vasak ülemine nurk hiirekursori järgi. Kuid kuna see piirkond ei ole enam elemendi sees, siis sündmust *mousemove* ja vastavalt ka liikumist enam ei toimu. Seetõttu tuleb elementi hiirekursori suhtes natuke nihutada, nii et hiirekursor jääks siiski elemendi sisse. Antud näites nihutatakse elementi nii, et hiirekursor jääb alati elemendi keskele. Selleks lahutakse hiirekursori x ja y positsioonidest 30 olles eelnevalt *style* atribuudiga määranud elemendi laiuseks ja kõrguseks 60 ja 60 pikselit.

Joonis 15



Tegelikult peaks elementide positsioneerimise korral määratlema ka ühikud, milles positsioneerimine toimub. Mõnede vanema versiooni brauserite puhul ei ole võimalik objekte muidu liigutada. Korrektne süntaks oleks niisiis:

Koodinäide 20

```
this.style.left=(event.clientX-30)+'px';
this.style.top=(event.clientY-30)+'px';
```

Antud lihtsustatud näite tulemusel tekib sinine ruuduke, mis alates hetkest, mil hiir selle kohalt üle libiseb, hakkab hiirega kaasa liikuma. See ei ole veel lohistamine, sest kaasaliikumine toimub kasutaja tahte vastaselt. Tagamaks kasutaja kontrolli, tuleb registreerida lohistamise algus ja lõpp. Kõige ühilduvam[*] on selleks kasutada *mousedown* ja *mouseup* sündmusi, mis registreerivad vastavalt hiireklahvi allavajutuse

[*] Internet Explorer registreerib ka spetsiaalselt lohistamiseks mõeldud sündmused *dragstart* ja *dragend*

ja klahvi lahtilaskmise. Kui hiireklahv on elemendi kohal alla vajutatud, siis alustatakse positsioneerimist hiirekursori järgi ja lõpetatakse positsioneerimine, kui hiireklahv lastakse lahti.

Selleks võib eelolevat näidet täiendada ühe globaalse muutujaga, mis registreerib hiireklahvi olukorra:

Koodinäide 21

```
<script>
var liigub=0; /* hiireklahvi olekut väljendav muutuja */
</script>

<div style="position:absolute;background-color:blue;
width:60px;height:60px"
onmousedown="liigub=1"
onmouseup="liigub=0"
onmousemove="if(liigub==1){
   this.style.left=(event.clientX-30)+'px';
   this.style.top=(event.clientY-30)+'px';
 }">
</div>
```

Eelolevas näites saab juba sõna otseses mõttes objekti võtta ja lohistada. Selle näite juures on veel mitu puudust, mis järgnevalt leiavad lähemat käsitlemist. Põhiline puudus on elemendi korrigeerimine hiirekursori suhtes, et element jääks alati hiirekursori alla. Antud näites on selleks mõlemast hiirekursori koordinaadist lahutanud 30, kuna on teada, et see on pool elemendi laiusest. Kuid alati ei pruugi elemendi laiust teada ning samuti pole suuremate elementide puhul ilus, et hiirekursor paigutub alati elemendi keskele. Rohkem ootuspärane oleks, et element “lukustuks” hiirekursori suhtes just nii nagu ta oli hiireklahvi allavajutamise hetkel. Seega tuleks fikseerida elemendi koordinaatide ja hiirekursori koordinaatide vahe ja kasutada saadud tulemust paigutuse korrigeerimisel:

Koodinäide 22

```
<script>
var liigub=0; /* hiireklahvi olekut väljendav muutuja */
</script>

<div style="position:absolute;background-color:blue;
width:60px;height:60px;left:100px;top:100px"
onmousedown="liigub=1;
kX=event.clientX-this.offsetLeft;
kY=event.clientY-this.offsetTop;"
onmouseup="liigub=0"
onmousemove="if(liigub==1){
   this.style.left=(event.clientX-kX)+'px';
   this.style.top=(event.clientY-kY)+'px';
 }"></div>
```

Ülatoodud näite (

Koodinäide 22) puuduseks on see, et ta ei järgi põhimõtete lahutada semantika, esitlus ja käitumine. Kasutaja seisukohal pole sellel erilist tähtsust, välja arvatud taoliste elementide rohkuse korral, mis suurendaks laadimisaega. Märksa elegantsem on hoida elemendid igasugustest atribuutidest nii puhtad kui võimalik ja omistada neile nii stiil kui sündmused väljastpoolt. Need tehnikad on käsitletud lähemalt peatükis Integreerimise mugavus. Kui tegu on dünaamiliselt kliendi pool genereeritavate elementidega, siis võib elementi loova stringi alusena sellist lähenemist siiski kasutada. Nagu optimeerimise peatükis näeme, on selline meetod ka märksa kiirem, kui standardne DOM meetod elementide loomiseks.

6.4 Dokumendi muutmine redigeeritavaks (WYSIWYG)

Dokumendi redigeerimine on halvasti standardiseeritud valdkond. Tegelikult ei puuduta standard isegi tekstisisestusvälja sümbolikursori (*caret*) haldamist. Siiski on brauseritootjad mõistnud redigeerimisfunktsioonide vajalikkust ning üksteise järel on

need võimalused eri brauseritel kasutusele võetud. Dokumendi redigeeritavaks muutmiseks on kolm põhilist moodust:

- dokumendi *designMode* omaduse muutmine,
- vastava komponendi kasutamine ja
- elemendi *contenteditable* atribuudi rakendamine.

Alternatiivseid katsetusi on tehtud, kuid need ei ole rahuldavaid tulemusi andnud. Kolm töötavat võimalust tulevad järgnevalt käsitlemisele.

6.4.1 Designmode

Dokumendi viimine redigeeritavasse olukorda, mille juhatas sisse Microsoft oma brauseriga Internet Explorer 4 on tavaliselt tuntud seoses `iframe` elemendi redigeerimisega. Tegelikult pole sellel `iframe` elemendiga mingit muud seost, kui et `iframe` on tihti kõige sobilikum element nimetatud tehnika rakendamiseks. Redigeeritavaks võib muuta ka kogu lehekülje või `frameset` elemendi osa. Kõikide nende kolme võimaluse puhul on tegu iseseisva dokumendi muutmisega.

Et muuta dokument redigeeritavaks `designmode` omaduse (*property*) abil, võib näiteks lisada dokumendi *body* elemendi sisse sündmusehalduri *onload* ja vastava käsu:

Koodinäide 23

```
<body onload="document.designMode='on'">
```

Seejärel muutub tekst ja kõik objektid leheküljel redigeeritavaks ning sinna on võimalik kopeerida sisu kopeeri-aseta (*copy-paste*) meetodil. Nagu sündmuste haldurite lisamise juures on ka siin võimalusi rohkem, kui *body* elemendi atribuudi kasutamine. Sündmuste käsitlemine ehk kliendipoolse käitumise kontroll (EcmaScript), lehekülje semantika (XHTML) ja lehekülje esitus (CSS) peaksid olema

lahutatud. Seega soovitavaks saab pidada redigeeritavuse määramist välise js faili abil, mis sisaldab samaväärset rida. Näiteks:

Koodinäide 24

```
window.onload=function(){
  document.designMode='on';
}
```

Eelnevas näites (Koodinäide 24) omistatakse redigeerimisomadus tervele leheküljele siis, kui see on laetud. Kuna tavaliselt tahetakse vaid teatud osa lehest muuta redigeeritavaks asendamaks näiteks *textarea* elementi, siis sobib selleks hästi `iframe` element. Üks võimalus `iframe` elemendi sisu redigeeritavaks muutmiseks on laadida seal sees dokument, mis on ühel eelmainitud viisidest (koodinäited 6 ja 7) redigeeritavaks muudetud. Teine ja parem lahendus on omistada redigeeritavus peamise (*parent*) dokumendi kaudu (Koodinäide 25):

Koodinäide 25

```
<script>
window.onload=function(){
  parent.frames['iframe_nimi'].document.designMode='on';
}
</script>
<body><iframe src='redigeeritav.html' name='iframe_nimi'
id='iframe_nimi' style='width:500px;height:250px'></iframe></body>
```

Kui vaadata Mozilla Midas projekti (Midas 2003) spetsifikatsiooni, siis soovitatakse seal kasutada natuke teistsugust pöördumist `iframe` elemendi poole, et mitte kasutada `name` atribuuti ja kollektsiooni `frames` poole pöördumist.

Koodinäide 26

```
window.onload=function(){
  redaktoriaken=document.getElementById('iframe_id').contentWindow;
  redaktoriaken.document.designMode='on';
}
```

Kahjuks ei tööta viimane näide (Koodinäide 26) Iinernet Explorer 5 brauseril, mille turuosa võib hetkel moodustada veel kuni 5% . Kuna eelneva (Koodinäide 25) *frames* kollektsiooni poole pöördumine töötab kõikide brauserite puhul, siis on tülikate tingimuslausete vältimiseks mõistlik kasutada seda.

6.4.2 DHTML (MSHTML) Edit komponent

Alapealkirjas mainitud nimega komponendi võttis kasutusele Microsoft samuti juba Internet Explorer 4 aegadel, kuid siis puudus sellele vee brauseri loomulik tugi ja kasutamiseks tuli vastav komponent eraldi installeerida. Alates versioonist 5, on brauserile DHTML edit komponent sisseehitatud. Selle Kasutamiseks tuli lisada leheküljele *object* element vastava *classid* atribuudiga (Koodinäide 27):

Koodinäide 27

```
<OBJECT
   CLASSID="clsid:683364AF-B37D-11D1-ADC5-006008A5848C"
   ID="DHTMLEdit"
   HEIGHT="500"
   WIDTH="250">
</OBJECT>
```

Loodud objekt on iframe sarnaselt *iframe* elemendile redigeeritav ning sinna saab kopeerida sisu teistest dokumentidest. Erinevalt *iframe* elemendist, mis moodustab iseseisva dokumendi objekti, saab *DHTMLEdit* objekti poole pöörduda otse nagu lehekülje iga muu objekti poole.

6.4.3 Contenteditable atribuut

Alates versioonist 5,5 võttis Internet Explorer kasutusele uue leheküljesisese redigeerimistehnika kasutades spetsiaalselt selleks loodud *contenteditable* atribuuti. Igale elemendile leheküljel võib selle atribuudiga omistada redigeerimisvõimaluse:

Koodinäide 28

```
<div id="redaktor" contenteditable="true">
siin on redigeeritav sisu
</div>
```

6.4.4 Milline redaktor on parim

Nagu näha statistikast (W3Shools 2005), on veel mingi hulk kasutajaid, kelle brauseriks on Internet Explorer 5. Windows 98SE paketti oli lisatud Internet Explorer 5, varasemal Windows 98'l isegi Internet Explorer 4. Windows 2000 oli alguses varustatud brauseriga Internet Explorer 5.01. Kuigi need brauseriversioonid on väga ebaturvalised ja nende kasutamist ei peaks propageerima, ei saa kasutajat muuta. Kuid muuta ja kohandada saab oma teenust kasutaja järgi. Kõik kasutajad ei tegele pidevalt oma süsteemi uuendamise ja turvaaukude parandamisega ja sõltuvalt kasutusharjumustest ei pruugi ka nende ebaturvaliste brauserite kasutamisel midagi halba juhtuda.

Ühesõnaga valik langeb vanimale neist redigeerimisvõimalustest: `designMode` omaduse kasutamisele.

Valikut on lihtsustanud ka muud arengud brauseriturul. Käesoleva töö kirjutamise alguses aasta 2003 sügisel, oli lisandunud veel üks värske WYSIWYG tehnoloogia toetaja: alates versioonist 1.3 beta (26.01.2003) implementeeris Mozilla dokumendi `designMode` omaduse (Midas 2003). Väikeste muutustega on enamus Internet Exploreri skriptijale tuntud `execCommand` meetodi käskudest üle võetud.

Mozilla Firebird ja hilisem Firefox toetavad samuti neid redigeerimistehnikaid. Juba siis oli teada, et sarnane arendustöö on käimas ka KHTML mootoril töötava Konqueror ja Safari brauseritega ning alates versioonist 1.3 toetab ka Safari WYSIWYG redigeerimist nii `iframe` elemendi `designmode` omaduse abil kui ka `contenteditable` atribuuti kasutades. Kuigi nende võimaluste kohta puudub veel

korralik dokumentatsioon, näib, et mõlemad võimalused ühilduvad Internet Exploreri omadega.

Opera kasutajate kogukonnas tekkis kohe peale Mozilla Midas projekti realiseerumist küsimus, kuidas reageerib sellele Opera. 2003 aprillis algatati selleteemaline lõim Opera kasutajate foorumis (Opera Forum 2003), kuid Opera ei andnud kuni viimase ajani kinnitust selle kohta, kas `designMode` omadus või `contenteditable` atribuut kuuluvad nende tulevikuplaanidesse. Seetõttu seisis veel käesoleva töö augusti 2005 redaktsioonis järgmine rida:

Opera pole mingi moel kinnitanud WYSIWYG redigeerimise toetuse sisseviimist. Ka hetke viimase Opera 8 versiooni puhul on lahendamata isegi textarea või input elemendis selekteeritud teksti muutmise võimalused.

Kuid üllataval kombel realiseeris ka Opera versioonis 9 WYSIWYG redigeerimise ühilduvalt Internet Exploreri `designmode` ja Mozilla Midas projektiga. Surve, mida avaldasid teenuste loojad ja kasutajad, osutus tõenäoliselt väga suureks.

Seega WYSIWYG redigeerimine, mis oli algselt vaid Internet Exploreri ja Windows platvormi eelis, on muutunud de facto standardiks. Loodetavasti saab sellest kunagi ka de iure standard, kuigi arendajate ja kasutajate seisukohast on alati tähtsam see, kui asi töötab ning töötab ühte moodi.

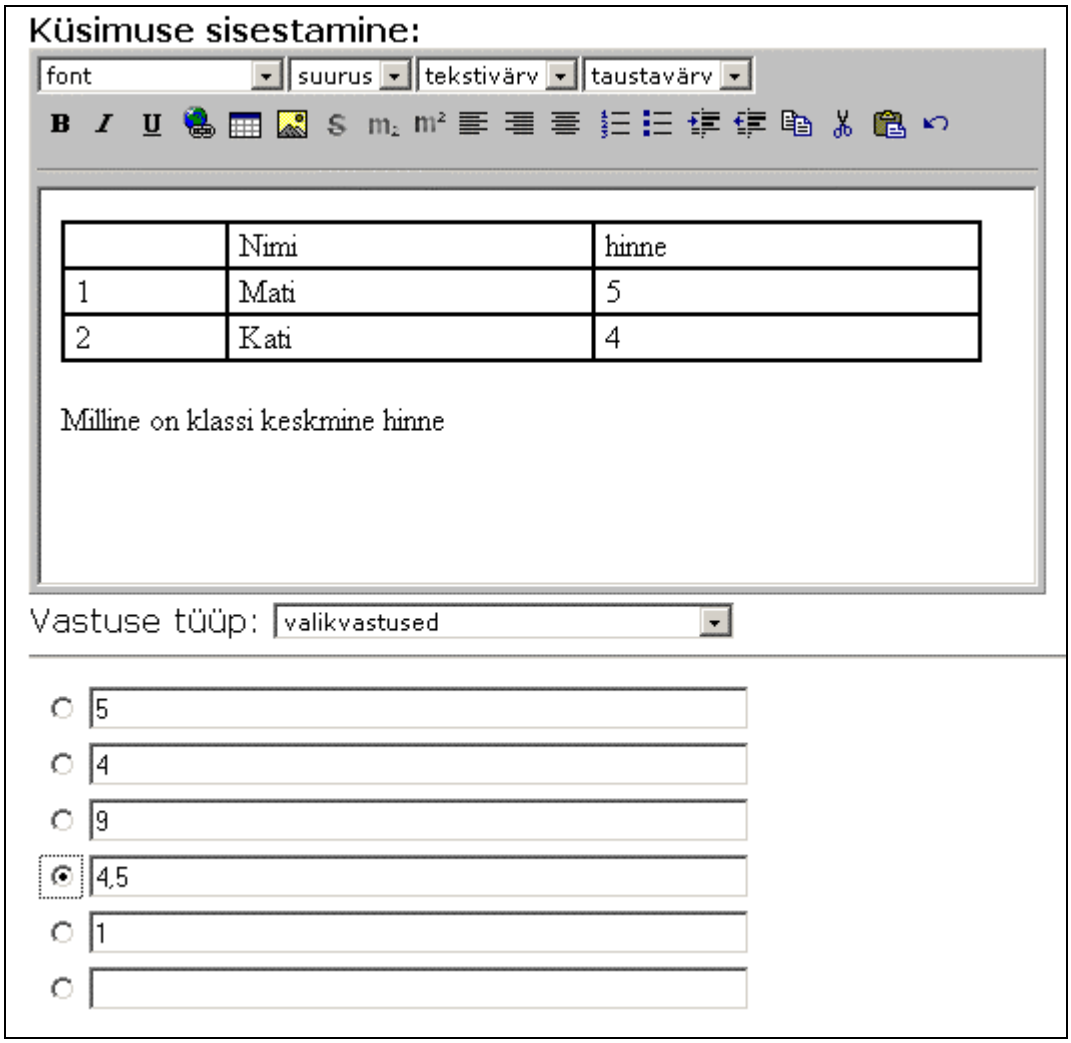
Ei tea, milline WYSIWYG meetod lõpuks võidab. *Contenteditable* on lihtsam ja võimalusterohkem, seega suurema potentsiaaliga. Kuid praegu kalduvad veel brauserite suurema toetuse tõttu eelistused `iframe` elementidel baseeruva redaktori poolele.

7 Esimesed tulemused

7.1 WYSIWYG redaktori realiseerimine IVA testimissüsteemis aastal 2004

Aasta 2004 suvel kirjutasin IVA testimissüsteemi jaks WYSIWYG redaktori prototüübi, mis on näha järgmisel pildil (Pilt 7):

Pilt 7



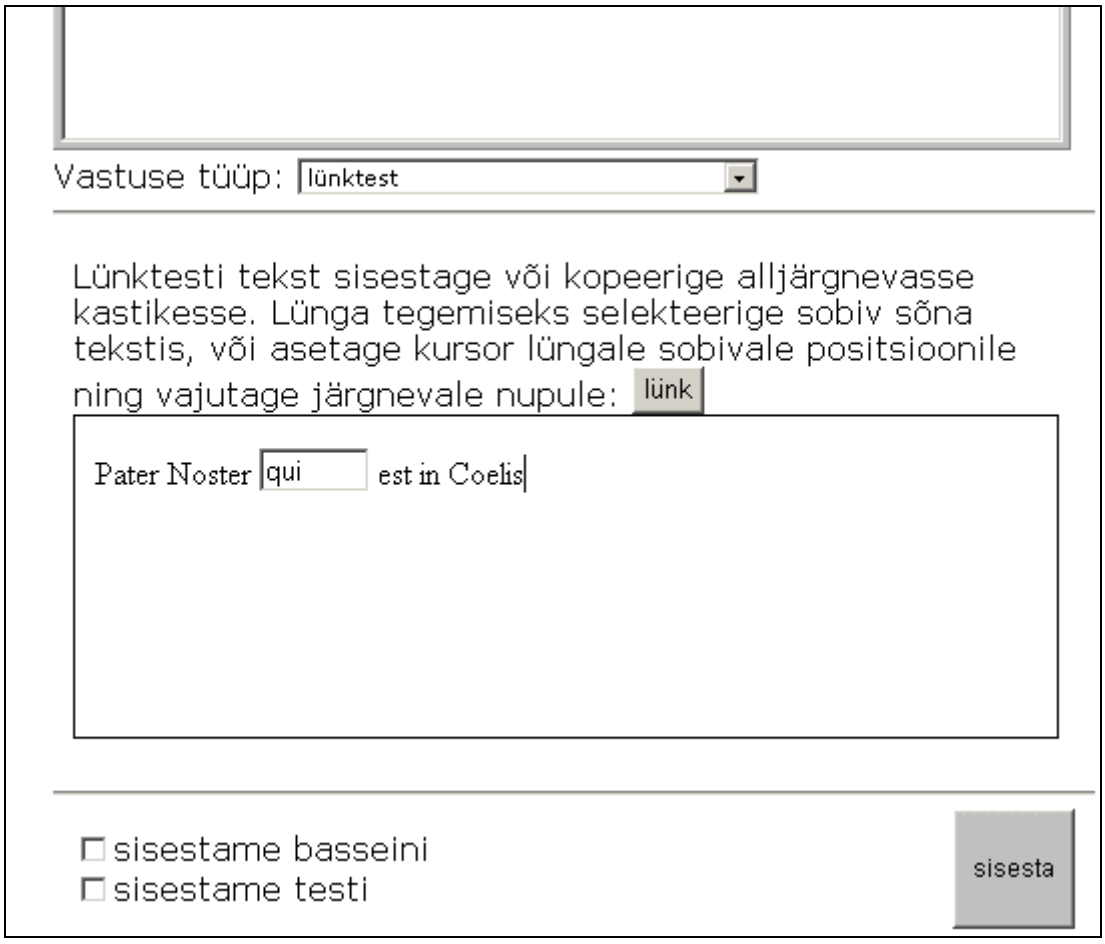
Redaktoriosa on praeguseni IVA testimissüsteemis kasutuses. Nüüd olen sellele kirjutanud mitmeid täiendusi, mis on seotud integreerimise mugavusega ning töötanud välja põhimõtted mitmeaknalise redaktori loomiseks, mis kasutaks ühist tööriistariba, ning mille abil saab redigeerida ka valikvastuseid või muid küsimuse liike. Loodud

süsteemis toimub vormielementide asendamine automaatselt, lehele tuleb lisada vaid `script` element viidaga js failile.

7.2 Lünktesti loomise kasutajaliides

Koos esimese IVA testide WYSIWYG redaktoriga tegin ka samal tehnikal baseeruva lünktestide koostamise prototüübi (Pilt 8). Lüngaks osutuv sõna tuleb selekteerida ning vajutada lünga tegemise nupule, mille tulemusel avanevas dialoogis võib sisestada alternatiivseid sõnu. Lüngaks oleva sõna kohale tekitati kohe tekstiväli, et tulemus oleks visuaalselt sarnane lõpptulemusele, mida õpilane täitma hakkab.

Pilt 8



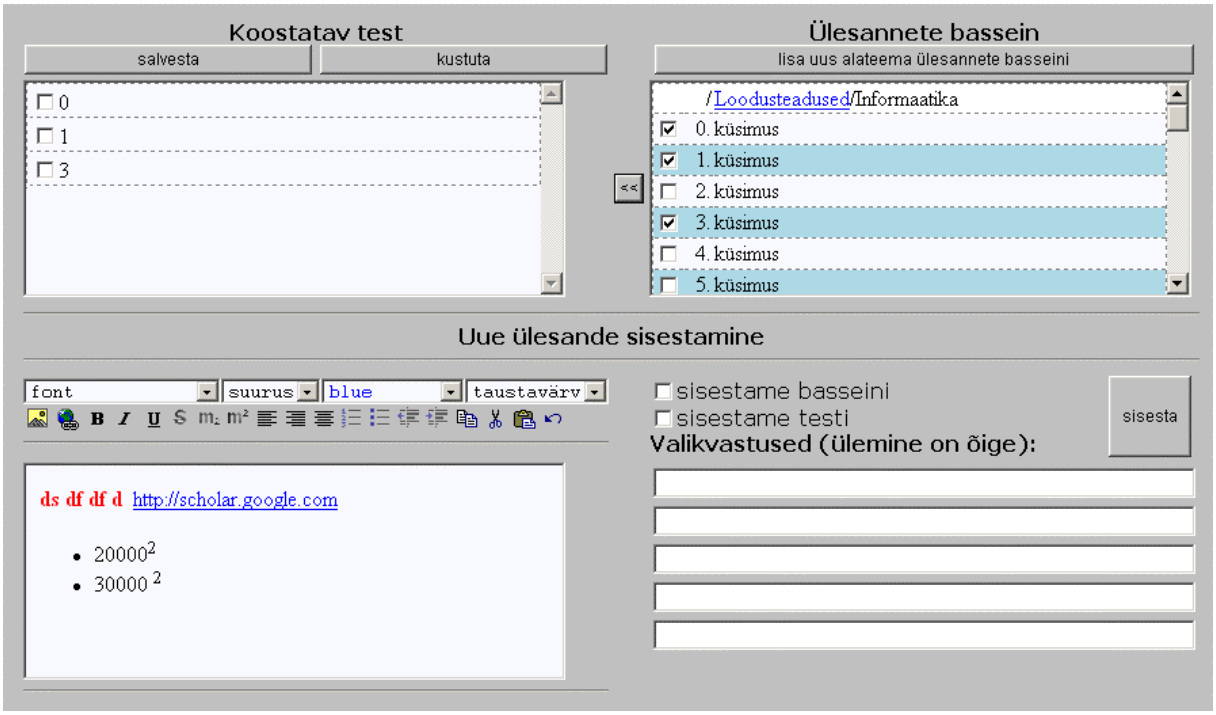
Lünktest pole IVA testimissüsteemis praeguseks realiseerunud. Ühest küjest kindlasti sellepärast, et sellist küsimusetüüpi IVAs ei olnud ka varem. WYSIWYG redigeerimise kasutuselevõtt ei eeldanud kogu süsteemis suuri muudatusi, sest oli ja on

praegugi pigem olemasoleva süsteemi täiendus. Mis puutub aga lünktesti, siis ühest küljest saab seda realiseerida ka mitme lühivastusega küsimusetüübi kogumina. Kuid arvestades küsimuse atomaarsust, on küsitav, kas sellistel üksikutel lühivastusega küsimustel säilib piisavalt konteksti, et omandada tähendust ka eraldiseisvate osadena. Arvestades küsimuste basseini põhimõtet, mis on IVA süsteemis realiseeritud, kujutaksid sellised lünktesti osad endast tähenduseta fragmente, mille kombineerimine uutesse seostesse ja kasutamine uutes testides oleks raske.

Tookord aastal 2004 ülejäänud küsimusetüüpide juures mingeid täiendusi antud prototüübis võrreldes IVA-ga ei olnud ja seetõttu ma neil pikemalt ei peatu. Kaalumisel oli küll juba siis pakkuda ka valikvastuste ja muude küsimuste puhul samuti WYSIWYG redigeerimise võimalust ja uue küsimusetüübina järjekorda seadmise küsimustetüübi realiseerimist.

Veel varasemas prototüübis püüdsin paigutada ühele lehele loogiliselt kokkukuuluvad tegevused nagu küsimuste valimise testi jaoks ja küsimuste ise kirjutamise. Seda sai siiski ekraani jaoks liiga palju nagu näha järgnevalt pildilt (Pilt 9):

Pilt 9



Selle ilmselt ülekoormatud paigutuse algne mõte oli vältida päringute arvu, mis kuluks erinevate vormide vahel navigeerimiseks. Peamine eesmärk oli tegelikult vältida ootamist, mis tuleneb serverisse tehtavatest päringutest. Vältima peaks nii üht kui teist, nii ülekoormatust kui liigseid päringuid serverisse.

7.3 Järjekorda seadmise ülesannete õpilasekeskkonna loomine.

Järjekorda seadmise küsimuse juures esitatakse testi lahendajale juhuslikus või määratud järjekorras elemendid, mis tuleb seada loogilisse järjekorda. Elementideks võivad olla sõnad, tähed, pildid või kombinatsioon neist kõigist. Milline saab olema konkreetne elemendi sisu, see sõltub eelkõige testi loomise keskkonna võimalustest, kuna väga suur osa neis probleemidest leiab käsitlemist WYSIWYG redigeerimist puudutavates peatükkides, siis hõlmab seal loodav lahendus universaalselt kõiki küsimusetüüpe, mille puhul küsimuse koostaja loob mitmeid valikuid. Lahtrid valikvastuse, õigete vastusevariantide märkimise või järjekorda seadmise valikute sisestamise jaoks ei erine teineteisest tehniliselt.

Mis puutub järjekorda seadmise ülesannetesse, siis sellist küsimusetüüpi ei ole praeguses IVA versioonis veel realiseeritud. Pole ka ime, sest seni on IVA esindanud peamiselt õhukese kliendi tehnoloogiat, millega on sellele küsimusetüübile väga raske leida alternatiiv. Ainus mõeldav alternatiiv tundub olevat elementide järjekorda seadmine nende märgistamisel järjekorranumbriga. Iga elemendi juurde tuleks teha lahter, kuhu kasutaja märgib numbri, mis vastab tema arvates selle elemendi õigele järjekorranumbrile. Sisestusvälja asemel võib kasutada ka valikmenüüd `select`. Kuid sellist lahendust ei saa kindlasti pidada intuitiivseks ega mugavaks.

Kuna minu poolt pakutav versioon on mõeldud töötama JavaScriptis realiseeritud Drag & drop tehnoloogial, siis oleks lihtsaim viis saata kliendi juurde vajalikud andmed JavaScripti massiivis. Siiski teen ülesande natuke keerulisemaks, pakkudes töötavat versiooni ka kasutajatele, kellel on JavaScript välja lülitatud või on brauseri toetus selleks liiga nõrk. Selleks alustan lihtsustatud HTML prototüübist, mis esindab selle küsimusetüübi kliendikeskkonda. Seejärel loon JavaScripti protseduuri, mis otsib üles dokumendist sisalduvad järjekorda seadmise elemendid ning loob neist lohistatavad

elemendid. Samas protseduuris peidan elementide juures paiknevad järjekorra manuaalseks sisestamiseks mõeldud sisestusväljad. Lõpuks loon kontrollfunktsiooni, mis käivitub iga lohistamise lõppedes ning mis kontrollib elementide järjekorda ja vastavalt sellele kirjutab järjekorranumbrid automaatselt peidetud vormielementidesse. Kui test on täidetud ja tulemused postitatakse serverisse, siis toimub selle küsimusetüübi andmete postitamine peidetud väljadest traditsioonilisel viisil.

HTML lehe struktuuris võib kasutada järjekorda seatavate elementide esitamiseks erinevat struktuuri, kuid nad peavad olema arusaadavalt tähistatud. Antud näites on tähistuse eesmärgil lisatud `class` atribuut väärtusega järjekorda. Objektide esitamiseks sobivad kõige paremini HTML elemendid `ul` ja `table`. Antud näites kasutan elementi `table` ja paigutan objektid horisontaalselt ühte ritta. Sisestusväljad paigutan teise ritta iga objekti alla. Annan aru, et lahenduse kood ei ole semantiline, kuna pole päris üheselt mõistetav, milline järjekorralahter kuulub millise objekti juurde. Arusaadavam oleks, kui objektid paikneksid teineteise all ja lahter neist vasakul või paremal. Veel mõistetavam oleks kui järjekorralahter paikneks sama elemendi sees, kus paikneb lohistatav objekt. Kõige semantilisem lahendus oleks hoopis `ul` elemendi kasutamine, mille alamelemendid `li` sisaldaksid sisestusvälja ja paigutatavaid objekte. Tõenäoliselt tuleb seda viimast võimalust ka rakendada lõplikus versioonis, antud prototüübi eesmärk on tagada süsteemi ülevaatlikkus ja visuaalne arusaadavus. Selleks lisan visuaalse näitlikkuse huvides ka tabeli raami. Lehekülje lihtsustatud HTML struktuur oleks järgmine:

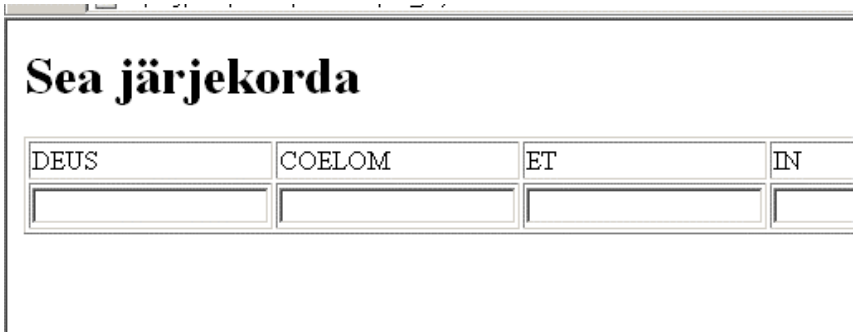
Koodinäide 29

```
<script src="sea_jarjekorda.js"></script>
<style>@import url(sea_jarjekorda.css);</style>
<h2>Sea järjekorda</h2>
<table class="jarjekorda" border="1"><tbody><tr>
<td>DEUS</td>
<td>COELOM</td>
<td>ET</td>
<td>IN</td>
<td>CREAVIT</td>
<td>PRINCIPIO</td>
<td>TERRAM</td>
```

```
</tr><tr>
<td><input name="y3"></td>
<td><input name="y3"></td>
<td><input name="y3"></td>
<td><input name="y3"></td>
<td><input name="y3"></td>
<td><input name="y3"></td>
<td><input name="y3"></td>
</tr></tbody></table>
```

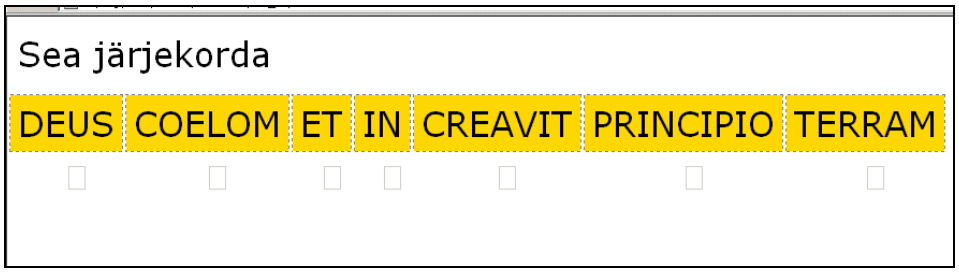
JavaScripti ja CSS-ita koodi põhjal moodustub järgmise struktuuriga kogum (Pilt 10):

Pilt 10



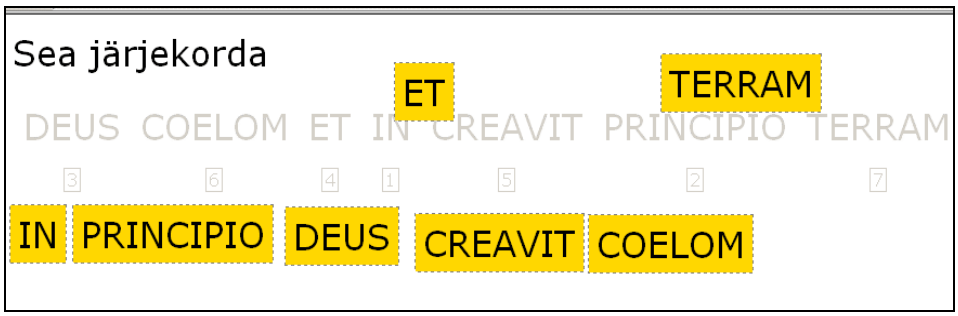
Lisades CSS faili võib muuta elementide suurusi, kindlasti tuleb lühendada *input* elementide laiust või teha seda elementide *size* atribuudi abil. Käesolevas näites kasutatav CSS fail teenib hetkel vaid lohistamistehnika lisamise huve ja jätab vaid veidi esialgsest struktuurist nähtavale. Eesmärgiks on saavutada olukord, kus kõik tabeli lahtrites esinevad objektid (antud näites sõnad, kuid sama hästi võib olla ka segu piltidest ning HTML-ist) kaetakse kinni oma lohistatavate kloonidega. Laetav lehekülg näeks peale JavaScripti lisamist välja nagu järgmisel ekraanipildil (Pilt 11):

Pilt 11



Nagu öeldud on tegu vaid olemasolevate objektide kinnikatmisega ning soovi korral võib algse järjekorra ja ka sisestuslahtrid alles jätta. Näitefailis teenivad need lahtrid ka kontrollivat eesmärki, näitamaks, et tõepoolest õige järjekorranumber vastava elemendi alla lisatakse.

Pilt 12



Saavutamaks esitatud olukorda, tuleb alustada lohistatavate objektide loomiseks. Selleks tuleb läbi käia kogu test, ja otsida tabeleid (või muid kokkuleppelisi elemente), mille `class` atribuudi väärtus on `jarjekorda`:

Koodinäide 30

```
vjtbl=document.getElementsByTagName('table'); // kõik tabelid
for(vst=0;vst<vjtbl.length;vst++){  // tuleb läbi käia
  if(vjtbl[vst].className=='jarjekorda'){ // ja otsida tähistust
```

Kuna antud näites asuvad järjestatavad objektid tabeli esimeses reas, siis järgmiseks sammuks on iga leitud tabeli juures läbi käia esimese rea kõik lahtrid:

Koodinäide 31

```
read=vjtbl[vst].rows;  // leitud tabeli kõik read
lahtrid=read[0].cells; // esimene rida
lhtrSisu=[];           // massiiv lahtrite sisu jaoks
kontroll=document.createElement('div'); // element objektidele
kontroll.id='kntr'+vst;   // unikaalne nimi elemendile

 for(i=0;i<lahtrid.length;i++){ // käime lahtrid läbi
```

Iga lahtri puhul tuleb lugeda selles sisalduv kood muutujasse (andud näites massiiv) ja luua sama koodi sisaldav lohistava objekt. Loodavad objektid võib paigutada erineval viisil, antud näites paigutan need täpselt samale kohale, kus nad asusid tabelis. Selleks talletan ka tabeli lahtrite x ja y koordinaadid:

Koodinäide 32

```
sv=lahtrid[i].offsetLeft; // lahtri x positsoon
sy=vjtbl[vst].offsetTop;  // lahtri y positsioon
lhtrSisu[i]=lahtrid[i].innerHTML; // lahtrisisu
```

Kuna lohistatavate objektide positsioon peab olema absoluutne, siis tuleb säilitada vana tabeli olemasolu testi struktuuris, et reserveerida ruumi järjekorda seadmise ülesande sooritamiseks. Nähtavus ei ole sel puhul oluline. Lohistavate objektide loomine vastab põhimõtteliselt juba seda tehnikat tutvustavas peatükis (6.3.1) käsitletule, kuid lisada tuleb pöördumine protseduuri *jrk()* poole, mis käivitub lohistatava objekti paigale asetamisel ja tuvastab objektide hetkejärjekorra, ning märgib selle lahtrite all paiknevatesse sisestusväljadesse:

Koodinäide 33

```
lohel= '<span id="e'+vst+'_'+i+'" style="'
  +'position:absolute;left:'+sv+'px;top:'+sy+'px;padding:7px; '
  +'border:dotted gray 1px;'
  +'font:2em verdana;cursor:move;background-color:gold;'
  +'-moz-user-focus:ignore;-moz-user-select: none;" '
  +'onselectstart="return false;" '
  +'onmousedown="this.style.zIndex=\'100\';liigub=1;" '
  +'onmouseup="liigub=0;this.style.zIndex=\'1\';jrk('+vst+');" '
  +'onmousemove="if(liigub==1){ '
  +'this.style.left=skrollX()+'
  +'(event.clientX-(this.offsetWidth/2))+\'px\'; '
  +'this.style.top=skrollY()+'
  +'(event.clientY-(this.offsetHeight/2))+\'px\';}">'
  +lhtrSisu[i]+'</span>';

 kontroll.innerHTML+=lohel; //lisame loodud objekti kontrollelementi
```

Seejärel lõpeb käesoleva tabeli lahtreid läbiv kordus ning dokumendistruktuuri lisatakse objekt `kontroll`, mis omakorda sisaldab lohistatavaid objekte. Et tagada õige asukoht, selleks on sobiv kasutada `insertBefore` meetodit lisades kontrollobjekti käesoleva tabeli ette. Tähtis on seejuures arvestada antud näitefailis erinevate olukordadega, kus käsitletav tabel võib asuda mingi muu elemendi sees:

Koodinäide 34

```
vanem=(vjtbl[vst].parentElement)?'Element':'Node';
vjtbl[vst]['parent'+vanem].insertBefore(kontroll,vjtbl[vst]);
```

Sellega on lohistatavad objektid ekraanile kuvatud. On vaja veel teatud korrigeerivaid lisafunktsioone ja protseduuri objektide järjekorra arvestamiseks. Korrigeerivad funktsioonid on seotud elemendi paigutamisega ja on vajalikud siis kui tegu on pika leheküljega, mille puhul on vajalik kerimine. Kui lehekülge on keritud, siis ei tagasta hiirekursori positsioonide arvutamiseks kasutatud `clientX` ja `clientY` enam neid koordinaate, mida vaja. Hiirekursori koordinaadid antakse brauseriakna ülanurga suhtes, mitte dokumendi ülanurga suhtes ja lehekülje kerides on need väärtused erinevad. Kuna DOM ei ole selles suhtes väga konkreetne, siis ei saa brauseritootjaid süüdistada puudulikus implementatsioonis.

Kõik tuntumad brauserid peale Internet Exploreri toetavad ka mittestandardset `pageX` ja `pageY` omadust ja nende kasutus annab antud ülesande jaoks just vajalikud koordinaadid. Internet Exploreri jaoks tuleb leida siiski mingi muu võimalus ja selleks on käepärased dokumendi `scrollTop` ja `scrollLeft` omadused. Neid omadused on toetatud ka muude brauserite poolt, mis jäävad käesolevas töös toetatavate brauserite hulka. Niisiis saab tuvastada hiire positsioonid, liites neile järgmiste funktsioonide tagastatavad väärtused:

Koodinäide 35

```
function skrollX(){ return document.body.scrollLeft||0;}
function skrollY(){ return document.body.scrollTop||0;}
```

Siiski on üks erijuht, mille puhul ka järgmised funktsioonid ei tööta. Nimelt on probleemseks brauseriks Internet Exploreri juhul kui lehele on lisatud DOCTYPE deklaratsioon. Vastavalt sellele on *scrollTop* ja *scrollLeft* kas *documentElement* omadus või *body* omadus. Seetõttu tuleb antud funktsiooni modifitseerida järgmiselt:

Koodinäide 36

```
function skrollX(){
return document.documentElement.scrollLeft||document.body.scrollLeft||0;
}
function skrollY(){
return document.documentElement.scrollTop||document.body.scrollTop||0;
}
```

Järgnevalt järjekorra arvutamise protseduurist. Funktsiooni argumendiks on muutuja, mis tähistab kontrollitava järjestamisülesande indeksit. Selle indeksi abil saab pöörduda kontrollobjekti poole käia läbi kõik lohistatavad objektid, mis selles asuvad. Vaatamata sellele, et lohistatavaid objekte võib lehe piires igale poole paigutada, jäävad nad dokumendi struktuuris endiselt loodud kontrollelemendi alamelementideks.

Koodinäide 37

```
function jrk(vt){
kntrElement= document.getElementById('kntr'+vt);
sElm = kntrElement.getElementsByTagName('span');
```

Seejärel refereerin käesolevas tabelis olevate *input* elementide kollektsioonile. Meeldetuletuseks nii palju, et need *input* elemendid olid järjekorra numbrite märkimiseks.

Koodinäide 38

```
tblElement = inp=document.getElementsByTagName('table')[vt];
inp= tblElement.getElementsByTagName('input');
```

Peale selle tuleb algväärtustega defineerida mitmed muutujad, mille kirjelduse hetkel vahele jätan ja käsitlen protseduurides, kus need kasutusse tulevad. Järgneb kõikide lohistatavate objektide läbikäimine korduslausega, milles registreerin kõigepealt elemendi kauguse brauseriakna vasakust servast ja omistan selle massiivi *kaugused[]* vastavale liikmele. Samamoodi saab väärtuse massiivi *nimi[]* vastav liige ja selleks väärtuseks korduslause käesolev indeks:

Koodinäide 39

```
for(i=0; i<sElm.length; i++){
 kaugused[i]=sElm[i].offsetLeft;
 nimi[i]=i;
```

Massiivi *kaugused[]* olemasolu on vajalik objektide järjekorra võrdlemiseks, massiivi *nimi[]* olemasolu vajab ehk lähemat käsitlemist. Tegelikult väljendavad mõlemad massiivid sama liigutatava objekti kahte omadust. Üks neist on elementide x koordinaatide massiiv, mis sorteeritakse järjekorda ning sama sorteerimise ajal muudetakse ka massiivi nimi elementide järjekorda vastavalt. Seega kui algselt esitatud objektidest element järjekorraindeksiga 3 paigutada esimeseks, siis saab massiivielement *nimi[0]* väärtuseks 3.

Koodinäide 40

```
if(i>1){   /* kuna võrdleme eelmist elementi, käesolevaga, siis
alustame sorteerimist teisest elemendist */
   j=i;
     while(j>0){ /* liigume tagasi kuni esimese elemendini */
      if(kaugused[j] < kaugused[j-1]) {  /* kui vaadeldav
            element asub vasakule servale lähema kui eelmine,
            siis tuleb vahetada elementide kohad */
        ajutine=kaugused[j-1]; /* abimuutuja vahetuse
                                  sooritamiseks */
        kaugused[j-1]=kaugused[j]; /* eelmine element saab
                        väärtuseks, vaadeldava elemendi väärtuse */
        kaugused[j]=ajutine; /* vaadeldav saab eelmise väärtuse */
        ajutine=nimi[j-1]; /* samamoodi sünkroniseerime massiivi */
        nimi[j-1]=nimi[j]; /* nimi[] elemendid */
        nimi[j]=ajutine;
     } // if lõpp
          --j; /* vähendame muutujat j ühe võrra */
```

```
   }     /* ehk liigume eelmise elemendi juurde */
} // while lõpp
```

Sorteerimise kasulikuks tulemuseks on massiiv *nimi[]*, mille elementide väärtused osutavad paigutatavate objektide indeksile dokumendistruktuuris, ehk nende algsele järjekorrale, samal ajal kui massiiviindeks näitab objekti nähtavat asukohta järjekorras vasakult paremale. Kokkuvõttes saame funktsiooni *jrk()*, mis paigutuse paremalt vasakule kindlaks teeb ja sisestuselementidesse lisab (Koodinäide 41):

Koodinäide 41

```
function jrk(vt){
kntrElement= document.getElementById('kntr'+vt);
sElm = kntrElement.getElementsByTagName('span');
tblElement = inp=document.getElementsByTagName('table')[vt];
inp= tblElement.getElementsByTagName('input');
kaugused=null;
nimi=null;
kaugused=[];
nimi=[];
ajutine='';
kaugused[0]=0;
  for(i=0;i<sElm.length;i++){
   kaugused[i]=sElm[i].offsetLeft;
   nimi[i]=i; //sElm[i].id;
       if(i>1){
         j=i;
         while(j>0){
          if(kaugused[j] < kaugused[j-1]) {
       /* sorteerime elemendid kaugused x koordinaadi järgi */
           ajutine=kaugused[j-1];
            kaugused[j-1]=kaugused[j];
            kaugused[j]=ajutine;
            ajutine=nimi[j-1];
            nimi[j-1]=nimi[j];
           nimi[j]=ajutine;
          }
          --j;
         }
```

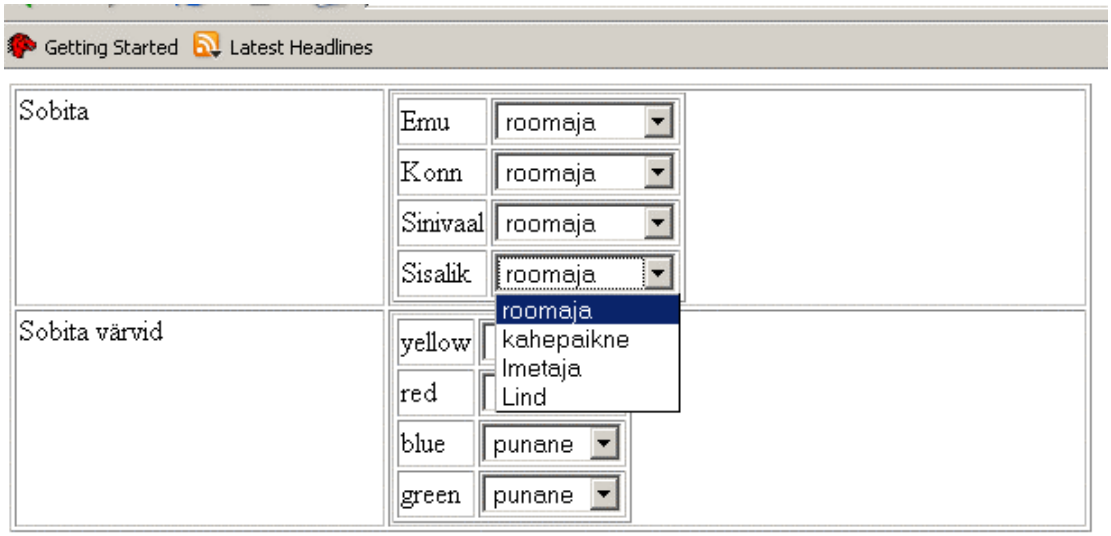
```
        }
  }
   for(i=0;i<inp.length;i++){
     if(inp[i].name.charAt(0)=='y'){
      inp[nimi[i]].value=(i+1);
     }
   }
}
```

Järjekorda seadmise ülesannet võib veel edasi arendada, et kontrollfunktsioon suudaks tuvastada ka järjekorda ülevalt alla või mitmerealiselt paigutatud elementide puhul. Samuti võib arendada edasi teatud mugavust lisavaid tehnikaid nagu lohistatavate objektide sees teksti selekteerimise keelamine või sündmuste püüdmine lohistatavasse objekti, et vältida lohistava objekti võimalikku mahajäämist väga kiirete hiireliigutuste korral.

7.4 Vastavusülesannete modifitseerimine

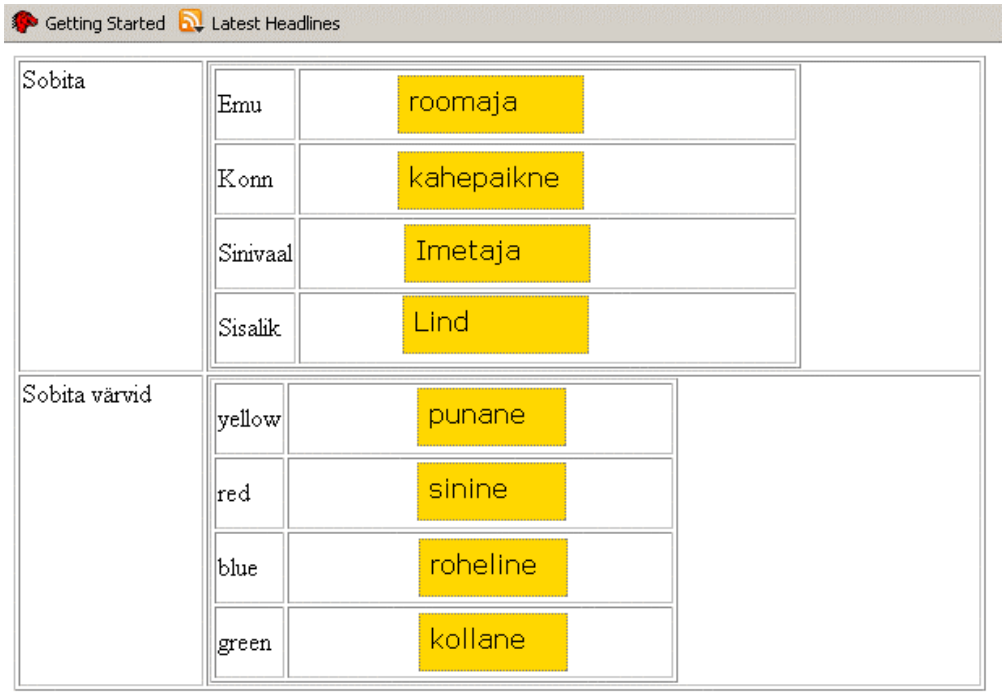
IVA vastavusse seadmise ülesannete õpilase poolne liides on lahendatud `select` elementide abil. Vasakpoolse tulba mõistetele kõrval asuvast `select` elemendist tuleb valida sobiv vaste. Keskkonna ilmestamiseks eraldasin süsteemist kahe küsimuse fragmendid ja struktuuri paremaks näitlikustamiseks lisasin paigutamisel kasutatud tabelile nähtavad raamid (Pilt 13):

Pilt 13: IVA vastavusküsimuste muudetud fragment



Seades eesmärgiks asendada `select` elemendid lohistatavate elementidega lähtume integreerimise mugavuse aspektist. Eesmärk on piirduda vaid välise skriptiga, mis on piisavalt intelligentne, et otsida üles vajalikud elemendid ja asendada need lohistatavate elementidega, kirjutades neile sisu, mis on loetud `select` elemendi valikutest. Vastavusülesanded peale skripti lisamist on kujutatud järgmisel pildil (Pilt 14):

Pilt 14



Esmalt peab kindlaks tegema, milliste tunnuste järgi skript vajalikud *select* elemendid leiab. Kuigi hetkel on tegu ainsate *select* elementidega ja võiks kõik järjest asendada, siis ei saa välistada, et need ainsaks jäävad ka tulevikus. Näiteks kui mingi lehekülje otsingu või navigatsiooni osa hakkab kasutama tulevikus *select* elementi, siis kaoks see testileheküljel.

IVA testimissüsteemi küsimused on identifitseeritavad vastavate vormielementide *name* atribuudi väärtuse järgi, mis koosneb tähest “y” ja järjekorranumbrist (Koodinäide 42):

Koodinäide 42: IVA vastavusküsimuste Select element

```
<select name="yl9">
<option value="0" selected>punane</option>
<option value="1">sinine</option>
<option value="2">roheline</option>
<option value="3">kollane</option>
</select>
```

Ühe küsimusegrupi vormiväljadel on kõigil samasugune nimi. Seega on esimesed sammud kõikide *select* elementide kollektsiooni liikmete läbikäimine ja kusjuures arvesse lähevad vaid need elemendid, mille *name* atribuut algab “y” tähega. Loomulikult pole välistatud, et samal lehel on veel teisigi “y” tähega algavaid *select* elemente, kuid antud juhul ei ole tegu tavapärase kasutusmugavuse ja vigade vältimise disainiga, vaid integreerimise mugavuse tagamisega. Juhul kui käesolev lahendus ei sobi, siis ei tohiks olla raske lisada elementidele kindlamaid eraldusmärke.

Select elemente, mille nimi algab “y” tähega läbiva tsükli kirjeldame järgmiselt (Koodinäide 43):

Koodinäide 43: Kõiki Select elemente läbiv korduslause

```
sE=document.getElementsByTagName('select'); // select elemendid
for(i=0;i<sE.length;i++){  //  tsükkel,  mis  läbib  kõik  select-id
if(sE[i].name.charAt(0)=="y"){ // tingimuslause, mis arvestab "y"-ga
    // iga select välja puhul tehtavad tegevused
}
}
```

Järgmiseks sammuks on iga elemendi juures sooritada asendus. See tähendab, et samasse kohta tuleb panna uus lohistatav element ning vana element tuleb kaotada. Selleks tuvastame kõigepealt elemendi, mis on selekteerimisväljale *parent* element. Siin on erinevused Internet Exploreri ja teiste brauserite vahel: kui esimene toetab antud juhul *parentElement* objekti, siis teised *parentNode* objekti. Defineerime selle objekti muutjana *vanem* lühendatud tingimuslausega (Koodinäide 44):

Koodinäide 44

```
vanem =(sE[i].parentElement)?sE[i].parentElement:sE[i].parentNode;
```

Nüüd võime juba teha esimese katse ja iga vanemelemendi sisse midagi kirjutada. Selleks võib kasutada praeguseks juba *de facto* standardiks saanud *innerHTML* omadust, mis on ühtlasi ka kõige kiirem meetod operatsioonideks dokumendi struktuuris. Proovime sinna kohe kirjutada vastava *select* elemendi valiku väärtuse sellises järjekorras, et küsimuse esimene vaste saab olema esimene valiku (*option* elemendi) väätus ja teine vaste teise valiku väärtus jne. Kui ma räägin siin valiku väärtusest, siis pole see terminoloogiliselt küll päris õige, sest see, mida kasutaja *select* elemendi juures näeb, ei ole mitte väärtus (*value*) vaid tekst, serverisse seda nähtavat osa ei postitata. Antud näites postitatakse serverisse hoopis numbrilised väärtused. Kuna lohistatavate elementide jaoks tuleb kätte saada see nähtav osa, siis selleks on JavaScripti abil võimalik pöörduda üle DOM puu vastava *textNode* väärtuse poole või kasutada juba kolmanda põlvkonna brauserite poolt toetatud *options* kollektsiooni elemendi *text property* küsimist. Kasutan käesolevas näites viimast meetodit. Selle süntaks oleks *selectelement.options[nr].text*, kus *selectelement* tähistab asendatavat *select* elementi ning *nr* asemel on vastava valiku järjekorranumber arvestades loendamist nullist. Kui tegu oleks vaid ühe vastavusülesannete grupiga, oleks lahendus lihtne – kuna ühe *select* elemendi sees on valikud sama palju, kui küsimuses *select* elemente, siis võib kasutada tsükli indeksit järgmiselt (Koodinäide 45):

Koodinäide 45

```
vanem.innerHTML+=sE[i].options[i].text
```

Viimases näites lisatakse esimese vastavusküsimuse juures *select* elemendile juurde õiged tekstid, kuid järgmise küsimustegrupi juures tuleb veateade. Lahtiseletatult on muutuja "i" väärtus meie näites teise küsimuse juurde jõudes 4, kuid valikute kollektsiooni poole peaks pöörduma nüüd taas nullist. Lahenduseks on muutuja "i" väärtuse ja käesoleva *options* kollektsiooni jagamise jäägi kasutamine *options* elementide indeksina:

Koodinäide 46

```
J = i % sE[i].options.length;
sisu = sE[i].options[J].text;
vanem.innerHTML += sisu;
```

Edasi on küsimus juba tehnilist laadi. Lohistamistehnikat käsitlevast peatükist võib võtta näitekoodi ja kombineerida sinna sisse vajalikud muutujad. Samuti tuleb igas korduses muuta nähtamatuks *select* elemendid. Oluline on hoolitseda ka skripti käivitumise eest. Nimelt esialgses näites hakkab skript kohe otsima *select* elementide kollektsiooni ja kui lehekülg ei ole veel täielikult laetud, siis ei pruugi *select* elemente veel leiduda. Lahenduseks on skripti lisamine lehekülje lõppu või stsenaariumi käivitamine *load* sündmuse peale. Oma näites kasutan viimast võimalust. Järgnevalt panengi kõik kokku:

Koodinäide 47

```
var liigub=0; // globaalne muutuja lohistamise registreerimiseks
window.onload = function(){
sE = document.getElementsByTagName('select');
 for(i=0;i<sE.length;i++){
   if(sE[i].name.charAt(0)=="y"){
    sEl = sE[i].offsetWidth;  // select elemendi laius
```

```
    sEk = sE[i].offsetHeight; // select elemnendi kõrgus
vanem=(sE[i].parentElement)?sE[i].parentElement:sE[i].parentNode;
       vanem.style.width = (sEl*3)+'px';
       vanem.style.height = (sEk*2)+'px';
       sE[i].style.visibility = 'hidden';
       sisu=sE[i].options[i % sE[i].options.length].text;
       lohel= '<span id="e_'+i+'" style="position:absolute; '
       +'padding:6px; border:dotted gray 1px; cursor:move; '
       +'background-color:gold; font-family:verdana; '
       +'width:'+sEl+'px;height:'+sEk+'px" '
       +'onmousedown="this.style.zIndex=\'100\'; '
     +'this.style.position=\'absolute\';liigub=1;x=this.offsetLeft;'
 +'y=this.offsetTop" onmouseup="liigub=0;this.style.zIndex=\'1\';" '
      +'onmousemove="if(liigub==1){ this.style.left=(event.clientX-'
     +'(this.offsetWidth/2))+\'px\'; this.style.top=(event.clientY-'
     +'(this.offsetHeight/2))+\'px\';}">'+sisu+'</span> ';
    vanem.innerHTML=lohel;
  }
 }
}
```

Võrreldes lohistamistehnika üldise tutvustamisega olen siia lisanud paar uut stiilimäärangut, milleks on punktiirjoonega raam (*`border:dotted gray 1px`*), lohistamise võimalusele viitav hiirekursor (*`cursor:move`*), samuti muudan dünaamiliselt hiireklahvi allavajutusega lohistatava objekti teistest “kõrgemale” tõstmist tagava *`zIndex`* väärust. Kui nüüd ülaltoodud skript lisada IVA testide lehekülje algusesse, siis muudetakse *`select`* elemendid automaatselt lohistavateks objektideks

Edasi on vaja tagada andmete serverisse saatmine. Selleks kasutan neidsamu, nüüd juba peidetud *`select`* elemente, muutes nende väärtused samaks, mis on neile vertikaalteljel lähimal lohistataval elemendil. Selle kontrollimiseks loon eraldi protseduuri, mis käivitub iga elemendi paigalelohistamisel.

Koodinäide 48

```
function muuda_ligema_peidetud_select_value(lohistatav_element){
 s = document.getElementsByTagName('select');
 y = lohistatav_element.offsetTop;
  for(i=0;i<s.length;i++){
vanem=(s[i].parentElement)?s[i].parentElement:s[i].parentNode;
   if(s[i].name == lohistatav_element.id.split('_')[0] && Math.abs(y-
leiaY(s[i])) < lohistatav_element.offsetHeight/2){
     s[i].selectedIndex = lohistatav_element.id.split('_')[1];
   }
  }
}
```

Tuvastamaks, et element on ikka lohistatud õiges blokis asuva välja juurde, on valitud samas blokis asuvate lohistatavate elementide ID väärtuste prefiksiks vastava bloki `select` elemendi nimi. Teine ID pool on number, mis vastab lohistatava elemendi sisu järjekorranumbrile `select` välja valikutes (`selectedIndex`).

Probleemseks kohaks on lohistatava elemendi ja `select` elemendi vahekauguste hindamine, sest `select` elemendid ei ole positsioneeritud absoluutsete positsioonidega ja nende puhul tagastab `offsetTop` omadus kauguse vanemelemendist. Seetõttu lõin abistava rekursiivse funktsiooni `leiaY()`, mis liidab `select` elemendi kaugusele vanemelemendi (meie näites tabeli lahtri) ülaservast selle elemendi kauguse tema vanemelemendi ülaservast ja nii edasi seni kuni vanemelemente enam ei ole:

Koodinäide 49

```
/* abifunktsioon tuvastamaks select (või iga muu) elemendi
absoluutset positsiooni *

function leiaY(obj){
var algneObj=obj; /* viitame rekursiooni alustavale elemendile */
/* defineerime select elemndi top koordinaadi nullina */

var sTop = 0;
if (obj.offsetParent){ /* kui objektil on vanemelement */
```

```
/* kordame kuni objektil on vanemelement */
 while (obj.offsetParent){
/* selecti top koordinaadile liidame juurde objekti top koordinaadi
*/
   sTop += obj.offsetTop;
  obj = obj.offsetParent;
/* seame muutuja obj viitama vanemelemendile */
 }
}
else if (obj.y)
sTop += obj.y;
if(!window.opera   &&   document.all   &&   document.compatMode   &&
document.compatMode !=
 "BackCompat" && algneObj.style.position!='absolute') {
sTop +=parseInt(document.body.currentStyle.marginTop);
}
return sTop;
}
```

`Select` elemendi sisu vahetava funktsiooni käivitamise lisama `select` elemente asendavate `span` elementide onmouseup sündmusehaldajasse:

Koodinäide 50

```
onmouseup="liigub=0;this.style.zIndex=\'1\';
muuda_ligema_peidetud_select_value(this);
```

Järgmine probleem, mis kohe silma ei hakka on see, et elementide positsioneerimisel arvestatakse kaugusi dokumendi ülaservast ja vasakust servast, aga hiire positsiooni tagastavad `event.clientY` ja `event.clientX`, arvetsavad seda vastavalt erinevatele sündmusemudeli implementasioonidele ka nähtava osa suhtes. See tähendab, et probleeme pole seni, kuni lehte pole keritud, kuid need tekivad kohe, kui pikemat lehekülge natuke alla kerida.

Seetõttu kirjutasin veel kaks lühikest lisaprotseduuri, mille tagastatavad väärtused tuleb juurde liita lohistatavate elementide x ja y koordinaatidele kui neid muudetakse onmousemove sündmusehaldaja stsenaariumis. Protseduurid ise on järgmised:

Koodinäide 51

```
/* tagastame lehekülje ülekeritud osa pikkused */
function skrollX(){
return document.documentElement.scrollLeft||document.body.scrollLeft||0;
}
function skrollY(){
return document.documentElement.scrollTop||document.body.scrollTop||0;
}
```

Hetkel on antud versioon testitud ja töötab Internet Exploreri versioonidega 5, 5.5 , 6 ja 7b, Mozilla versioonidega 1.3-1.8 ja vastavate Firefox ning Netscape versioonidega ning Opera versioonidega 7.6, 8 ning 9.

Mõistagi ei ole antud prototüüp lõplik. Piiritleda võiks piirkonna, mille sees võib elementi lohistada, antud näites võib seda teha kogu lehekülje piires, mis võib tekitada olukorra, kus ühe küsimuse vastavuselemendid lähevad kogemata teise küsimuse juurde, ehkki see tulemuses ei kajastu. Eri küsimustegrupi elemendid võib veel tähistada eri värviga. Samuti ei pruugi olla piisavaks kontrollfunktsiooni käivitamine mingi elemendi kohalelohistamise järel. Kui kasutaja arvab, et mingi element ongi kohal ja seda üldse ei puuduta, siis `select` elemendi väärtust muutev funktsioon ei käivitugi. Olukorda parandaks see, kui kogu lohistamispiirkond oleks laiem ja vastavusse seatavad elemendid asuksid teineteisest kaugemal, mis annaks kasutajale vihje, et elemente tuleb liigutada vasakule vastavusse seatava elemendiga kohakuti.

8 WYSIWYG tehnoloogia mugavam integreerimine.

Käesolevas peatükis tulevad vaatluse alla võimalused, kuidas muuta loodud komponentide integreerimine erinevatesse süsteemidesse võimalikult lihtsaks. Käsitlen nelja tehniliselt erinevat lahendust ja nende plusse ning miinuseid.

Üks tähtis aspekt, mida tuleb järgida, käsitletavus. Juhul kui keegi tahab olemasolevasse süsteemi lisada WYSIWYG komponente, siis peaks tavapäraste HTML vormi sisestuselementide asendamine toimuma võimalikult lihtsalt. Ei saa lugeda soovitavaks eriliigiliste skriptide läbipõimumist. Ehkki internetibrauserid suudavad renderdada väga mitmeid erinevaid keeli, on mõistlik hoida need teineteisest lahus. Nagu hoitakse lahus lehekülje sisu ja vorm. Nagu hoitakse lahus HTML ja CSS, nii oleks hea lahus hoida ka JavaScript muust sisust.

Seetõttu on soovitatav valmistada kliendipoolsed skriptid lihtsa käsitletavuse printsiibist lähtudes, mis tähendaks seda, et ideaaljuhul peaks piisama vaid ühest lisareast dokumendis, millega lingitakse külge väline .js fail. See tähendab, et mingid JavaScripti sündmusi haldavad atribuudid (*event handlers*) HTML elementides pole soovitavad. Võimalusi selleks on mitu:

- Võib luua skripti, mis orienteerub dokumendi objektimudelis ja loob dünaamiliselt uued elemendid ja paigutab need vanade asemele.

- Võib käsitleda kogu dokumenti või selle osi stringina, mille sisus teha asendused ja vahetada välja näiteks kõik vormi `textarea` tüüpi elemendid või isegi kõik sisestusväljad (st peale `textarea` veel `input` tüübinimega `text`).

- Võib jätta kontrolli integreerijale ja pakkuda välja protseduurid, mille ta peab käivitama teatud kohal, kus väljastatakse vajalik element ja peidetakse ebavajalik.

Elemendi peitmine võib tähendada kolme asja:

- täielikku kaotamist,
- nähtavuse kaotamist või
- kinnikatmist.

Esimese puhul ei saa kasutada varjatud vormielemente andmete ärasaatmiseks ning peab saatmiseks kasutama näiteks `XMLHttpRequest` meetodit. Küsimus on kas selle meetodi peale võib loota. Iseenesest ei tohiks olla mingit probleemi Mozilla ja üldse Gecko mootoril baseeruvate brauserite puhul, kuna seal oli toetus `XMLHttpRequest`-ile olemas juba enne kui toetus `designMode` omadusele (*property*) `iframe` elemendi puhul. Samuti ei tohiks olla probleemiks KHTML mootoril baseeruv Safari 1.2 ja uus Konqueror. Ainsaks probleemiks võib siin osutuda Internet Explorer, mis toetas küll kõige varem `designMode` omadust, kuid erinevalt sellest sõltub `XMLHttpRequest` ActiveX-ist, milline toimub küll tavaseadete puhul ilma mingi turvadialoogita, kuid või osutuda probleemiks karmimate turvaseadete korral. Kuigi Internet Explorer 7 realiseeris XMLHttpRequest objekti loomulikuna (native), siis kulub aega, et kasutajad sellele üle lähevad. Pealegi on Internet Explorer 7 mõeldud vaid Windows XP SP2, Windows 2003 Server ja Windows Vista jaoks, varasemate versioonide jaoks vähemalt esialgsete väidete põhjal uut Brauserit ei tehta.

Kui valitav interaktsioonimudel lubab andmete postitamise ajal tegevuse katkestust siis võib kasutada vormiväljasid, mis on peidetud. Vahepealne versioon XMLHttpRequest meetodi kasutamise ning traditsioonilise postitamise vahel oleks dünaamiliselt kliendi pool varjatud iframe elemendi tekitamine, ning vajalike vormielementide kirjutamine sinna sisse. Töö salvestamisel kirjutatakse väärtused uutest elementidest varjatud raamis asuvasse vormi ning postitatakse taustal ilma kogu lehekülge taaslaadimata.

Kokkuvõtteks võib öelda, et kõik käsitletud viisid komponentide integreerimiseks on aktsepteeritavad, võimalik on isegi nende kooskasutus. Kriteeriumid valikute tegemisel on integreerimise lihtsus ja paindlikkus, ühilduvus ja komponendi renderduskiirus. Integreerija jaoks mugavaim oleks `<script>` märgendi lisamise dokumendi

päisesse, millest on `src` atribuudiga lingitud vastavale skripti failile ja ülejäänud toimetaks juba skript ise. Ka sellise lahenduse puhul on olemas kaks võimalust.

Skript orienteerub elementide järgi ja asendab kõik asendamisele kuuluvad elemendid. Asukohad arvestatakse absoluutsete positsioonidega asendatavate elementide järgi , kuid veel parem kui asendatav element pannakse samale kohale. Absoluutsed positsioonid nõuaksid teatavasti ümberarvestusi juhuks kui brauseriakna suurust muudetakse.

Skript orienteerub konstantidega etteantud muutujate järgi. Vastavad konstandid võib ära tuua skripti alguses ja osutada neile vastavatele `id` ja `name` atribuutidele, mille järgi asendusi toimetada. Samas sektsioonis võib määrata ka midagi muud: näiteks uute komponentide mõõtmeid ja värve jne.

Arvestades kasutatavaid tehnoloogiaid toetavate brauserite võimet dokumenti skripti abil manipuleerida võib lugeda parimaks lahenduseks ühe skripti lisamist, mis orienteerub elementide järgi ning asendab kõik `textarea` elemendid ja vajadusel ka ka `input` elemendid. Orientiiriks võib seada ka elementide `id`, `name` või `class` atribuudid, kui neid kasutatakse järjepidevalt.

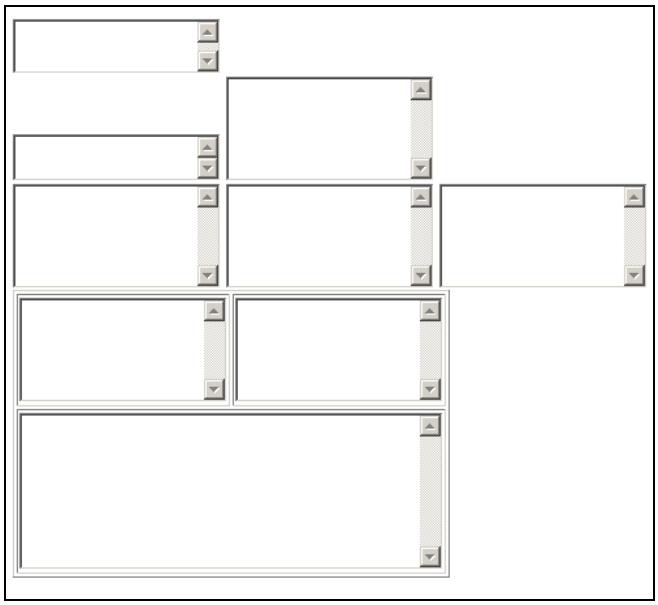
8.1 Kõikide textarea elementide asendamine iframe elemendiga

Käesolves alalõigus leiab käsitlemist katse, milles püüdsin saavutada välise skripti loomist, mis täiesti universaalselt orienteeruks tundmatu ülesehitusega dokumendis ja asendaks kõik määratud elemendid täpselt sama suurte elementidega ning paigutaks nad samadele kohtadele. Järgnevas näites on arvestatud, et asendada tuleb kõik dokumendis sisalduvad `textarea` elemendid.

Kui lohistamistehnika integreerimisel IVA vastavusülesannete juurde sai lähtuda alati `parent` elemendist, siis selline lahendus töötab vaid juhul, kui selles `parent` elemendis midagi muud peale asendatava elemendi ei ole. Kuid paigutada võib mitmel moel: tabelitega ja muude elementide abil või lihtsalt reavahetuse `<br />` märgiga

või sootuks teksti sisse. Niisiis tuleb leida selliste olukordade jaoks universaalne meetod, mis võimaldaks piirduda vaid välise .js faili linkimisega. Katsefail, mis on järgneva näite aluseks, paigutab *textarea* elemendid leheküljel väga ebasüstemaatiliselt ja eri tehnikaid kasutades. Kasutatud on paigutamist nii tabeli abil, kui ka reavahetuse märke kasutades. Eesmärgiks on skripti abil loodavatele elementidele saavutada samasugune paigutus ja mitte kasutada seejuures absoluutset positsioneerimist:

Pilt 15: Ebasüstemaatiliselt paigutatud elementidega katsefail



Alustada tuleb ikka algusest, pöördudes asendatavate elementide kollektsiooni poole ja käia see ükshaaval läbi (koodinäide 19):

Koodinäide 52

```
a=document.getElementsByTagName('textarea');
 for(i=0;i<a.length;i++){
 // siin loome uued ja asendame vanad elemendid
}
```

Korduslausete sisu algab uue *iframe* elemendi loomisega (koodnäide 20). Seekordses näites kasutan selleks *createElement* meetodit. Peale elemendi loomist omistame talle kohe asendatava elemendi (antud juhul siis *textarea*

elementide kollektsiooni liikme *a[i]*) mõõdud ja igaks juhuks ka *top* ja *left* stiilimäärangud:

Koodinäide 53

```
ir=document.createElement('iframe');
ir.style.width=a[i].offsetWidth;
ir.style.height=a[i].offsetHeight;
ir.style.left=a[i].offsetLeft;
ir.style.top=a[i].offsetTop;
```

Vajadusel võib omistada ka muid stiile, kuid *iframe* elemendi puhul ei jõustuks taustavärv, tekstivärv ja tekstitüüp, sest need tuleb määrata *iframe* elemendi sisesele dokumendile.

Vahepeal tuleb anda elemendile nimi ja id ning seejärel asuda paigutamise juurde. DOM määratleb sellise meetodi nagu *insertBefore* (Gecko DOM; MSDN insertBefore), millel on kaks argumenti: esimene, mis osutab loodavale elemendile (käesolevas näites tähistatud muutujaga *ir*), ja teine argument osutab elemendile, mille kõrvale või õigemini ette tuleb uus element paigutada. Seega sobib see meetod leheküljesisesteks elementide asendamiseks ideaalselt ja tekitab kõige vähem probleeme:

Koodinäide 54

```
pe=(a[i].parentNode)?'Node':'Element';
 a[i]['parent'+pe].insertBefore(ir,a[i]);
 a[i].style.display='none';
```

Ülaltoodud kolm rida teostavad uue elemendi paigutamise. Esimene rida defineerib *parent* elemendi, mille poole pöördumine on Mozillal ja Internet Exploreril erinev.

Erinevalt lohistamistehnika integreerimise näitest, kasutan siin ära mõlemale meetodile ühist esimest osa `parent` ning liidan sellele eelnevalt defineeritud lõpud `Node` või `Element.`

Kummaline konstruktsioon `a[i]['parent'+pe]` vajab ehk pisut lahtiseletamist, sest selline tehnika pole tavapärane. Seda oleks võinud vältida, defineerides esimesel real terve meetodi nime, kuid selle tehnika kasutamisel ongi siin demonstratiivne eesmärk, sest kindlasti on olukordi, kus seda kasutades on võimalik koodile kuluvat ruumi olulisel määral kokku hoida. Element `a[i]` viitab asendatavale `textarea` elemendile. Saamaks teada, mis elemendi sees ta asub, peaks rakendama Internet Exploreri puhul järgmist konstruktsiooni `a[i].parentElement` ning Mozilla ja teiste DOMi toetavate brauseritega `a[i].parentNode`, kuid JavaScript võimaldab objektile meetodite ja omaduste poole pöörduda nii punktnotatsiooniga kui sulgnotatsiooniga, mille puhul kasutatakse kantsulge ja nende sees stringi tüüpi meetodi, omaduse või objekti nime. Seda nimetatakse ka *hash* tüüpi pöördumiseks, kuna see sarnaneb PERLi ja teiste keelte *hash* tüüpi massiivi elemendi poole pöördumisega.
Välja näeks see siis `a[i]['parentElement']` ja `a[i]['parentNode']` ning antud näites on kantsulgudes olev string moodustatud stringi `parent` ja muutuja `pe` ühendamisest kokku `a[i]['parent'+pe]` .

Saadud konstruktsioon on tegelikult objekt, mis on saadud teisele objektile rakendatud meetodi tulemusel. Antud näites on tegu objektiga, milles asub `textarea` element. Selleks võib olla nii `body` kui tabeli lahter `td.` Rakendades saadud objektile meetodit `insertBefore`, seades teiseks argumendiks `textarea` elemendi `a[i]`, ongi eesmärk saavutatud ning `iframe` asub alati vahetult asendatava elemendi ees.

Kui lisada koodi veel *textarea* elementide peitmine, raamide sisu kirjutamine ja `designMode` omaduste muutmine, siis on tulemuseks täpselt samasuguse paigutusega lehekülg, kuid selle vahega, et `textarea` elementide asemel on WYSIWYG redigeerimist võimaldavad elemendid:

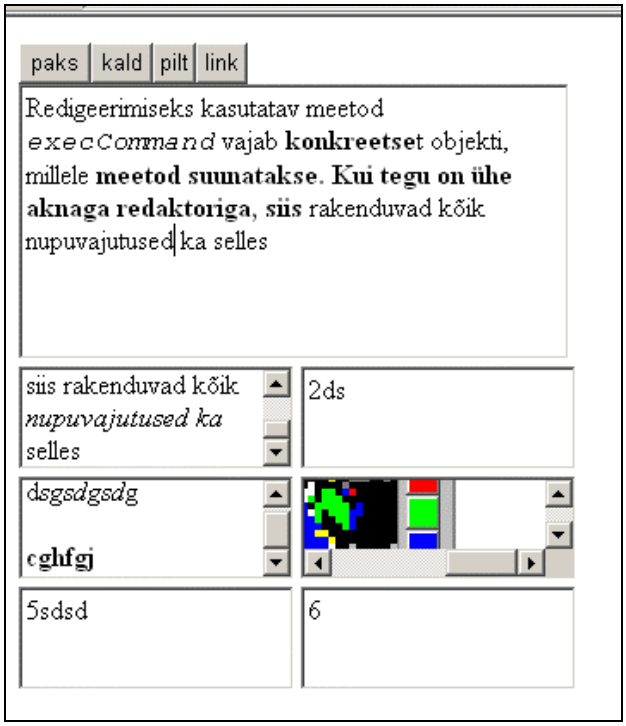
Koodinäide 55

```
window.onload=function(){
 a=document.getElementsByTagName('textarea'); // läime läbi kõik
 for(i=0;i<a.length;i++){                // textarea elemendid
   ir=document.createElement('iframe');  // loome iframe elemendi
   ir.style.width=a[i].offsetWidth;      // ja omistame talle
   ir.style.height=a[i].offsetHeight;    // textarea mõõdud
   ir.style.left=a[i].offsetLeft;        // ja positsiooni
   ir.style.top=a[i].offsetTop;
   ir.setAttribute('id','r'+i); // seame iframe elmendile id
   ir.setAttribute('name','r'+i); // ja name atribuudi
   ir.name='r'+i;           // kordame seda teatud brauserite jaoks
   pe=(a[i].parentNode)?'Node':'Element'; // textarea vanemelement
   a[i]['parent'+pe].insertBefore(ir,a[i]);//asetame iframe textarea
   a[i].style.display='none'; // ette ning peidame viimase
   nimi='r'+i

    irr=parent.frames[nimi].document; // defineerime oframe elemendi
                                      // dokumendi
       irr.open();                 //ja kirjutame sinna raami numbri
       irr.write('<html><body style="background-color:white; ' );
       irr.write('margin:2px">'+i+'</body></html>');
       irr.close();

       if(irr.designMode){
         irr.designMode='on';
       }
    }
}
```

Seni kuni näide piirdub üksnes elementide asendamisega on kõik korras, probleemid tekivad alates hetkest, kui on vaja akna sisu redigeerida kõikide akende jaoks ühise nupurea abil. Redigeerimiseks kasutatav meetod *execCommand* vajab konkreetset objekti, millele meetod suunatakse. Kui tegu on ühe aknaga redaktoriga, siis

rakenduvad kõik nupuvajutused ka selles ühes aknas. Käsk antakse siis näiteks järgmiselt: `redaktor.execCommand('bold',false,null);`. Mitme akna puhul tuleb arvet pidada selle üle, milline aken on parasjagu aktiivne, ehk milline `iframe` element on refereeritav kui `redaktor`.

Pilt 16: Mitme aknaga WYSIWIG redaktori prototüüp



Uurimuse käigus selgus, et aktiivse akna kindlakstegemine ei ole sugugi triviaalne ja mingeid dokumenteeritud meetodeid sellise mitmeaknalise redaktori rakendamiseks ei leidunud. Et olla päris täpne, siis Internet Exploreriga erilisi probleeme ei olnud. Probleem oli seoses Mozilla ja muude brauseritega, millest lähemalt järgmises alpeatükis.

8.1.1 Ifreimi raaminime tuvastus

Käesolevas alapeatükis leiab lähemat käsitlemist pealtnäha tühine, kuid tehniliselt lahenduselt üsna keerukaks osutunud probleem - kuidas tuvastada, milline redigeerimisaken on parasjagu aktiivne. On üsna tõenäoline, et leitud lahendus on unikaalne ja esmakordselt dokumenteeritud just selles töös.

Põhiline erinevus Internet Exploreri ja Mozilla vahel on selles, et Mozilla ei tunnista sündmust *focus*, *iframe* elemendil. See on mõistetav, kuna *iframe* elemendis on teine dokument ja see registreerib ise oma sündmused. Seega soovides, et raam reageeriks mingitele sündmustele peame Mozilla puhul omistama sündmuse raamis olevale dokumendile.

Kui sündmuste kuulaja lisamine Mozillal raamis olevale dokumendile toimus probleemideta, siis hoopis keerukamaks probleemiks osutus nendest registreeritud sündmustest kasu saamine. Internet Exploreriga polnud mingit probleemi, sest sündmushaldur oli lisatud kohe *iframe* elemendile, mistõttu sealt oli juba lihtne saada kätte meid huvitavad andmed – objekti *ID* või *Name* atribuudi väärtus. Mozilla puhul, kus sündmusele reageeris *iframe* sees asuv dokument, oli olukord hoopis keerulisem. Nii üllatav kui see ka pole, ei leidu ühtegi standardset ega spetsiifilist meetodit, kuidas *iframe* elemendi sees olev dokument võiks teada, mis on selle *iframe* *id*, või indeks *frames* kollektsioonis.

Otsingute tulemuselt leidsin mitmeid sarnase probleemiga silmitsi seisvaid hädalisi, kes foorumitest ja useneti gruppidest selles küsimuses rahuldavat abi ei saanud. Ainus reaalselt töötav lahendus, mida aegajalt pakuti oli see, et raamis asuv dokument pöördub *parent* dokumendi poole, ja käivitab korduse, mis läbib kõik *iframe* elemendid kontrollides nende *src* atribuudi väärtust ja võrreldes seda enda *location.href* väärtusega. Kui need väärtused kattuvad, siis võib järeldada, et tegu on sama raamiga ja korduse järjekorranumber vastab selle raami indeksile *iframe* elementide kollektsioonis. Kuid antud lahendus kehtib vaid eeldusel, et kõikide raamide sisuks on unikaalse aadressiga dokumendid. Meie redaktori puhul peaks olema uute küsimuste loomisel *src* atribuudi väärtus kõikidel sama *about:blank* või on sootuks tühi ja raamide sisu on JavaScriptiga genereeritud, mistõttu pakutud lahendus ei töötaks.

Selles lootusetus olukorras püüdsin probleemi lahendada sel moel, et genereerisin *iframe* elemendi sisus *body* elemendi sisse *id* atribuudi, mille väärtus vastas selle *iframe* elemendi *id* väärtusele ning omistasin sellele *body* elemendile sündmusehalduri, mis raami nime registreeriks. Järgmine probleem oli muidugi see, et

teatud klikid registreeriti väljaspool `body` elementi ja refereerisid seega vastava HTML elemendi `id` väärtusele. Kui redigeeritavas dokumendis elemente lisandub, siis reageerivad need kõik sündmustele ja tekitavad palju segadust. Pärast väikeste JavaScripti stsenaariumide juurdekirjutamist hakkas asi sel moel siiski tööle ja seejärel, lugenud probleemi lahendatuks, ma selle temaatikaga edasi ei tegelenud.

Hiljem selle juurde tagasi pöördudes avastasin puhtjuhuslikult konstruktsiooni, mis vajaliku tulemuse hoopis elegantsemalt väljastab (Koodinäide 56):

Koodinäide 56

```
e.target.ownerDocument.defaultView.name
```

Esimene katse ei andnud mingeid tulemusi, kui püüdsin nii viidata `id` atribuudile. Kuid `name` väljastas soovitud tulemuse. Nagu arvata võis ei andnud ükski otsing ülaltoodud koodilõigule mingeid vasteid ka siis, kui olin kõrvaldanud algusest oma defineeritud `e.` Peale lõpust `.name` kõrvaldamist andis Google otsing siiski 4 asjasse mittepuutuvat vastet.

Kõnealuse testfaili kood nägi välja järgmine:

Koodinäide 57

```
<script>
window.onload=function(){
for(i=0;i<4;i++){ // genereerime neli iframe elementi
   if(document.createElement){
    aken=document.createElement('iframe');
    aken.setAttribute('width','200');
    aken.setAttribute('height','200');
    aken.setAttribute('id','ir'+i);
    aken.setAttribute('name','ir'+i);
    document.body.appendChild(aken);
   }
r=parent.frames['ir'+i];
sisu='raami id ja name atribuudi väärtus on ir'+i;
r.document.open();
```

```
r.document.write('<html><body>'+sisu+'</body></html>');
r.document.close();
r.document.designMode='on';
/* sündmuse kuulaja omistamine Mozillale */
  if(document.addEventListener){
   r.document.addEventListener('click',function(e){
     alert(e.target.ownerDocument.defaultView.name);
     },
  false);
/* sünmuse kuulaja omistamine Internet Explorerile */
 }else if(document.attachEvent){
   aken.attachEvent('onfocus',function(){
   alert(event.srcElement.id);
 }
);
}
}
}
</script>
<body>
<h1>Testfail aktiivse iframe nime või id tuvastamiseks</h1>
<em>Klikkige raamil kontrollimaks, kas raam teab oma nime</em><hr>
</body>
```

Testides erinevate Geckol baseeruvate brauseriversioonidega, selgus, et lahendus töötab laitmatult. Ka uus tulija Opera tunnistas seda tehnikat samuti.

8.1.2 Probleeme seoses Internet Explorer 5 brauseriga

Internet Explorer 5 hakkab oma kasutajaskonda küll kaotama, kuid mõned statistikad (W3Schools 2005) näitavad siiski veel 5% suurust turuosa, mis on suurem, kui Opera turuosa. Seetõttu ei tasu teda veel päris maha kanda. Põhiline probleem, selle brauseri puhul on suutmatus luua dünaamiliselt `createElement` meetodiga `iframe` elemente. See ei vasta küll päris tõele, ehkki Microsoft väidab, et meetod `createElement()`, mis implementeeriti alates versioonist 5, ei tööta `iframe` elemendi puhul (MSDN createElement). Tõepoolest järgnev näide sellisel kujul Internet Explorer 5 puhul ei tööta ja soovitud `iframe` element ei ilmu nähtavale.

Koodinäide 58

```
aken = document.createElement('iframe');
aken.style.width= '300px';
aken.style.height= '200px';
document.body.appendChild(aken);
```

Kuid Microsofti väide, et *createElement* meetodi abil *iframe* elementi luua ei saa, ei pea siiski paika.

Kommenteerides välja või kustutades ülaltoodud näitest rea:

Koodinäide 59

```
document.body.appendChild(aken);
```

ning asendades selle järgmisega

Koodinäide 60

```
document.body.innerHTML+=aken.outerHTML;
```

tekib element ka IE5 brauseril. Viga pole seega IE5 suutmatuses luua *iframe* elementi *createElement* meetodil, vaid suutmatus loodud elementi lisada *appendChild* meetodil. Tegelikult pole ka see päris õige, kuna peale dokumendi HTML struktuuri kirjutava protseduuri kirjutamist ja käivitamist nägin, et ka *appendChild* või muul DOM meetodil lisatud *iframe* element oli koodina ja koos kõigi defineeritud atribuutidega olemas. Tekkis mõte, et kui see kood kuskile ümber paigutatada, siis peaks *iframe* ometi nähtavale ilmuma. Ja oletus vastas tõele.

Kirjutades näitefaili kogu *body* elemendi sisu sinna samasse body elementi ilma midagi muutmata tagasi ilmus iframe element nähtavale. Selleks tuli lisada *appendChild* meetodi järele vaid üks rida:

Koodinäide 61

```
document.body.appendChild(aken);
document.body.innerHTML+='';
```

Muidugi ei dokumendi lõppu *innerHTML* meetodil lisamist lugeda soovitavaks, sest tegelikult toimuv protsess on selline, et loetakse dokumendi sisuks olev HTML kirjutatakse üleni üle. Selle tulemusel kaotavad oma funktsionaalsuse skriptid, mis on *body* elemendi sees ning kõik sündmusehaldurid, mis on lisatud *attachEvent* meetodiga või *on(event)* meetodil. Samal põhjusel ei saa *innerHTML* abil lisada dünaamiliselt uusi *script* elemente. Kuna *innerHTML* on praegu ka teiste brauserite dokumendi objektimudeli laiendus, siis puudutab see ka sündmusehaldureid, mis standardse *addEventListener* meetodiga lisatud. Ainsad sündmuse haldurid, mida *innerHTML* ei kahjusta on elementide atribuutidena lisatud sündmusehaldurid.

Seetõttu, et mitte kahjustada üldist dokumenti, tuleb Internet Explorer 5 versiooni iga *iframe* elemendi jaoks luua eelnevalt *createElement* meetodiga mingi konteinerelement, üks *div* või muu ja lisada *ifreimi* sisu sinna sisse. See päästab ülejäänud dokumendi sündmustehaldurid, kuid nõuab täiendavat stsenaariumit veel loodud *iframe* elemendile *onfocus* sündmusehalduri lisamiseks peale seda, kui ta on juba tekitatud.

Igale versioonile sobiva lahenduse jaoks tuleb rakendada eraldi tingimuslikku stsenaariumit, mis vastavalt versioonile esimesel või teisel moel elemendi tekitab. Kui tuleb arvestada sellega, et vanale tehnoloogiale kohandamine ei osutuks liialt keerukaks. Tõenäoliselt just nende probleemide tõttu, mida Internet Exploreriga ette tuleb seisab mitmete WYSIWYG redigeerimist võimaldavate veebipõhiste rakenduste juures miinimumtingimuste seas Internet Explorer alates versioonist 5.5.

8.2 Funktsiooninuppude genereerimine.

Nagu redaktoriakende genereerimise juures olid skripti jaoks ainsaks orientiiriks vormi textarea tüüpi *elemendid*, mille põhjal otsustada, kuhu loodud uus element paigutada, nii ka funktsiooninuppude loomise juures tuleb saavutada sõltumatus konkreetse rakenduse dokumendistruktuurist. Kuna nupud ilmuvad meie süsteemis esimese redaktoriakna kohale, siis sobib selleks näiteks *insertBefore* meetod. Järgides universaalsuse nõuet, ei tohiks limiteerida kasutatavate funktsioonide arvu, vaid anda selles osas süsteemi hooldajatele ja arendajatele vabad käed määramaks, millist komplekti redaktori funktsiooninuppe soovitakse kaasata.

Üks küsimus, mis vajab lahendamist on see, milliseid elemente kasutada nuppude genereerimiseks. Kuna niikuinii on nupud JavaScripti poolt genereeritud ja nende olemasolu ja funktsionaalsus sõltub JavaScriptist, siis võik nuppude rolli täita põhimõtteliselt iga element. Siiski eelistan ma kasutada nuppude jaoks elemente, mis on selle jaoks loodud nagu *button*, *input* tüübinimega *button* või lingi element *a*. Selle eelistuse põhjuseks on nende loomulik omadus alluda navigeerimisele tabulaatori abil, loomulik omadus omandada fookus ning loomulik omadus käivitada *click* sündmus fokuseeritud nupul sõltumata sisendseadmest. See tähendab, et kui vajutatakse klaviatuuril enter klahvile ja mõnede brauserite puhul ka tühikuklahvile, siis aktiivse nupu onclick sündmusehalduris defineeritud funktsioon käivitub. Kõike seda saab loomulikult emuleerida JavaScripti abil ka teiste elementide jaoks, kuid see eeldaks täiendavaid stsenaariumeid ja tarbetut lisakoodi.

Kui vaadata viimastel aastatel ilmunud populaarseid WYSIWYG veebiredaktoreid, siis enamuse neist nupud tabulaatoriga navigeerimisele ei allu ja on kasutatavad vaid hiire abil. Näiteks võib tuua populaarsete emailiklientide nagu Gmail, Hotmail ja Yahoo emailiklientide redaktorid või Elioni veebimaili, kuid ka otseselt redaktorina mõeldud teenused nagu Writely, HTMLArea ja teised.

Aastal 2004 loodud IVA testide redaktori nuppude tegemisel kasutasin *button* elemente ja seetõttu need alluvad tabulaatoriga navigeerimisele. Käesolevas versioonis on peamiseks muudatuseks, nuppude lisamise paindlikum tehnoloogia. Samuti mõned

navigeerimisega seotud mugavusfunktsioonid nagu nooleklahvidega liikumine nuppude vahel.

Praeguses IVA versioonis toimub redaktori lisamine leheküljele nii, et küsimuse sisestamise *taxtarea* elemendi järele on koodis kirjutatud järgmised read:

Koodinäide 62

```
<script src="js/ivaeditor"> </script>
<script>
 if(document.getElementById && navigator.userAgent.indexOf('pera')==-1
            && navigator.userAgent.indexOf('queror')==-1){
 document.write(editoriaken());
}
 </script>
```

Nagu koodist näha kaasatakse väline js fail *ivaeditor* ja järgnevalt kirjutatakse *document.write* meetodil lehekülje struktuuri string, mis on kaasatud failis defineeritud funktsiooniga *editoriaken().* Tingimuslausega on välistatud Opera ja Konqueror brauserid, mis oli aktuaalne siis, kuid mis ei pea enam paika praegu, kuna nende uuemad versioonid suudavad samu redigeerimisfunktsioone täita.

Kuid peamised põhjused, miks see lahendus pole rahuldav on selles, et *document.write* pole meetod, mida võiks tänapäeval soovitada lehekülje üksikute osade genereerimiseks. Kui mõelda mitmeaknalise redaktori peale, siis tuleks ju dokumendi struktuuris kirjutada mitmesse kohta sarnaseid koodilõike ja see poleks sugugi kooskõlas struktuuri, esituse ja käitumise lahususe põhimõtetega.

Seega tuleb luua intelligentsemad meetodid, mille järgi vajalik uus sisu lehele õigetesse kohtadesse lisatakse. Üks soovitatav võimalus oleks asendada ka kõik valikvastuste kirjutamiseks mõeldud sisestusväljad *textarea* tüüpi elementidega ning eeldades, et neid lehel rohkem ei ole lasta skriptil kõik *textarea* elemendid *iframe* elementidega asendada. Peale skripti universaalsuse, mis seisneks selles, et ei orienteeruta konkreetsete tunnustega määratletud sisestusväljade järgi vaid vahetatakse alati kõik *textarea* elemendid, lahendaks see ka elementide mõõtmete probleemid.

Kui anda loodavatele redaktoriakendele asendatavate elementide mõõtmed, siis sisestusvälja `input` kõrgus jääb liiga madalaks WYSIWYG akna jaoks.

Kuna lahendused eriti ei erine, siis käsitlen lahendust, kus tuleb elementide nimesid arvesse võtta. On teada, et küsimuse lahtri `name` ja `id` atribuudi väärtus on `kirjeldavTekst`, vastuste lahtrite `name` atribuudi väärtus `vastus` ja tagasiside lahtrite `name` atribuudi väärtus on `selgitus`.
Seega näeks protseduur välja järgmine: Skript otsib dokumendistruktuuris elemendi nimega `kirjedavText` ja asendab selle funktsiooninuppude ning küsimuse aknaga, seejärel otsib skript välja elemendid nimega vastus ja selgitus ning asendab need vastavate redigeeritavate akendega.

Kõigepealt käsitlen eraldi nuppude loomise protseduuri. Nagu mainisin eelnevalt, võiks jätta süsteemi haldajatele ja arendajatele vabaduse funktsiooninuppude lisamiseks või kaotamiseks. Selleks loon massiivi, mis koosneb `execCommand` meetodi argumentidena kasutatavatest string tüüpi muutujatest. Ka nuppudel kuvatavad ikoonid olen nimetanud nende argumentide nimedega, et ühe massiivi elemente saaks kasutada kahel erineval eesmärgil. Massiiv võiks olla järgmine:

Koodinäide 63

```
var m=["bold","italic","underline", jne];
```

ja vastavalt sellele, mida antud massiiv sisaldab võib ta genereerida näiteks järgmise nupurea:

Pilt 17

On suur hulk funktsioone, mille defineerimisel `execCommand` meetodiga, piisab vaid ühest argumendist. Selliste funktsioonidega nuppude genereerimisel pole vaja nende funktsioonide lisamiseks teha muud, kui lisada vastava argumendi väärtus massiivi soovitavale kohale ja veenduda, et sellenimeline pilt on nupu piltide kataloogis olemas. Siiski on teatud funktsioonid, mis nõuavad, spetsiifilisemat käsitlemist nagu näiteks pildi lisamine või lingi lisamine, kus `execCommand` meetodile peab andma veel ühe argumendi. Samuti kuuluvad selliste funktsioonide hulka teksti stiilide määramisega seotud funktsioonid. Peale eelnimetatute on vähemalt üks hädavajalik funktsioon, mida `execCommand` meetod ei toeta ja selleks on tabeli loomine. Kui erijuhud kokku võtta on nendeks nuppude puhul tabeli, lingi ja pildi lisamise funktsioonid, teksti vormistamist puudutava funktsionaalsuse võib usaldada ka `select` elementide hoolde.

Seega sobib ülaltoodud atribuutide (ja nupu ikoonide nimede) massiivi kasutamine ja tuleb vaid kirjutada stsenaariumid nende kolme erijuhu käsitlemiseks. Erijuhtude juurde pöördun hiljem tagasi, praegu käsitlen nuppude genereerimise üldist protsessi.

Vastavalt massiivi pikkusele võib käivitada tsükli:

Koodinäide 64

```
 for(i=0;i<m.length;i++){ /* loome iga massiivi elemndi kohta nupu */
  np=document.createElement('button');
  np.style.width='30px';
  np.style.height='30px';
/* np.style.background='url('+nupukataloog+m[i]+'.gif) no-repeat 50% 50%'; */
  np.setAttribute('type','button');
         np.korraldus=m[i];
         np.title=m[i];
         np.innerHTML='<img src="'+nupukataloog+m[i]+'.gif" '
          +'alt="'+m[i]+'" valign="middle" align="center">';
/* siin tuleks omistada nupule click sündmusehaldur */
npd.appendChild(np); /* kus npd on objekt, kuhu nupud lisatakse */
}
```

Eelolevas näites kommenteerisin välja nupule ikooni omistamise taustapildiga. Taustapilte soovitatakse kasutada kõikjal, kus pilt teenib kujunduslikke eesmärke, kuid

antud juhul ei allu taustapilt nupu käitumisele, mis jätab mulje nupu allavajutamisest. Kui pilt on eraldi `img` elemendina nupu sees nagu ülatoodud näites, siis on tulemus realistlik, kuid taustapildina jääb mulje nagu nupp oleks läbipaistev ning sellele vajutades jääb seal taga olev ikoon liikumatuks. Kui alljärgnevaid pilte katsefailidest võrrelda, siis ülemine on tehtud `img` elemendiga ning alumine nupu taustapildiga. Vahe on pildil vaevumärgatav, kuid ülemises näites on allavajutatud nupul olev pilt nihutatud paari pikseli võrra paremale ja alla kui teda võrrelda kõrvaloleva identse nupuga. Ometi mõjub see väike nihe paju loomulikumalt kui paigalolek.

Pilt 18: nupud pildiga

Pilt 19: nupud taustapildiga

Liikumist saaks emuleerida taustapildi nihutamisega vastavalt `mousedown` ja `mouseup` sündmustele, aga sellel pole mõtet, kui on võimalus ära kasutada eeldefineeritud käitumist. Peale selle on pildi puhul võimalik pakkuda alternatiivteksti neile, kel piltide vaatamine on mingil põhjusel keelatud. Stiiliga taustapildi puhul peaks sellise alternatiivteksti saavutamiseks kirjutama jälle eraldi JavaScripti stsenaariumi.

8.2.1 Sündmusehaldurite lisamine

Nuppude genereerimise ajal võib lisada neile sündmusehaldurid, mis reageerivad näiteks `click` sündmusele ja käivitavad vajaliku funktsiooni. Olemasolevas IVA versioonis on sündmusehaldurid lisatud kõige algsemal viisil – elementide atribuutidena. Kuigi struktuuri ja käitumist ei tasu segada, siis dünaamiliselt genereeritud objektide puhul pole see probleemiks. Antud juhul, kui nupud on genereeritud `createElement` meetodiga, siis puudub võimalus määrata sellisele elemendile atribuudiks sündmusehaldurit. Järgmine kood ei tööta:

Koodinäide 65

```
// järgnev ei tööta
np.setAttribute('onclick', 'mingiFunktsioon()');
```

Selle asemel võib kasutada järgmist sündmusehalduri omistamist:

Koodinäide 66

```
np.onclick=function(){
 // teeme midagi sellele nupule vajutamisel
}
```

Kolmas võimalus on kasutada DOM meetodit `addEventListener` sündmuste registreerimiseks, kuid see meetod ei ole toetatud Internet Exploreri poolt, sest Internet Explorer oli juba enne DOM meetodi väljatöötamist loonud oma sarnase meetodi `attachEvent`. Viimatimainitud meetodid on teatud olukordades asendamatud, sest ei kahjusta objektile varem sama sündmusega omistatud funktsionaalsust. Antud näites sellise kahjustamise pärast muretsema ei pea ja võib kasutada varasemat meetodit.

Koodinäide 67

```
np.onclick=function(){
    irm=parent.frames[aktiivneRaam];
    irm.focus();
    irm.document.execCommand(this.korraldus,false,null);
}
```

Viimatimainitud näite puhul tähistab muutja `aktiivneRaam` redaktori akent, milles toimub tegevus, ning `execCommand` meetodi esimese argumendina kasutatakse igale nupule just selleks otstarbeks tekitatud atribuudi `korraldus` väärtust. See väärtus vastab elemendile massiivis, mille järgi loodi nupu pilt ja määrati tema funktsioon. Dünaamiliselt lisatud sündmusehaldurite puhul on selline või mõni muu kõrvaltee vajalik, kuna järgnev kood, mis püüaks seda atribuuti otse lisada ei töötaks:

Koodinäide 68

```
irm.document.execCommand(m[i],false,null);
```

Selle asemel viitaks massiivielement `m[i]` alati ühele ja samale defineerimata elemendile, sest massiiviindeks `i` vastab massiivi elementide koguarvule, ehk on ühe võrra suurem kui viimase elemendi indeks.

Antud näide lisas kõigile nuppudel `execCommand` meetodid neile vastava argumendiga, kuid ei arvestanud sellega, et teatud funktsioonide puhul on vajalik ka teise argumendi olemasolu. Nagu varem mainisin puudutavad need erijuhud lingi, pildi ja tabeli sisestamist ning mitmeid teksti vormistamisega seotud funktsioone. Täitmaks neid eritingimusi tuleb sündmusehalduri määramisel kirjutada eelmisest lihtsustatud näitest natuke keerukam tingimuslik stsenaarium.

9 Skriptide lisamise ja käivitamise problemaatika

Käesolevas peatükis käsitleme, skriptide lisamisega seotud probleeme. Kuna testid võivad sisaldada mitmeid eri küsimuseliike, siis on vajalik samaaegselt lisada neile erinevaid käitumisloogikaid defineerivaid stsenaariume. Haldaja seisukohast oleks kõige lihtsam lisada tervikliku testi (kui see kujutab endast tehnoloogiliselt ühte tervikliku lehte) päisesse üks skript millesse on kokku koondatud kõikide küsimuseliikide loogika. Elegantseks lahenduseks seda kahtlemata pidada ei saa, sest nii laaditakse kasutaja juurde ka täiesti ülearust koodi. See on siiski üks mõeldav stsenaarium, kui skript pole väga suur ja eriti siis, kui õnnestub elimineerida dubleerivaid stsenaariume.

Teine võimalus oleks kaasata üksikutele küsimuseliikidele mõeldud skriptid vastavalt esinevatele küsimuseliikidele. Serveri pool ei ole tehniliselt keerukas arvet pidada kasutatud testiliikide üle ja vastava info põhjal lehekülje päis genereerida. Kuid sel juhul tekib järgmine probleem, mis nõuab siiski näidetes toodud skriptide modifitseerimist. Nimelt käivitati näidetes kõik skriptid objekti *window load* sündmusega. See sündmus leiab aset siis, kui kogu lehekülg on laetud. Probleem on selles, et sündmus leiab aset vaid üks kord. Kui lisada nüüd mitu skripti, milles kõikides on käivitav loogika seatud sõltuvusse *load* sündmusest, siis käivitub vaid esimene. Õigupoolest on eri brauseritel sündmuste käsitlemine erinevalt realiseeritud ja tulemused ei ole päris samasugused, kuid ühelgi juhul ei hakkaks kõik skriptid nii tööle.

Seega tuleb asendada iga skripti *load* sündmusega käivitatav anonüümne funktsioon mingi unikaalse funktsiooniga või näiteks uue loodava objekti test meetodiga. Õigupoolest on JavaScriptis ka tavaline funktsioon tegelikult meetod – brauseris kasutatuna on see objekti *window* meetod. See tähendab, et pole vahet, kas kirjutada:

Koodinäide 69

```
function test(){/*tegevus*/} või
window.test=function(){/*tegevus*/} või
window['test ']=function(){/*tegevus*/}
```

Nii võime kasutatud näiteskriptides asendada reas *window.onload=function(){* sündmusekäsitleja *onload* mingi endadefineeritud unikaalse nimega. Olgu selleks siis näiteks järjekorda seadmise testi puhul *window.jarjekorda=function(){* ja vastavusse seadmise testi puhul *window.vastavusse=function(){*. Või kui selline konstruktsioon tundub ebasümpaatne siis kasutada nende asemel lihtsalt funktsiooni defineerimist *function jarjekorda(){* ja *function vastavusse()}*.

Nüüd kui on olemas unikaalsed meetodinimed, siis tuleb need käivitada ühe väljakutsuva skriptiga, mis ise load sündmusega käivitub:

Koodinäide 70

```
window.onload=function(){
  jarjekorda();
  vastavusse();
 // muud võimalikud tegevused
}
```

Kuna eelmainitud skript tõotab tulla suhteliselt lühike, siis selle kaasamine eraldiseisva failina tundub raiskamisena kuna fail tuleks niikuinii dünaamiliselt genereerida serveris vastavalt kaasatavatele stsenaariumitele (alternatiiviks oleks kõikvõimalike kombinatsioonide valmistegemine) ning eraldi *script* sektsiooni tekitamine päisesse oleks tehniliselt samaväärne, seda enam, et viited välistele .js failidele tuleb sinna niikuinii genereerida.

Lahendus, mis on veel mainimata ja on tegelikult üks efektiivsemaid on see, et skript elemendid ei sisalda mingit onload sündmusehaldurit vaid nad käivitatakse kohe, kui nendeni on jõutud. Sündmusehaldur on vajalik selleks, et oodata ära, millal DOM struktuur on tekkinud ja lehekülg laetud, sest vastasel korral üritaks skript pöörduda elementide poole, mida veel ei ole. Mõnikord on seda olukorda püütud lahendada teatud ajalise viite tekitamisega, kuid see ei ole usaldusväärne, kui puuduvad andmed laaditava lehe suuruse ja ühenduskiiruse kohta. Kindel meetod on skripti lisamine lehekülje lõppu, siis jõuab interpretaator skriptini alles siis, kui vajalikud elemendid, mida skript hakkab manipuleerima, on juba tekkinud. Tegelikult käivitub nii skript

isegi natuke varem, kui onload sündmusehalduri kaudu, ses kui viimane peab ära ootama kogu lehekülje, piltide, stiilide ja muu laadimise, siis esimene käivitub kohe kui temani on jõutud.

Siiski on ka sellel meetodil puudusi, mis on eelkõige stiililised, sest `script` element võiks olla lehe päises. Kuigi süsteemi arendaja ja haldaja seisukohalt poleks vahet, kuhu see rida lisada. Seni kuni IVA rakendus on veel suhteliselt õhuke, võib osutuda mõistlikuks mitmete stsenaariumite ühendamine ühte scripti, sest niikuinii tuleb alati revideerida erinevaid skripte sellest seisukohast vaadata, kas neis pole kattuvaid muutujate või funktsioonide nimesid, või ei kahjusta nad üksteist muul moel.

Kõige korrektsem on lisada ühele sündmusele juurde funktsionaalsust ilma juba varem omistatut tühistamata läbi DOM meetodi `addEventListener` ja Internet Exploreri `attachEvent` meetodi. Käivitav funktsionaalsus esitatakse siis nende meetodite teise argumendina, kui esimene argument väljendab sündmust, millega on tegu.

10 Võimalikud vead ja nende vältimine

10.1 Kopeerimine teisest dokumendist

Kui kopeerida teksti teisest dokumendiredaktorist nagu näiteks MS Word, siis tekstis olevad pildid lähevad kaduma. Tegelikult puudutab see kõiki objekte, mida veebikeskkonnas saab käsitleda vaid pildina. Näiteks võib tuua MS Office lisana kasutatava matemaatiliste valemite redaktori Microsoft Equation abil loodud matemaatilised avaldised, TextArt lisa abil loodud tekst, tekstis kasutatud nooled ja kastid. Ja tegelikult polegi selle vastu eriti midagi ette võtta.

Me tegime katsekeskkonna uurimaks, mis tegelikult sellise kopeerimise puhul toimub. Kõigepealt proovisime kopeerida MS Wordi pilte sisaldavat dokumenti selle selekteerimise ja *copy-paste* meetodil testimiseks loodud WYSIWYG redaktori aknasse. Testimise eesmärgi kasutatud lihtne redaktor baseerus iframe elemendil (tegelikult me testisime ka `contenteditable` atribuudiga ka), lisafunktsioonina sisaldas testkeskkond nuppu, mille vajutus kuvab meile redaktori iframe elemendi lähtekoodi. Meid huvitas, milline on lähtekood peale MS Wordi pilti sisaldava dokumendi kopeerimist sinna aknasse.

Esimeseks katseks valisime pildi MS Wordi dokumendi jaoks Clip Art Gallery kollektsioonist, seejärel selekteerisime pildi ning lisasime *copy-paste* meetodil katseredaktorisse.
Mis juhtus oli see, et Internet Explorer ei näidanud kopeeritud pilti üldse, pildi oodatavasse asukohta ilmus hoopis tühi pildi suurune kast (*image placeholder*); Mozilla näitas pilti ilusti nagu näha järgnevalt pildilt (Pilt 20):

Pilt 20: Kopeeritud pilt Mozilla veebiredaktoris

Siiski pole sugugi lihtne saada seda pilti serverisse. Alljärgnevalt demonstreerime kahte lähtekoodi, mis tekivad pildi kopeerimisel redaktoriaknasse. Esimene koodinäide (Koodinäide 71) on Internet Explorerilt ja teine (Koodinäide 72) Mozillalt.

Koodinäide 71: Kopeeritud pildi lähtekood Internet Exploreri redaktorist

```
<SPAN lang=EN-US style="FONT-SIZE: 12pt; FONT-FAMILY: 'Times New Roman';
 mso-fareast-font-family: 'Times New Roman'; mso-ansi-language: EN-US;
mso-fareast-language: EN-US; mso-bidi-language: AR-SA">
<?xml:namespace prefix = v ns = "urn:schemas-microsoft-com:vml" />
<v:shapetype id=_x0000_t75 stroked="f" filled="f" path="m@4@5l@4@11@9@11@9@5xe"
o:preferrelative="t" o:spt="75" coordsize="21600,21600"> <v:stroke
joinstyle="miter"></v:stroke><v:formulas><v:f eqn="if lineDrawn pixelLineWidth
0"></v:f><v:f eqn="sum @0 1 0"></v:f><v:f eqn="sum 0 0 @1"></v:f>
<v:f eqn="prod @2 1 2"></v:f><v:f eqn="prod @3 21600 pixelWidth"></v:f>
<v:f eqn="prod @3 21600 pixelHeight"></v:f><v:f eqn="sum @0 0 1"></v:f>
<v:f eqn="prod @6 1 2"></v:f><v:f eqn="prod @7 21600 pixelWidth"></v:f>
<v:f eqn="sum @8 21600 0"></v:f><v:f eqn="prod @7 21600 pixelHeight"></v:f>
<v:f eqn="sum @10 21600 0"></v:f></v:formulas>
<v:path o:connecttype="rect" gradientshapeok="t" o:extrusionok="f"></v:path>
<?xml:namespace prefix = o ns = "urn:schemas-microsoft-com:office:office" />
<o:lock aspectratio="t" v:ext="edit"></o:lock></v:shapetype>
<v:shape id=_x0000_i1025 style="WIDTH: 84.75pt; HEIGHT: 90pt"
type="#_x0000_t75"><v:imagedata o:title="pe03166_"
src="file:///C:/WINDOWS/TEMP/msoclip1/01/clip_image001.wmz">
</v:imagedata></v:shape></SPAN>
```

Koodinäide 72: Kopeeritud pildi lähtekood Mozilla redaktorist

```
<span lang="EN-US" style="font-size: 12pt; font-family: &quot;Times New
Roman&quot;;"><!--[if gte vml 1]><v:shapetype
 id="_x0000_t75" coordsize="21600,21600" o:spt="75" o:preferrelative="t"
```

```
 path="m@4@5l@4@11@9@11@9@5xe" filled="f" stroked="f">
 <v:stroke joinstyle="miter"/>
 <v:formulas>
  <v:f eqn="if lineDrawn pixelLineWidth 0"/>
  <v:f eqn="sum @0 1 0"/>
  <v:f eqn="sum 0 0 @1"/>
  <v:f eqn="prod @2 1 2"/>
  <v:f eqn="prod @3 21600 pixelWidth"/>
  <v:f eqn="prod @3 21600 pixelHeight"/>
  <v:f eqn="sum @0 0 1"/>
  <v:f eqn="prod @6 1 2"/>
  <v:f eqn="prod @7 21600 pixelWidth"/>
  <v:f eqn="sum @8 21600 0"/>
  <v:f eqn="prod @7 21600 pixelHeight"/>
  <v:f eqn="sum @10 21600 0"/>
 </v:formulas>
 <v:path o:extrusionok="f" gradientshapeok="t" o:connecttype="rect"/>
 <o:lock v:ext="edit" aspectratio="t"/>
</v:shapetype><v:shape id="_x0000_i1025" type="#_x0000_t75" style='width:84.75pt;
 height:90pt'>
 <v:imagedata src="file:///C:/WINDOWS/TEMP/msoclip1/01/clip_image001.wmz"
  o:title="pe03166_"/>
</v:shape><![endif]--><!--[if       !vml]--><img       width="113"       height="120"
src="file:///C:/WINDOWS/TEMP/msoclip1/01/clip_image002.gif"
v:shapes="_x0000_i1025"><!--[endif]--></span>
```

Nagu näeme on lähtekoodid erinevad. Tegelikult on enamus koodi meile kasutu ja oluline tuleb sealt välja sõeluda. Ainuke info, mis meid tegelikult huvitab sisaldub viimasel paaril real. Seal näeme ka põhjust, miks Mozilla näitab sel moel kopeeritud pilti, kuid Internet Explorer mitte. Nimelt Mozilla lähtekoodis sisaldub `<img>` märgend koos src atribuudiga, mis viitab lokaalsele .gif failile temp kataloogis. Internet Exploreri lähtekoodis ei sisaldu `<img>` märgendit ja seal pole ka viidet .gif laiendiga pildile. On hea teada, et tegelikult see .gif fail on siiski loodud mõlemal juhul samasse temp kataloogi, see ei sõltu brauserist. Täpselt teadmata põhjusel Internet Explorer sellele failile ei viita kopeeritud koodis. Ainus viit, mis seal on, on .wmz failile `<v:imagedata>` märgendi sees. Sama viit sisaldub ka Mozilla versioonis. WMZ on pakitud Windows Metafile failiformaat, mida saab avada MS Wordi abil, kuid mitte veebibrauseris.

Selgituseks veel niipalju, et sama Exploreri kopeeritud osa HTML koodist genereeritakse ka MS Wordi pool, kui katsealust dokumenti salvestada meetodil “Save

as web page". Kuid see pole täpselt sama. Esiteks erineb see juba selle pooles, et kõik minda me redaktoriaknasse lisame copy-paste meetodil, paigutub selle lähtekoodis body elemendi sisse, kuid salvestades faili HTML formaadis, jääb suur hulk informatsiooni väljaspoole `body` elementi. Selleks on lehekülje stiilidefinitsioonid ja html elemendis defineeritud xml nimeruumid ning palju muud.

Peale selle on tähelepanuväärne, et ka `body` elemendist ei lähe kõik redaktorisse üle samamoodi nagu on html formaati salvestatud failil. Nimelt on viimases olemas siiski ka `img` element nagu näha järgmiselt koodinäitelt:

Koodinäide 73:

```
<![if !vml]><img width=246 height=260
src="./katse_files/image002.gif" v:shapes="_x0000_i1025"><![endif]>
```

Kuna see rida puudub Internet Exploreri veebiredaktoris, siis tuleb oletada, et tingimuslikud kommentaarid `<![if !vml]> <![endif]>`, mille eesmärk on pakkuda alternatiivset sisu neile brauseritele, mis ei toeta VML (Vector Markup Language) keelt (W3C 1998) ja Internet Explorer VML keelt toetavana (alates viiendast versioonist) tegelikult ignoreerib seda *img* elementi ja kasutab pildi näitamiseks VML keelt. Redaktoriaknasse seega lihtsalt ei kopeerita seda koodi, mis Internet Exploreri vastava versiooni kohaselt oluline ei ole.

Kuid miks siis pilt nähtavale ei ilmu? Põhjused on nüüd selles, et vektorgraafika näitamiseks peab lehekülje `html` element viitama vastavale nimeruumile: `<html xmlns:v="urn:schemas-microsoft-com:vml">.`
See teatab brauserile, et XML elemendi, mis algab prefixiga v, tõlgendamisel ja esitamisel tuleb ümber lülituda nimeruumile `urn:schemas-microsoft-com:vml` ja anda andmed edasi VML protsessorile. Protsessor rollis on antud juhul fail VGX.DLL. Teiseks tuleb `style` sektsioonis määratleda kasutatud VML elemendi või elementide stiil:

Koodinäide 74

```
<style>
v\:* { behavior: url(#default#VML); }
</style>
```

MS Office genereerib selle stiili ka tingimuslike kommentaaride vahele, et teised brauserid neid interpreteerida ei püüaks. Kasutatud katsefailis oli see järgmine:

Koodinäide 75

```
<!--[if !mso]>
<style>
v\:* {behavior:url(#default#VML);}
o\:* {behavior:url(#default#VML);}
w\:* {behavior:url(#default#VML);}
.shape {behavior:url(#default#VML);}
</style>
<![endif]-->
```

Nagu eelolevast koodinäitest näha, kasutakse stiili määratlemisel siin Internet Exploreri spetsiifilist `behavior` omadust (MSDN Behavior 2005), mille abil saab viidata brauseris vaikimisi realiseeritud võimalustele või enda kirjutatud .htc failile. Alates versioonist 5, kuulub VML brauseris vaikimisi realiseeritud CSS `behavior` omaduste hulka (MSDN VML 1998).
Seega soovides kuvada MS Office dokumendist kopeeritud vektorgraafikat, tuleb alustada lisada juba eelnevalt redaktoriakna päisesse vastav stiil ja `html` elementi VML nimeruumi viide. See lahendaks kõige lihtsamalt visuaalse tagasiside probleemi, kuid ei aitaks kaasa vajaliku faili serverisse laadimisel.

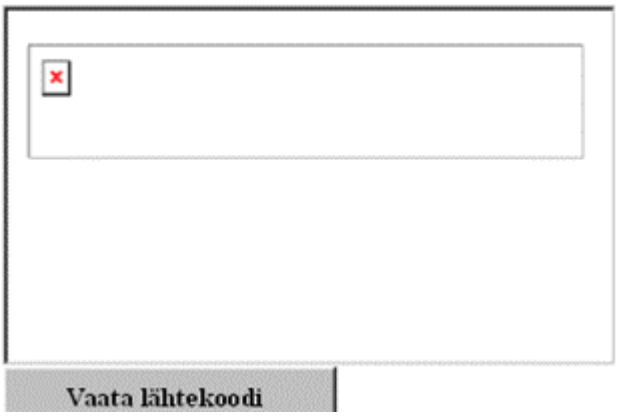
palju muud, m MS Word 8 (Office 97) loob erineva lähtekoodi, mis on natuke puhtam ja lühem. Kui vaadelda sellest seisukohast Mozilla katseeksemplari lähtekoodi, siis samasugune kood tekib, kui kopeerida antud dokument Mozilla Composerisse.

Kui tagasi tulla võimalike põhjuste juurde, mis Internet Exploreri redaktor vektorgraafika formaadis pilte ei näita, siis üheks tõenäoliseks põhjuseks pean MS Wordist HTML failiks salvestamisel lisanu

Järgmises näites me vaatleme, mis juhtub selliste redaktoritega nagu Wordpad, millel enda sisemised HTML salvestamise võimalused puuduvad.

Pildi kopeerimise Wordpadi dokumendist (seesama puudutab ka MS Word versiooni 6) Internet Exploreri redaktoriaknasse ei õnnestunud. Tulemuseks oli vaid tühi kast punase ristiga nagu näha järgmiselt pildilt (Pilt 21).

Pilt 21



Ning seda põhjustav lähtekood oli järgmine:

Koodinäide 76

```
<FONT size=2><P>
<IMG height=51 src="file:///C:/apache/htdocs/Image1.gif" width=272>
</P></FONT>
```

Pilt ei olnud tegelikult viidatud aadressil, vaid asus tegelikult aadressil C:\windows\temp\Image1.gif
Nii juhtus ka teiste objektidega nagu MS Equation ning teised.

Mitte kõike ei saa kopeerida *iframe* elemendil baseeruvasse veebiredaktorisse Näiteks ei saa selekteerida mingit osa programmis Paint ja kopeerida ning asetada

(copy-paste) tulemus veebiredaktorisse. *Paste* käsk pole lihtsalt aktiivne, ning samuti ei anna tulemust klahvikombinatsioon control + v.

Kopeerimine MS Wordist (Office 2000) põhjustab mitmeid eri tulemusi sõltuvalt sellest, millises formaadis pildiga on tegu. Kui lisatud pilt on mingis rasterformaadis (proovisin .bmp, .jpg ja .gif formaate), siis viidatakse lähtekoodis alati .png failile nagu näha järgnevas koodis:

Koodinäide 77: Viide lokaalsele png failile

```
0. <v:imagedata o:title=""
   src="file:///C:/WINDOWS/TEMP/msoclip1/01/clip_image001.png">
</v:imagedata>
```

See fail asubki viidatud kohas and sealsamas kõrval on ka clip_image002.jpg fail samast pildist. Kogu kood, mis redaktorist tule on järgnev:

Koodinäide 78: Rastervormingus pildi kopeerimisel tekkiv lähtekood

```
<SPAN lang=EN-US style="FONT-SIZE: 12pt; FONT-FAMILY: 'Times New
Roman'; mso-fareast-font-family: 'Times New Roman'; mso-ansi-
language: EN-US; mso-fareast-language: EN-US; mso-bidi-language: AR-
SA"><?xml:namespace prefix = v ns = "urn:schemas-microsoft-com:vml"
/><v:shapetype id=_x0000_t75 stroked="f" filled="f"
path="m@4@5l@4@11@9@11@9@5xe" o:preferrelative="t" o:spt="75"
coordsize="21600,21600"> <v:stroke
joinstyle="miter"></v:stroke><v:formulas><v:f eqn="if lineDrawn
pixelLineWidth 0"></v:f><v:f eqn="sum @0 1 0"></v:f><v:f eqn="sum 0 0
@1"></v:f><v:f eqn="prod @2 1 2"></v:f><v:f eqn="prod @3 21600
pixelWidth"></v:f><v:f eqn="prod @3 21600 pixelHeight"></v:f><v:f
eqn="sum @0 0 1"></v:f><v:f eqn="prod @6 1 2"></v:f><v:f eqn="prod @7
21600 pixelWidth"></v:f><v:f eqn="sum @8 21600 0"></v:f><v:f
eqn="prod @7 21600 pixelHeight"></v:f><v:f eqn="sum @10 21600
0"></v:f></v:formulas><v:path o:connecttype="rect"
gradientshapeok="t" o:extrusionok="f"></v:path><?xml:namespace prefix
= o ns = "urn:schemas-microsoft-com:office:office" /><o:lock
aspectratio="t" v:ext="edit"></o:lock></v:shapetype><v:shape
id=_x0000_i1025 style="WIDTH: 244.5pt; HEIGHT: 195pt"
o:borderrightcolor="this" o:borderbottomcolor="this"
o:borderleftcolor="this" o:bordertopcolor="this" o:ole=""
```

```
type="#_x0000_t75"><v:imagedata o:title=""
src="file:///C:/WINDOWS/TEMP/msoclip1/01/clip_image001.png"></v:image
data><?xml:namespace prefix = w ns = "urn:schemas-microsoft-
com:office:word" /><w:bordertop type="single"
width="4"></w:bordertop><w:borderleft type="single"
width="4"></w:borderleft><w:borderbottom type="single"
width="4"></w:borderbottom><w:borderright type="single"
width="4"></w:borderright></v:shape></SPAN>
```

Niisiis on teada pildi asukoht, kuid kas seda on võimalik laadida serverisse? Ainus standardne viis piltide üleslaadimiseks HTTP kaudu on kasutada selleks ette nähtud vormielementi `input` tüübinimega `file`: `<input type="file">`. Oleks kerge JavaScripti abi otsida lähtekoodist viiteid piltidele ja vastavalt sellele muuta üleslaadimisväljade sisu, kuid seda viimast kahjuks teha ei saa. Tavalist `input` välja sisu saab JavaScripti abil muuta, kuid üleslaadimiseks mõeldud input välja täitmise üle peab jääma kasutaja vahetu kontroll. See on veebitehnoloogia puhul ka igati mõistetav, kui neid välju saaks ilma kasutaja teadmata muuta, siis kujutaks see suurt turvariski, kuna suvaline veebileht võiks kasutajate arvutis asuvaid faile nende tahte vastaselt üles laadida.

Seega jääb kolm võimalust:

- Võib teha Internet Exploreri jaoks ActiveX skripti ja Mozilla jaoks XPConnect tehnoloogial rajaneva suuremate privileegidega skripti, mis hakkab tegelema nende failide üleslaadimisega.

- Võib avada spetsiaalse üleslaadimisakna, kus kõik vastavad viited lokaalsele temp kataloogile on ülesloetletud ning soovitada kasutajal need manuaalselt üles laadida.

- Võib ignoreerida sellist kopeerimist täielikult ning luua protseduur, mis sellest ka märku annab, et taoline pildi kopeerimine käesolevas süsteemis ei tööta.

Ükski neist lahendustest ei ole rahuldav. Esimene lahendus ei jätaks välja KHTML mootoril töötavad Konqueror ja Safari brauserid ning viimasena designMode tehnoloogiat toetama asunud Opera. Samuti võib ActiveX tehnoloogia kasutamine ebaõnnestuda rangemate turvaseadete tõttu. ActiveX ja XPConnect tehnoloogia kasutamisega kaasnevad turvadialoogid teadetega sellest, et järgnev tegevus toimub kasutaja omal vastutusel ja võib kahjustada kasutaja arvutit. Need kahandavad üsna tõenäoliselt kasutaja usaldust rakenduse vastu.

Teine lahendus võib osutuda väga tüütavaks, kui on vaja palju erinevaid faile üles laadida. Samuti tuleb arvestada inimliku vea võimalusega, sellega, et kasutaja ei vali üldse õigeid faile.

Kolmas võimalus on tegelikult realiseeritud juba Opera 9 puhul, kus kopeerida saabki ainult teksti. See tähendab muuhulgas, et Opera 9 puhul pole võimalik, teisest dokumendist või veebilehelt kopeerida redaktoriaknasse ka tabeleid ega mingil moel kujundatud teksti. Kahtlemata lihtsustab see süsteemi hallatavust, kuid kasutajale võib selline käitumine tunduda harjumuspäratu.

Siin on kahtlemata koht sügavamateks uuringuteks nii kasutaja ootuste kui ka tehnoloogiliste võimaluste koha pealt. Tuleb jälgida ka standardiseerimisettepanekute tööversioonide arenguid, eriti Apple, Mozilla ja Opera töögrupi WHATWG (*Web, Hypertext Application Technology Working Group*) tööversioone Web Application 1.0 (WHATWG, Web App 2005) ja Web Forms 2.0 (WHATWG, Web Forms 2005). Seni võib jätta asjad nii nagu on ja loota, et selliseid pildi kopeerimisi ei esine tihti ja kui esineb, siis kontrollida pidevalt redaktori lähtekoodi ning koodilõigu ilmnemisel, mis viitab pildifailile lokaalses temp kataloogis, teavitada kasutajat sellise kopeerimise puudustest ja võimalikest lahendustest.

10.2 Tabelite kopeerimine

Teine kopeerimisest tulenev erinevus ilmneb tabelite puhul. Erinevusi pole, kui tegu on Openoffice või MS Office versioonidega, mis toetavad juba HTML formaati

salvestamist. See tähendab MS Office puhul versiooni 97 või enam. Ilmnevad erinevused on seotud vaid tekstitöötlussüsteemiga, millest selekteeritud tabelit kopeeritakse, kuid erinevusi ei ilmne eri veebiredaktoris tekkinud koodis. Erand on vaid Opera, mis kopeerib sel moel vaid paljast teksti.

Kuid erinevused ilmnesid tekstitöötlussüsteemide puhul, kus HTML formaati salvestamise võimalus puudub. Näiteks proovisin, kuidas õnnestub tabelite kopeerimine sellisest tekstiredaktorist nagu MS Wordpad. Nagu teame Wordpadil puudub endal spetsiaalne tabeliredaktor, kuid ta suudab tabeleid kuvada, kui need on sinna kopeeritud teisest tekstiredaktorist nagu näiteks MS Word. Niisiis huvitas mind küsimus, mis juhtub, kui sel moel kopeeritud tabelit püüda Wordpadist kopeerida eksperimentaalsesse veebiredaktorisse. Katse tulemusel tabeli kopeerimine tekstiredaktoritest Wordpad ning MS Word 6 õnnestus Internet Exploreris avatud veebiredaktoris, kuid mitte Mozilla puhul. Nagu juba varem mainisin, ei õnnestu see ka Opera puhul.

10.3 Piltide üleslaadimine

Kui mõelda WYSIWYG redaktori võimaluste kasutamisele testimissüsteemi kontekstis, siis üheks olulisemaks funktsiooniks on kindlasti piltide lisamine. Lihtsaim viis piltide lisamiseks redigeeritavasse dokumenti, on linkida `<img>` märgendis teatud pildi veebiaadressile. See tähendaks, et pildilisamise nupule vajutus avab dialoogi, milles küsitakse pildi aadressi. Mitmed veebiredaktorid (Elioni veebimail, WebCT) vaid sellist võimalust kasutavadki. See on kahtlemata üks võimalus veebipõhises testimissüsteemis ja alternatiivina tuleb sellega arvestada. Kuna tehnoloogiliselt on aadressi lisamine liiga lihtne, et sellel pikemalt peatuda, siis tuleb juttu üleslaadimise realiseerimisest.

10.4 Piirangud suurusele

Kõigepealt on enamikes serverites üleslaaditava faili suuruse piirangud. Kahjuks ei saa ilma suurendatud privileegideta kasutaja arvutis oleva faili suurust mõõta ja teatada,

kui lubatud suurus on ületatud. Ainus, mille kohta saab anda vahetu tagasiside on kontrollida üleslaadimis- väljas oleva faili laiendit ja kui see on sobimatu, siis kasutajat sellest teavitada.

10.5 Ajaline piirang

Tavaliselt on serverites seatud ka ajaline piirang, mille jooksul peab toiming olema sooritatud. Näiteks PHP konfiguratsioonifailis on see vaikimisi seatud 30 sekundile. Juhul, kui on suur kogus üleslaaditavaid faile, siis on risk, et lõplik postitus ületab ajalise piiri ja katkeb.

10.6 Soovimatud postitused küsimuse loomise ajal

Ideaalis ei tohiks testi loomise ajal teha katkestavaid ühendusi serveriga. See põhimõte on hea ühest küljest säästmaks dial-up ühenduse kasutajate raha ning vähendamaks ebamugavust, mis on seotud katkestusega, mille üleslaadimine paratamatult tekitab. Muidugi või suunata üleslaadimispäringu varjatud aknasse ja sellega kasutaja tegevust mitte katkestada, kuid probleem on selles, et pildi lisamisel ilmub ka pilt alles nähtavale peale seda kui see on serverisse jõudnud. Hea oleks kasutada pildi kuvamiseks lokaalset aadressi ja pärast muuta need serveris olevatele vastavaks, kuid kahjuks ei ole see tehnika rakendatav Opera 9 PR versioonis.

Kui testis on ainult ainsad üleslaadimise lahtrid nagu valikvastuste sektsioonis, siis pole probleemi, kõik valitud pildid võivad oodata kuni lõpliku andmete postitamiseni. Ainus, millega riskime on ületada ajalist ja mahulist piirangut. Seetõttu mitmed teenusepakkujad nagu näiteks veebipõhise emailiteenuse pakkuja hot.ee kasutavad sellist tehnikat, et iga manus tuleb üles laadida eraldi enne kirja lõplikku ärasaatmist. Oletame, et meie ei taha sellist tehnikat kasutada, kas meil on siis mõni muu põhjus, miks vahepealne andmete üleslaadimine peaks osutuma vajalikuks testi loomise protsessi keskel?

Tuleb välja et on. WYSIWYG redigeerimise juures on nimelt käsk *insertimage* *execCommand* meetodil, mis nõuab ühe argumendina pildi URLi või asukohta. Selle tulemusena lisatakse redaktoriaknasse kursori positsioonile valitud pilt ja kui me vaatame tekkinud lähtekoodi, siis see on näiteks järgmine:

Koodinäide 79

```
<img src="c:\mingikataloog\mingipilt.jpg">
```

Nagu näha viitab eelolev kood lokaalsel kettal asuvale pildile ja sealt pildi serverisse saatmise probleemid on samad, mis teisest dokumendist kopeeritud piltide puhul. Siiski on nüüd olukord natuke parem ja selles on kaks võimalikku lahendusteed:

Esimene on valida pilt kõigepealt üleslaadimisvälja ja seejärel lugeda selle välja väärtus muutujasse ning anda argumendina *execCommand* meetodile kasutamaks koos *insertimage* käsuga. Tegelikult seda lokaalset viita on vaja vaid testi loomise käigus, et sisestatud pildid redaktoris ka nähtavale ilmuksid. Kui kõik failid on lõpuks üles laetud, siis tuleb lähtekoodis piltide asukoha viidad ära muuta.

Selle lahenduse probleemid on järgmised:

Tuleb simuleerida dialoogiakent, et üleslaadimisvälja kasutada. Simuleerida seetõttu, et üleslaadimisväli peab jääma sama dokumendi sisse. Kuni praeguseni peavad kõik ühe vormi elemendid paiknema ühes aknas, ühe dokumendi sees ühes *form* elemendis. Nagu eelnevalt mainitud, ei ole mingeid võimalusi modifitseerida üleslaadimisvälja sisu skripti abil. Kui mingi võimalus leidub, siis on see turvaauk ja puudutab vaid mingit brauseriverisooni. Sellise turvaaugu peale ei või igatahes lootma jääda.

Kuna ainus võimalus selle lahenduse korral on dialoogiakna simuleerimine, siis saab seda teha näiteks mingi varjatud kihi abil, et teha see vajalikul hetkel nähtavaks JavaScripti abil. Kuid see tekitab järgmise probleemi: nagu mainitud WYSIWYG

redaktorit tutvustavas alapeatükis – *iframe* element evis kuni Internet Exploreri versioonini 5.5 omadust paista kõikidest teistest elementidest läbi.

Kuna redaktoriaken on *iframe* element, siis simuleeritud dialoog avaneks redaktoriakna taga ja kasutaja ei näekski seda. Mozilla ja Internet Explorer 5.5 ning järgnevad võimaldavad kasutada *iframe* elemendil *z-index* stiilimäärangut, mille abil saab kontrollida, milline element on teiste elemendi kohal või all, kuid Internet Explorer 5 kasutajad oleksid probleemi ees.

Tegelikult on üks mitteametlik lahendus juba alates Internet Exploreri versioonist 4, mis võimaldab tekitada pseudokihi ka *iframe* elemendi ja teiste analoogsete (*Windowed Control* tüüpi) elementide kohale. Kuid nähtavus pole ainus probleem.
Tuleb pidevalt peita juba täidetud üleslaadimisvälju ja tekitada uute piltide jaoks uusi. See kõik on teostatav, kuid pidades silmas nüüdseks juba üsna laialt levinud püsiühendusi ning riske, mis kaasnevad sellega, kui kasutaja ületab üleslaadimise ajalise või mahulise piiri, siis tõenäoliselt on otstarbekam leppida vahepealsete täiendavate ühendustega serverisse ja piltide ükshaaval üleslaadimisega samal hetkel kui kasutaja mingi pildi sisestamise sooritab. Seda meetodit kasutasin ka aastal 2004, IVA süsteemi integreeritud WYSIWYG redaktori loomisel.

Selle meetodi eelised on järgnevad:

- Ei pea muutma pildi asukohaviiteid *img* elemendis peale seda, kui testi küsimuse fail on postitatud serverisse. Pilt laaditakse kohe oma kohale ja töödeldavas testis viidatakse temale kohe lõpliku URLi kaudu
- Nagu mainitud, saab nii minimeerida ajalise limiidi ületamise riski
- Ei pea looma jätkuvalt uusi üleslaadimisvälju ning andma neile unikaalseid nimesid. Kõik failid saab üleslaadida läbi ühe ja sama dialoogiakna.

Selle meetodi puudused on järgmised:

- Täiendavad ühendused testi loomise käigus

- Kogu lahendus on rohkem sõltuv serveripoolsest skriptist, mis teeb portimise teistesse süsteemidesse keerulisemaks.

Kolmas võimalus on veel ja see kombineerib enda mõlema eelpoolmainitud tehnika paremad omadused. Piltide kuvamiseks kasutatakse lokaalset aadressi ja seetõttu saab pildi üleslaadimisdialoogi kaotada kohe, kui vajalik protseduur on sooritatud ning pilt ilmub redaktoriaknasse momentaalselt. Pildi tegelik saatmine serverisse toimuks varjatult taustal ja ei häiriks kasutajat.

Selle meetodi ainsaks puuduseks on see, et Opera 9 brauseril ei ole see seni rakendatav kuna Opera oma viimastes versioonides lubab lugeda JavaScripti abil vaid üleslaadimisväljas oleva faili nime, aga mitte lokaalketta täielikku teekonda failini.

See on Opera tahtlik poliitika ja võibolla ajendatud väikestest turvaprobleemidest, mis nende brauseri varasemates versioonides olid (üleslaadimisväljale sai anda vaikeväärtuseks mingi lokaalse faili aadressi). Teatud loogika Opera käitumises on, sest turvaprobleemiks võib ju ka seda pidada, et mingi koduleht JavaScripti abil saab lugeda lokaalse faili teekonda ja seeläbi koguda andmeid kasutaja lokaalse süsteemi kohta.

Kui ka teised brauserid järgivad sama eeskuju, siis on välistatud nii viimati kirjeldatud kombineeritud võimalus, kui ka esimene võimalus, milles koguti üleslaaditavad andmed pidevalt juurdetekitatavatesse üleslaadimisväljadesse. Kuigi üleslaadimine oleks võimalik, siis pildid ei ilmuks redaktoriaknas nähtavale. Seega jääks ainsa võimalusena pilti näidata redaktoriaknas viidates selle aadressile serveris alles siis kui see on serverisse jõudnud.

Kuna tegu on kahtlemata kasutaja tegevuse katkestamisega, siis tuleb pakkuda mingit adekvaatset tagasisidet toimuva kohta. Lihtsaim võimalus on mingi riba, või vilkuv kiri või liivakell, mis teatab, et pilti laetakse serverisse. XMLHttpRequest objekti abil saaks serverist ka tagasisidet selle kohta kui palju täpselt on serverisse laetud ja anda kasutajale sellele vastav tagasiside protsentide või mingi riba pikenemise näol.

10.7 Andmete lohistamise probleem

Mõnikord võib kasutajal tekkida idee lohistada, mingi objekt failihaldurist (pildi fail näiteks) redaktoriaknasse. Kasutaja ootus sellisel juhul on arusaadav, ta püüab sel moel pilti lisada. Internet Exploreris tekib sel juhul olukord, kus kogu redaktoriakna sisu asendatakse failiga, mis sinna on lohistatud. See võib olla väga ebameeldiv, kui kasutaja on eelnevalt juba redaktori sisuga tegelenud ning kogu töö raisku läheb.
Mozilla käitub sellise tegevuse korral üllatavalt adekvaatselt ja kasutaja ootustele vastavalt, lisab pildi, kui lohistatavaks objektiks on pilt ning lisab lingi, kui lohistatav objekt on mingi muu fail. Viited on tehtud muidugi lokaalsetele failidele, sinna, kus nad asuvadki. Vaatamata Mozilla kenale käitumisele antud olukorras, ei aita see kuidagi valitud faile automaatselt üles laadida. Nii esimesel kui teisel juhul tuleb panustada sellele, et kõnealust olukorda üldse tekkida ei saaks.

Need on vaid mõned näited probleemidest ja vigadest, mida võib kasutamise käigus ette tulla. Siin on lai valdkond edasiarendusteks ja põhjalikumateks uuringuteks. Ka tehnoloogia selles valdkonnas areneb praegu jõudsalt ning võib kaasa tuua meeldivaid üllatusi nii kasutajatele kui arendajatele.

11 Kokkuvõte: igavesti beeta

Viimaste aastate jooksul toimunud kiired muudatused veebi arengus on esile tõstnud kaks erinevat rühma: veebilehed ja veebirakendused. Veeb on algsest staatilisest artiklikogumist kujunenud dünaamilisemaks ja võtnud üle palju selliseid funktsioone, mida varem täitsid vaid lokaalsesse arvutisse installeeritud programmid. Sellises ülesannete täitmisele orienteeritud veebis kehtivad teised reeglid ja nõuded kui infole orienteeritud veebis. Esiplaanile tõuseb ülesannete täitmise lihtsus, kiirus ja mugavus ja keskkonna võime adekvaatselt regeerida kasutaja tegevusele. Olulisel kohal on ka harjumuspärasus, mis väljendub sarnasuses lauarvuti analoogsetele rakendustele.

Veebirakenduste traditsiooniline õhukese kliendi mudel ei toeta ühtegi neist nõuetest. Võib öelda, et plussid, mis veebirakenduse kasutamisest tulenevad on olnud pigem süsteemi haldaja ja omaniku poole peal. Kasutajale on jäetud suhteliselt vaene keskkond ja primitiivne interaktsioonimudel.

Siiski on võimalik ka kliendi rikastamine ning seda toetavad veebibrauserite üha avarduvad sisemised võimalused. Paljudele märkamatult on veebibrauseritest saamas täiesti arvestatav kliendi arendusplatvorm, mis võimaldab märksa paremini tasakaalustada rakenduse loogika jaotust kliendi ja serveri poolel. Kliendipoolsete tehnoloogiate varasemaid arenguid iseloomustanud ühilduvusprobleemid on vähenenud ning uusi tehnoloogiaid kasutavad populaarsed teenused sunnivad positiivset arengut jätkuma.

Kuid veeb tarkvaraarenduse platvormina on siiski midagi niivõrd hajusat ja määratlematut, et selle puhul ei kehti enam vanad tarkvaraarenduse reeglid. Kõigepealt tuleb see sellest, et kliendipoolset tarkvara ei saa arendada konkreetsele platvormile vaid muutuvale ja kiiresti arenevale süsteemile, kus konkreetsuse asemel valitseb suhteliselt suur paljusus. Keegi ei tea, milline on kasutaja tehniline varustatus, milline on tema brauser, operatsioonisüsteem, arvuti ja millised on selle seaded, aga me tahame, et rakendus töötaks.

Teiseks ei saa kliendi rakendust arendada ühe spetsifikatsiooni järgi. Selle asemel tuleb uurida eri brauserite kohati puudulikke spetsifikatsioone ja tugineda isiklikule kogemusele või katseeksitus meetodile.

Kolmandaks ei saa kliendipoolsele skriptimisele rajatud rakenduste arendamisel kasutada teiste poolt kirjutatud universaalseid funktsioonide kogumikke. Iga bait loeb ja seetõttu tuleb laadida infot vaid nii vähe kui võimalik ning kui vaja siis tuleb seda parajate portsude kaupa juurde laadida.

Neljandaks ei toimi enam plaanipärane arendustöö, mis kestab tarkvara versioonist järgmise versioonini. Selle asemel on pidev kohandamine ja adekvaatne reageerimine olukorrale. Veebirakendused keskselt hallatavate süsteemidena annavad selleks väga hea võimaluse. Nagu bioloogilises evolutsioonis puuduvad versiooninumbrid ja lõppversioonid nii on ka veebirakendus süsteem, mis kujuneb üksikute loomulike valikute kaudu järkjärgult paremaks, mitte ei planeerita teda algusest peale mingi oletusliku ideaali järgi. Selles mõttes on hea veebirakendus igavesti beeta.

12 Summary

WWW was designed as medium for Hypertext documents. Therefore the interaction model defined by HTTP standard was good enough for requesting documents from remote server but today it is not sufficient any more. It's because today we have Applications in the web. Difference between Web pages and Web applications is in purpose. When Web Site is Content Oriented, then Web Application is Task Oriented.

Naturally there are different requirements too for Applications and Web Sites. When for Web Site design we can find many Guidelines (actually Google search gives for 3,700,000 of them) then for Web Application there isn't much. Maybe it is because of novelty of this kind of Applications, maybe because of not ease to differentiate Web sites and Web Applications. But maybe because Applications are Applications – no difference are they it in the Web or desktop, anyway they must support certain tasks; they must be responsible and easy to use.

In this paper I support this last opinion. I analyze most common Web Usability Guidelines and show what different meaning they have in context of Web sites in context of Web Applications. Web applications can much more benefit from Human Interface Guidelines published by Apple Computers or an analogue from Microsoft, than from mostly content centric Web Usability Guidelines mentioned above.
But supporting conventions familiar in desktop applications isn't easy in web client. Web Applications are among client-server systems usually so called thin client systems and that means limited interactions. Nowadays we used to see Applications as highly interactive environments with direct manipulation and WYSIWYG capabilities, but in thin client system we go back to the age of terminals, command prompts and form fills.

I shall show that during 15 year of evolution Web browser has become from simple document viewer to powerful development platform for client side Application layer. DOM, EcmaScript, CSS are good standardized development tools. Many not yet standardized objects and methods like XMLHttpRequest, which enables developers to establish interaction between Client and Server without loading new Pages,

document.designMode property, that enables WYSIWYG editing on the Web and others techniques are already very well supported by different versions of Browsers.

Then different techniques for client enrichment will be discussed by historical examples and by code examples for achieving different behaviors, which are usual for desktop applications. In last part I describe building process particular prototypes for Learning System IVA Testing environment. Thus functional prototypes include Multiwindow WYSIWYG editor for creating Quizzes, Questionnaires and so on. Multiwindow means that not only Question part can be edited with Rich Editing capabilities, but also choice answers and feedback fields. Different data fields are manipulated with same button bar, enabling to format text, insert hyperlinks, images and tables. Some of techniques I describe here aren't (I have strong reason to believe it) documented anywhere before.

Other prototypes are based on Drag & Drop technique. One of them is providing new Question type for IVA testing system, namely ordering test. Another test type is matching test, which was till now based poorly on select fields.

Web Application client development differ from desktop client development. Here we have limited resources like download speed and browser speed, not mentioning browser support. But we do not have to worry about distribution and maintenance, as much as desktop application developers have to. We do not have certain specification but evolutionary environment with multiple different specifications and some common denominator. Designing in such environment has to be evolutionary too. There is no big plan, there is no discrete software releases but continuing little changes and improvements. Like in biological world, it is Forever Beta Software.

13 Bibliograafia

1. Addision, E. R. (2003), Leveraging the Horizon: Secrets of a Serial Entrepreneur, iUniverse
2. Apache mod_proxy, http://httpd.apache.org/docs/2.0/mod/mod_proxy.html, 10,2005
3. Apple Human Interface Guidelines, (2005), http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines/OSXHIGuidelines.pdf, 10.2005
4. Apstest, (2004), Apstesti kodulehekülg, http://www.ce.ut.ee/apstest/, 10.2005
5. Baxley, B. (2003), What is a Web Application?, Boxes and Arrows, http://www.boxesandarrows.com/archives/what_is_a_web_application.php, 07.2005
6. Baxley, B. (2002), Making the Web Work: Designing Effective Web Applications, New Riders
7. Berners-Lee, T. (2003), World Wide Web Consortium Presents US Patent Office with Evidence Invalidating Eolas Patent, W3C, http://www.w3.org/2003/10/28-906-briefing.html.en, 10.2004
8. Betz, F. (2003). Managing Technological Innovation: Competitive Advantage from Change, Wiley-IEEE
9. Bosworth, A. (1998), Microsoft's Vision for XML, http://www.infoloom.com/gcaconfs/WEB/paris98/bosworth.HTM, 11.2005
10. Bosworth, A. (2005), Intelligent Disain vs Intelligent Reaction, http://salesforce.breezecentral.com/intelligentreaction/, 11.2005
11. Csikszentmihalyi, M, (1991), Flow: The Psychology of Optimal Experience, Harper & Perennial, New York
12. DasGupta, S. (1991), Design Theory and Computer Science: Processes and Methodology of Computer Systems Design, Cambridge University Press
13. Davis, T. (2003), UC, Eolas win verdict against Microsoft in Web browser case; Federal jury awards $520.6 million in damages. University of Clifornia, http://www.ucop.edu/news/archives/2003/aug11art1.htm, 10.2005
14. Dawkins, R. (1999) The Extended Phenotype: The long reach of the Gene. Revised Edition, Oxford University Press

15. DOM (1998), Document Object Model (DOM) Level 1.0 Specification, W3C, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/, 04.2004
16. ECMA Internatinal, (2003), "Web Leaders agree to add Native XML to ECMAScript" http://www.ecma-international.org/news/ECMA%20E4X%20Final%20Final.pdf, 05.2005
17. Ecmascript, http://www.ecma-international.org/publications/standards/Ecma-262.htm, 10.2005
18. Eich, B. (1998), Making Web Pages Come Alive, Netscape Columns TechVison, http://wp.netscape.com/comprod/columns/techvision/innovators_be.html, 11.2005
19. Eolas Techologies, (1999), EOLAS SUES MICROSOFT.., 10.2005
20. Flanagan, D. (1997), User Interfaces with HTML and JavaScript, Advancing HTML: Style and Substance Volume 2, Issue 1 (Winter 1997), http://www.w3j.com/5/s3.flanagan.html, 09.2005
21. Foley, J. D. (1997), Computer Graphics: Principles and Practice in C, Addison Wesley
22. Fraser, J. (2005), It's a Whole New Internet, Adaptive Path, http://www.adaptivepath.com/publications/essays/archives/000430.php, 06.2005
23. Fried, I, (2005), Microsoft tweaks browser to avoid liability, CNET, http://news.com.com/Microsoft+tweaks+browser+to+avoid+liability/2100-1012_3-5980658.html, 11.2005
24. Garret, J. J., (2002), Elements of User Experience, New Riders.
25. Garret, J. J., (2005), Ajax: A New Approach to Web Applications, Adaptive Path, 2005, http://www.adaptivepath.com/publications/essays/archives/000385.php, 05.2005
26. Gecko DOM, InsertBefore Method, http://www.mozilla.org/docs/dom/domref/dom_el_ref47.html, 05.2005
27. Gell-Mann, M. (1995), Quark and Jaguar: Adventures in the Simple and the Complex, W.H. Freemanan Company
28. Gmail, http://gmail.google.com, 05.2005
29. Google Maps, http://maps.google.com, 05.2005
30. Google Suggest, http://www.google.com/webhp?complete=1&hl=en, 05.2005
31. Gordon, S. (2003), Computing Information Technology: The Human Side, Idea Group
32. Goodman, D. (2003), JavaScript & DHTML Cookbook, O'Reilly

33. Herrnstein J. R., Murray C. (1996), Bell Curve: intelligence and Class Structure in American Life, Free Press
34. Hotmail, http://www.hotmail.com, 07.2005
35. HTMLArea, A directory of browser-based WYSIWYG editors, http://www.htmlarea.com/, 05.2004
36. Hutchins E. L., Hollan J. D., Norman D. A. (1985), "Direct Manipulation Interfaces", HUMAN-COMPUTER INTERACTION, 1985, Volume 1, pp. 311-338, http://hci.ucsd.edu/120/direct-manip.pdf, 05.2005
37. IMS QTI, IMS Question & Test Interoperability Specification, http://www.imsglobal.org/question/, 11.2005
38. IVA, E-õppekeskkond IVA, http://iva.htk.tlu.ee, 11.2005
39. Johnson, J. (2000). Web Bloopers: 60 Common Web Design mistakes and How to Avoid Them, Morgan Kaufmann
40. Järvinen, P. (2001). Reaserch Methods, Opinpajan Kirja
41. King, A. B. (2002), Speed Up Your Site: Web Site Optimization, New Riders
42. Krug, S. (2006), Don't Make Me Think, A Commaon Sense Approach to Web Usability, Second Edition, New Riders
43. Lindstedt, P., Burenius, J. (2003). The Value Model: How to Master Product Development and Create Unrivalled Customer Value, Nimba
44. Lynx, Lynx Browser Info, http://lynx.browser.org/, 05.2005
45. Maaameti teenused, http://www.maaamet.ee/index.php?lang_id=1&page_id=5&menu_id=64, 08.2005
46. Macromedia Flash, http://www.macromedia.com/shockwave/download/download.cgi?P1_Prod_Version=ShockwaveFlash, 08.2005
47. Massy, D. (2000), The Importance of Good Dialog, Microsoft Corporation, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndude/html/dude06052000.asp, 04.2005
48. McGurn, J. (1987). An Illustrated History of Cycling, John Murray Publishers, London
49. Miller, G. (1956). The Psychological Review, 1956, vol. 63, pp. 81-97
50. Miller, R, B. (1968). Response Time in Man Computer conversational transactions. AFIPS Spring Joint Computer Conference Vol. 33, 267-277

51. Mozilla Rich Text Editing Specification (2002), http://www.mozilla.org/editor/midas-spec.html, v. 11.2003
52. Mozillazine (2005), Publish: Mozilla Firefox 1.1 to Support SVG, http://www.mozillazine.org/talkback.html?article=6393, 10.2005
53. MSDM VML, Basic Topics, Appendix, 1998, http://msdn.microsoft.com/library/default.asp?url=/workshop/author/vml/shape/introduction.asp, 04.2005
54. MSDN CreateElement, http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/methods/createelement.asp, 04.2005
55. MSDN Home, http://msdn.microsoft.com/workshop/author/dhtml/reference/properties/designmode.asp, 11.2004
56. MSDN Introduction to DHTML behaviors, (2005) http://msdn.microsoft.com/library/default.asp?url=/workshop/author/behaviors/overview.asp, 11.2005
57. MSDN InsertBefore Method, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/xmlsdk/html/xmmthinsertbefore.asp, 11.2005
58. Murugesan, S., Desphande, Y. (2001), Web Engineering, Springer
59. Mänd, M. (1999), ee.www.webmasters, 25.12.1999, http://groups.google.com/group/ee.arvutid.www.webmasters/tree/browse_frm/thread/ea9068275803b60b/255f90545b8f7886?rnum=1&hl=et&q=Brauseri+asemel+%C3%BCtle+sirver+(sirvija%2C+minu+isetreitud+s%C3%B5na)&_done=%2Fgroup%2Fee.arvutid.www.webmasters%2Fbrowse_frm%2Fthread%2Fea9068275803b60b%2Fc495656df9e72f34%3Flnk%3Dst%26q%3DBrauseri+asemel+%C3%BCtle+sirver+(sirvija%2C+minu+isetreitud+s%C3%B5na)%26rnum%3D1%26hl%3Det%26#doc_906c8a90aea54a77, 11.2005
60. Nielsen, J. (1993), Usablity Engineering, Morgan Kaufman, Academic Press
61. Nielsen, J. (1997), Need for Speed, http://www.useit.com/alertbox/9703a.html, 11.2003
62. Nielsen, J. (1996), Why Frames Suck, http://www.useit.com/alertbox/9612.html, 12.2005

63. Nielsen, J. (2005), R.I.P. WYSIWYG, http://www.useit.com/alertbox/wysiwyg.html, 11.2003
64. Nilesen, J. (1997), Changes in web usebility since 1994, Alerbox: http://www.useit.com/alertbox/9712a.html, 11.2003
65. Norman, D. (2002), The Design Of Everyday Things, 2002 Edition, Basic Books
66. Norman, D. (2003), Emotional Design: Why We Love (or Hate) EveryDay Things, Basic Books
67. OneStat (2005), Mozilla's browsers global usage share is still growing according to OneStat.com, November 2 2005, http://www.onestat.com/html/aboutus_pressbox40_browser_market_firefox_growing.html, 11.2005
68. Opera Forum (2003), http://my.opera.com/community/forums/topic.dml?id=15362, 05.2004
69. Opera Changelog (2004), Opera Changelog 24.08.2004, http://snapshot.opera.com/windows/w760p1.html, 02.2005
70. Opera (2005), Ahead of the game: Opera Introduces Native SVG Support in Desktop Release, http://www.opera.com/pressreleases/en/2005/03/16/, 08.2005
71. Patent 5838906, (1998), http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=/netahtml/srchnum.htm&r=1&f=G&l=50&s1=5,838,906.WKU.&OS=PN/5,838,906&RS=PN/5,838,906, 11.2005
72. Regio (2005), Regio Interaktiivne Eesti kaart, http://www.regio.ee/?op=body&id=24, 09.2005
73. Selvidge P. (1999), How Long is Too Long to Wait for a Website to Load? Usability News 1999 1.2.
74. Shneiderman, B. (1982), "The future of interactive systems and the emergence of direct manipulation". Behavior and Information Technology, I , 237-256.
75. Shneiderman, B. (1983), "Direct manipulation: a step beyond programming languages," IEEE Computer 16(8) (August 1983), 57-69.
76. Shneiderman, B. (2003), "Leonardo's Laptop: Human needs And The New Computing Technologies, The MIT Press
77. SVG: Scalable Vector Graphics (SVG), 1.1 Specification, W3C, http://www.w3.org/TR/SVG/, 08.2005
78. Tulving, E. (2002), Mälu, Tartu Ülikooli Kirjastus

79. W3C Recommendation, (2004), DOM Level 3 Load and Save Specification, http://www.w3.org/TR/DOM-Level-3-LS/, 08.2005
80. W3C CSS, Cascading Stye Sheet, http://www.w3.org/Style/CSS/, 11.2003
81. W3C DOM, Document Object Model, http://www.w3.org/DOM/, 11.2003
82. W3C VML, (1998) Vector Markup Language, http://www.w3.org/TR/1998/NOTE-VML-19980513, 11.2003
83. W3Schools, (2005), Browser Statistics, http://www.w3schools.com/browsers/browsers_stats.asp, 11.2005
84. Wayner, P. (2005), AJAX breathes new life into Web apps, Infoworld, http://www.infoworld.com/article/05/05/23/21FEwebapp_1.html 05.2005
85. Web Analytics (2005), Firefox's Market Share Nears 7 Percent, WebSideStory, http://www.websidestory.com/products/web-analytics/datainsights/spotlight/05-10-2005.html, 08.2005
86. WHATWG Web Apps 1.0, (2005), Web Applications 1.0 Working Draft, http://whatwg.org/specs/web-apps/current-work/, 11.2005
87. WHATWG Web Controls 1.0, (2004), Web Controls Working 1.0 Draft, http://whatwg.org/specs/web-controls/current-work/, 11.2005
88. WHATWG Web Form 2.0, (2005), Web Forms 2.0 Working Draft, http://whatwg.org/specs/web-forms/current-work/, 11.2005
89. Writely, http://www.writely.com, 08.2005
90. Yahoo, http://www.yahoo.com, 11.2005
91. Yang, K., EI-Haik, B. S. (2003). Design for Six Sigma, McGraw-Hill