PHP-SKRIPTIEN AJASTETTU KÄYTTÖ UNIX-YMPÄRISTÖSSÄ: CASE FUTURALITY-OPPIMISYMPÄRISTÖN AUTOMATISOINTI

Jarmo Kortetjärvi

Opinnäytetyö
Marraskuu 2011
Tietojenkäsittelyn koulutusohjelma
Tietoverkkopalveluiden suuntautumisvaihtoehto
Tampereen ammattikorkeakoulu

TAMPEREEN AMMATTIKORKEAKOULU

Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Tietoverkkopalveluiden suuntautumisvaihtoehto

KORTETJÄRVI, JARMO: PHP-skriptien ajastettu käyttö UNIX-ympäristössä: Case Futurality-oppimisympäristön automatisointi
Opinnäytetyö 49 sivua
Joulukuu 2011

Opinnäytetyössä perehdyttiin PHP-skriptien ajon ajastukseen ja automatisointiin UNIX-webpalvelimella. Käsiteltäviä tekniikoita tarkasteltiin ajastuksen toteutustavan valinnan kannalta perehtyen vahvuuksiin, heikkouksiin ja vaatimuksiin. Ratkaisujen käyttöönottoon ei syvennytty. Toimeksiantajana oli sähköisten oppimisympäristöjen kehittämiseen erikoistunut yritys Futurable Oy, jonka toimeksiantona oli selvittää toimivin tapa toteuttaa webpalvelimen ajasta riippuvaisia automatisoituja toimintoja ja implementoida valittu tekniikka tuotantoympäristöön. Automatisoidut toiminnot hoitavat virtuaalisen oppimisympäristön simulointia ja taustaprosesseja, kuten tietokantojen automaattista päivitystä, määrämuotoisten raporttien luontia, virtuaaliympäristön taustalla tapahtuvaa laskentaa sekä kuukausittaisia ylläpitotoimia.

Tavoitteena oli koota kattavat pohjatiedot erilaisista ajastuksen toteutustavoista ja mahdollistaa parhaan mahdollisen tekniikan valitseminen vertailemalla vaihtoehtojen toimivuutta erilaisissa tilanteissa. Oikean toteutustavan valinta oli oleellista toivotulla tavalla toimivan automaation saavuttamiseksi, sillä eri vaihtoehdoissa on merkittäviä eroja palvelinvaatimuksissa, ylläpitorasitteessa, luotettavuudessa, käyttöönotossa, laajennettavuudessa ja toiminnallisissa rajoituksissa. Työn tarkoituksena oli löytää sopivin tapa toteuttaa virtuaalisen oppimisympäristö Futuralityn tarvitsemat ajastukset ja toteuttaa se tuotantoympäristöön.

Avainsanat: UNIX, PHP, ajastus, automatisoidut toiminnot

ABSTRACT

Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Special Option of Network Services

KORTETJÄRVI, JARMO: Scheduling PHP-scripts in UNIX environment:
Case Automating Futurality Learning Environment
Bachelors thesis 49 pages
December 2011

This thesis focuses on running scheduled and automated PHP-scripts on UNIX web server. The chosen techniques are examined considering the selection of the method of executing the scheduling. The techniques are compared according to their strengths, weaknesses and requirements. The implementation process of these techniques is not treated within the thesis. The client for the study is Futurable - a company specializing in developing e-learning. The assignment was to research the best way to carry out time-based automated tasks on web server, and to implement the selected method to their production environment. Automated tasks are supposed to take care of simulation and background processes of a virtual learning environment. These tasks include automatically updating databases, creating pre-formatted reports, handling the required math behind the virtual environment and doing monthly maintenance.

The aim of the thesis was to gather comprehensive information about different methods of handling scheduled tasks, and to provide tools for selecting the best possible technique by comparing the functionality of the various alternative solutions in different situations. Selecting the proper technique is essential for achieving a solution that meets the requirements of the task as all the alternatives have significant differences in their server and maintenance requirements, reliability, implementation, scalability and functionality limitations. The goal of the thesis was to find and implement the most suitable scheduling technique to the virtual learning environment Futurality.

Keywords: UNIX, PHP, scheduling, automated tasks

SISÄLTÖ

KESKEISIÄ KÄSITTEITÄ

Anacron

Ajastukseen ja automatisointiin käytettävä ohjelma UNIX-pohjaisille käyttöjärjestelmille. Mikäli ajastus on merkitty toteutettavaksi Anacronin ollessa pois päältä, osaa Anacron suorittaa sen takautuvasti.

Apache httpd

Avoimen lähdekoodin webpalvelinohjelmisto, joka toimii useissa käyttöjärjestelmissä. Apache httpd on Netcraftin (2011) tekemän tutkimuksen mukaan tällä hetkellä maailman käytetyin webpalvelinohjelmisto yli 65% suhteellisella käyttöosuudella (Netcraft, 2011).

Cron

Kuten *Anacron*, mutta ei osaa suorittaa ajastuksia takautuvasti. Mikäli ajastus on merkitty toteutettavaksi Cronin ollessa pois päältä, jää se suorittamatta.

crontab

Cron-ohjelman luettavaksi tarkoitetut taulukkomuotoiset tiedostot, joissa määritellään komentoja ja niiden ajastushetkiä, sekä joskus käyttäjä.

Daemon

UNIX-ympäristössä toimiva taustaprosessi. Daemon-ohjelmat eroavat muista ohjelmista siinä, että ne ovat jatkuvasti päällä sen sijaan, että niiden ajo tehtäisiin vain käyttäjän pyynnöstä. Daemon-ohjelmat ovat yleensä joko järjestelmän ylläpitoon liittyviä (mm. *Anacron, Cron)* tai palvelutyyppisiä (mm. *MySQL*, *Apache httpd*), ohjelmia.

ERP

Enterprise Resource Planning, toiminnanohjausjärjestelmä. Ohjelma, joka sisältää yritystoiminnan työkaluja, kuten kirjanpito, palkanlaskenta, osto- ja myyntireskontrat, varastonhallinta ja tuotannonohjaus.

Futurality

Futurable Oy:n kehittämä virtuaalinen oppimisympäristö. Futurality on kehitetty toimimaan *Pupesoft*-toiminnanohjausjärjestelmän kanssa.

LAMP

LAMP on avoimen lähdekoodin tekniikoilla toteutettu webpalvelinratkaisu. Lyhenne sanoista *Linux, Apache httpd, MySQL* ja *PHP*. Lyhenteen ”P” saattaa tarkoittaa myös *Perl*- tai *Python*-kieltä, mutta tämän tutkimuksen yhteydessä se tarkoittaa aina PHP:ta.

MTA

Message Transfer Agent / *Mail Transfer Agent*. Sähköpostin välittäjäohjelma, jonka tehtävänä on vastaanottaa ja lähettää sähköpostia.

MySQL

My Structured Query Language, relaatiotietokantaohjelmisto, joka on toteutettu avoimelle GNU GPL lisenssille.

PHP

PHP: Hypertext Preprocessor on avoimen lähdekoodin ohjelmointikieli dynaamisten websivujen luontiin. PHP on suosittu sen helppouden ja yhteensopivuuden vuoksi. Sitä voi ajaa useilla eri alustoilla, ja se toimii monien tietokantasovellusten kanssa.

Shell Script

UNIX-järjestelmissä käytettävä ohjelmointikieli, jolla järjestelmälle voidaan tehdä komentoja. Shell scriptillä tehdyt ohjelmat tulkitaan nimensä mukaan shellissä, eli UNIXin sisäisessä komentotulkissa (kuten sh tai bash).

Timestamp

Aikaleima, eli yksiselitteisen ajan hetken kuvaus. Vaikka aikaleimaksi voitaisiin periaatteessa laskea ajan kuvaus missä tahansa ajan formaatissa, on aikaleima yleensä joko ”*UNIX Epoch*”-muodossa (montako sekuntia on kulunut hetkestä 1.1.1970), tai ISO-formaatissa (vvvv-kk-pp) tai (vvvvv-kk-pp tt:mm:ss).

UNIX

Käyttöjärjestelmä, jonka pohjalta on kehitetty paljon suosittuja avoimen ja suljetun lähdekoodin käyttöjärjestelmiä, kuten Linux, BSD, Sun OS ja Mac OS X.

(X)HTML

(eXtensible) Hyper Text Markup Language on websivujen rakentamiseen tarkoitettu merkintäkieli. XHTML vastaa muuten täysin HTML-kieltä, mutta sen muotoilulla on tarkemmat säännöt. XHTML:n täytyy olla XML-muotoista, ja vastata käytettyä XML-määrittelyä.

XML

eXtensive Markup Language on standardi, jonka avulla tekstimuotoisen dokumentin sisältämä tieto voidaan muotoilla. XML-muotoiltu data on helposti siirrettävissä kaikkien XML-standardia tukevien ohjelmien välillä.

1 JOHDANTO

Sähköisiin oppimisympäristöihin erikoistunut yritys, Futurable Oy, kehittää virtuaalista Futurality-oppimisympäristöä, josta on tarkoitus saada vuoden 2011 viimeiselle neljännekselle valmiiksi beta-versio, ja edelleen vuoden 2012 ensimmäiselle neljännekselle toimiva pilottiversio. Futurablen tuotesivuilla Futuralitya kuvataan seuraavasti: ”*Futurality on pk-yrityksen liiketoimintaympäristö pienoiskoossa. Futuralityn markkinoilla on kysyntää kaikenlaisille tuotteille ja palveluille*” (Futurable 2011). Futurality yhdistää toiminnanohjausjärjestelmän virtuaaliseen oppimisympäristöön, ja mahdollistaa opiskelun realistisessa ympäristössä käyttäen reaalimaailman työkaluja. Oikeiden yritysten käyttämiä työkaluja opiskelussa hyödyntäneen on helpompi siirtyä harjoitusympäristöstä todellisen maailman yritystoimintaan mukaan.

Opinnäytetyön tavoitteena on tutkia PHP:lla toteutettujen skriptien käyttöä ajastetusti ja automatisoidusti. Työ keskittyy UNIX-ympäristöön, mutta monet ajastusten toteutustavoista ovat täysin toimivia myös esimerkiksi Windows-palvelimilla. Automatisoidun ajon toteutukselle on olemassa lukuisia eri vaihtoehtoja, joista ajastusta tarvitsevan on valittava sopiva. Toteutuksen onnistumisen ja optimaalisen toiminnan kannalta on tärkeää punnita tarkkaan eri vaihtoehtojen edut ja rajoitukset. Ajastustekniikkaa valittaessa täytyy huomioida sekä käyttötarpeen vaatimukset, mutta myös toteutusympäristön rajoitteet. Osa ratkaisuvaihtoehdoista täytyy rajata pois, jos toteutus tehdään sellaiseen ympäristöön, jossa käyttäjällä ei ole mahdollisuutta asentaa lisäohjelmia tai käyttöoikeuksia kaikkiin tarvittaviin toimintoihin.

Tarkoituksena on löytää ja toteuttaa Futurality-oppimisympäristön tarpeisiin sopiva tapa toteuttaa sen toiminnan kannalta välttämättömiä toimintoja. Tekniikan tulee olla mahdollisuuksien mukaan niihin kaikkiin sopiva. Ajastusta tarvitaan skriptien ajastettuun ajoon erilaisilla aikaintervalleilla, raporttien automaattiseen luomiseen, sekä tietokannan taulujen päivittämiseen RSS-feedeistä. Toteutus tulee toimimaan Futuralityn taustalla ohjaillen kaikkia sen tarvitsemia automatisoituja ja ajastettuja toimintoja. Tutkimuksen toiminnallinen osa on tehty Futurality-testipalvelimella, jonka käyttöjärjestelmänä on Fedora Linux (versio 13).

2 WEBSOVELLUKSET PALVELIMEN KANNALTA

2.1 Websivujen ja -sovellusten toteutus

Suunniteltaessa ajastuksia webkäyttöön tarkoitetuille tekniikoille täytyy aivan ensimmäisenä tehdä pintapuolinen katsaus websivujen ja -sovellusten rakentamiseen käytettäviin tekniikoihin. Websivujen monipuolistuessa ja niiden toiminnallisuuden vaatimusten kasvaessa, vaaditaan sivustojen luonnissa entistä monimutkaisempia ratkaisuja. Ohjelmistosovelluksen kaltaisia websivuja tai niiden osia voidaan kutsua websovelluksiksi, joskaan tarkkaa rajanvetoa sivun ja sovelluksen välille on vaikea tehdä.

Yksinkertaisimmillaan sivut voi toteuttaa pelkällä (X)HTML-koodilla, mutta näin luodut sivut ovat toiminnaltaan hyvin rajoitetut, eikä palveluihin voida luoda juurikaan interaktiota. Staattisesti toteutetuilla sivuilla voidaan tarjota lähinnä lehteä tai kirjaa vastaavia palveluita, eli käytännössä luettavaa tietoa. Vastaavasti dynaamisesti luoduilla ja interaktiivisilla sivuilla voidaan tarjota käyttäjälle monipuolisia sähköisiä palveluita. Dynaamisesti luotavien ja interaktiivisten websovellusten rakentamiseen käytetään ohjelmointikieliä, kuten ASP, PHP, JavaScript, ActionScript ja Java. Ohjelmoinnin apuna voi olla myös ohjelmointiympäristö, kuten Microsoft Silverlight tai Adobe Flash.

Loppukäyttäjän näkökulmasta toimintaperiaate on valitusta tekniikasta riippumatta sama: käyttäjä, eli asiakas, pyytää palvelimelta webkäyttöliittymän, eli Internet-selaimen, kautta palveluita joita palvelin toteuttaa. Vaikka käytännössä palvelut suoritetaan ainakin osittain asiakaskoneella, täytyy suoritettava koodi aluksi pyytää palvelimelta. Asiakaskoneella suoritettavia tekniikoita aiemmin luetelluista ovat kaikki muut paitsi PHP ja ASP. Palveluita ei ole välttämätöntä toteuttaa yhdellä tekniikalla, vaan esimerkiksi JavaScriptin ja PHP:n käyttö yhteistyössä on yleistä.

Asiakaskoneella suoritettavien tekniikoiden ongelmana on, että ne tarvitsevat yleensä selainliitännäisen toimiakseen. Silverlightilla tehdyt sovellukset vaativat toimiakseen ”Microsoft Silverlight”-ohjelman (Microsoft 2011), Flashilla toteutetut sovellukset tarvitsevat toimiakseen ”Adobe Flash Player”-ohjelman (Adobe 2011) ja Javalla toteutetut sovellukset tarvitsevat ”Java Web Start”-

ohjelman (Oracle 2011). Jokainen liitännäinen täytyy normaalisti asentaa selaimeen erikseen, jotta sillä toteutettuja sivuja, tai sivujen osia voidaan käyttää.

Osittaisena poikkeuksena on JavaScript, jonka tulkki on ainakin sisäänrakennettuna yleisimmissä selaimissa, kuten *Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, Apple Safari* ja *Opera*. Luotettavaa kaikkia internetin käyttäjiä koskevaa on niukasti. Esimerkkinä kuitenkin huomattavasti rajatumpi tilasto W3Schools-sivuston käyttäjistä, joista elokuussa 2011 yli 99% käytti jotain näistä kuudesta selaimesta (W3Schools 2011). W3Schoolsin käyttäjäkunta ei kuitenkaan vastaa rakenteeltaan koko internetin käyttäjäkuntaa, vaan on enemmän webkehittäjäpainotteista. Kuitenkin tämänkin selainjakauman perusteella, voidaan suhteellisella varmuudella luottaa lähes jokaisen käyttäjän selaimen pystyvän toteuttamaan JavaScriptiä käyttäviä sivuja.

Palvelinpuolen tekniikoiden etuna onkin niiden toimintavarmuus kaikilla käyttäjillä. Näiden kielten yhteisenä ominaisuutena on se, että kaikki tuottavat lopulta normaalia (X)HTML-koodia. Erona tekniikoilla on tekninen toteutus: JavaScriptin ajettava koodi ladataan palvelimelta asiakaskoneen selaimelle, jossa suoritus tehdään. ASP- ja PHP-koodin kutsu tehdään yleensä myös asiakaskoneelta, mutta koodi ajetaankin palvelimella, ja vasta toteutettu (X)HTML-koodi lähetetään asiakaskoneen esitettäväksi.

Lopputuloksena kaikissa kolmessa tekniikassa on kuitenkin asiakaskoneen selaimella esitettävä (X)HTML-koodi, eli websivu. JavaScriptiä käyttäessä palvelimelle ei tarvitse tehdä joka toiminnon kohdalla uutta kutsua ja latausta, vaan erilaisia toimintoja, kuten lomakkeiden tarkistusta tai navigaation selaamista, voidaan tehdä jo asiakaskoneella olevilla tiedoilla. ASP- ja PHP-toteutuksessa jokainen navigaatiopainikkeen painallus ja lomaketarkastus vaatii, että koko sivu ladataan uudestaan palvelimelta.

ASP:lla PHP:lla tehdyt sivut ovatkin palvelimelle raskaampia toteutettavia. Niiden etuna on kuitenkin parempi turvallisuus. Käyttäjällä ei ole mahdollisuutta nähdä ajettavaa koodia, tai vaikuttaa siihen. Toisin on JavaScriptissä, jota osaava käyttäjä voi halutessaan tarkkailla ja jopa muokata ennen sen ajoa. HTML-koodia toteuttavina kielinä näitä tekniikoita käyttäessä voi törmätä

ongelmiin eri selainten välillä. Selaimilla on taipumusta toteuttaa HTML-koodin visuaalisesta muotoilusta vastaavia CSS-määrittelyjä hieman eri tavalla, mikä saattaa aiheuttaa ongelmia objektien sijainneissa ja muissa ulkoasuun vaikuttavissa seikoissa.

Koska Flash, Silverlight ja Java tarvitsevat kaikki erillisen liitännäisen niillä toteutettujen sivujen suorittamiseen, täytyy websivujen toteutustapaa suunniteltaessa ottaa huomioon näiden liitännäisten käyttöasteet, sillä näillä tekniikoilla tehtyä sivun osaa ei esitetä lainkaan käyttäjälle, jolta puuttuu oikea liitännäinen. Syyskuussa 2011 Flashin käyttöaste oli yli 95%, Javan jopa 77%, ja Silverlightin 64% (Statowl 2011). Näiden tekniikoiden etuna on toisaalta kehityksen helppous ja alustariippumattomuus. Erilaisia siirtymien ja efektien teko on jokseenkin helpompaa kuin pelkkää HTML-kieltä käyttäen, ja rakennettujen sivujen pitäisi näkyä pitkälti samalla tavalla riippumatta käyttöjärjestelmästä tai selaimesta.

Ajastuksen kannalta liitännäisen vaativat webtekniikat ovat kuitenkin huomattavasti hankalampia käsiteltäviä. Siinä missä palvelinpuolella suoritettavia tekniikoita on kohtuullisen helppo hallita ja ajastaa, ovat asiakaskoneen selainliitännäisen suorittamat sivut huomattavasti vaikeampia seurata ja automatisoida. Tämä johtuu siitä yksinkertaisesta syystä, että palvelin ei voi nähdä asiakaskoneella tehtäviä toimintoja, ellei niille ole erikseen ohjelmoitu toimintoja, jotka lähettävät käyttödataa takaisin palvelimelle. Palvelinpuolen tekniikoissa jokainen toiminto tehdään palvelimen kautta, mikä tekee seurannasta helppoa.

2.2 Webpalvelinohjelmat ja ohjelmointikielet

Ajastuksen toteuttaminen vaatii toteutustavasta riippuen erilaisia palvelinohjelmia. Näiden ohjelmien valinnalla on paljon vaikutusta varsinaiseen toteutustapaan. Rakennettua ratkaisua rajoittaa, ja toisaalta määrittää, sen tekemiseen käytetyt palvelinohjelmat. Koko palvelinympäristö on aina rakennettu jonkin käyttöjärjestelmän päälle, joka onkin useimmiten merkittävin yksittäinen huomioitava seikka. Suosittuja palvelinohjelmistoja ovat muun muassa *FreeBSD, OpenBSD, Solaris, Mac OS X Server* (ServerSchool 2011),

sekä *Windows Server*. Palvelimen käyttöjärjestelmä rajaa jonkin verran valittavissa olevia ohjelmistoja, joskin moni ohjelma toimii usealla alustalla.

Webpalvelimen ohjelmiston kokoonpano rajaa valittavissa olevia tekniikoita, sillä kaikki ohjelmistot eivät toimi kaikilla käyttöjärjestelmillä. Erilaisia webpalvelinkokoonpanoja on runsaasti, sillä erilaisia osia voi yhdistellä käyttötarpeen ja mieltymysten mukaan. Oleellisempana osana webpalvelimella on itse webpalvelinohjelma, kuten *Apache httpd, Internet Information Services (IIS)* ja *lighttpd*. Webpalvelinohjelman funktio on yksinkertaisesti suorittaa Internet-sivuja asiakkaan kutsusta. Pelkkä webpalvelinohjelma riittää websivuihin, joilla käytetään vain asiakaspuolen tekniikoita. IIS on vain Microsoft Windows-palvelimilla toimiva webpalvelinohjelma, kun taas Apache httpd ja lighttpd toimivat sekä Windows-palvelimilla, että useilla UNIX-pohjaisilla palvelimilla.

Monipuolisempaan sivustoon tarvitaan myös tietokantapalvelinohjelma, kuten *MySQL, MSSQL* tai *PostreSQL*. Tietokantapalvelinta käytetään datamassojen tallentamiseen ja järjestämiseen. Tietokantaa voidaan käyttää esimerkiksi asiakasrekisterin luontiin tai sivujen sisältämän määrämuotoisen datan säilyttämiseen. Dynaamisuutta sivustoille voidaan luoda palvelinpuolen tekniikoilla, kuten *ASP, PHP, Perl*, *Python* ja *Ruby.* Näillä tekniikoilla voidaan tehdä turvallisesti toimintoja, joiden rakentaminen asiakaspuolen tekniikoilla saattaisi vaarantaa palvelimen tietoturvan.

Mikäli palvelinta aiotaan käyttää sähköpostin lähettämiseen, tarvitaan myös *Message Transfer Agent* (MTA), eli sähköpostin välittäjäohjelma. MTA on yleensä vastuussa sekä sähköpostin lähettämisestä, että sen vastaanottamisesta. Eräitä välittäjäohjelmia on *Exim, Postfix, Sendmail* ja *qmail*. Välittäjäohjelmien merkittäviä eroja on tietysti tuetuissa käyttöjärjestelmissä, mutta myös ylläpidon helppoudessa ja turvallisuudessa. Sendmailia on moitittu sen tietotototurvaongelmista, kun taas qmail on laajuutensa takia hankala ylläpidettävä (Shearer 2007; qmail 2011).

2.3 PHP

PHP: Hypertext Preprocessor on laajalti käytössä oleva, erityisesti webkehitykseen sopiva skriptauskieli, joka voidaan upottaa HTML-koodin sisään (php.net 2011a), jonka rakenteessa on vaikutteita C-, Java- ja Perl-ohjelmointikielistä. Kielen pääasiallisena tarkoituksena on mahdollistaa dynaamisesti luotavia websivuja nopeasti (php.netb 2011), ja kieli onkin tällä hetkellä yksi suosituimmista webkehitykseen käytettävistä palvelinpuolen tekniikoista (Langpop 2011; php.net 2011c).

PHP on ohjelmointikielenä lähtökohtaisesti proseduraalinen, mutta tukee versiosta PHP 5 lähtien moottorina on Zend Engine 2.0, joka tukee myös oliopohjaista ohjelmointitapaa (php.net 2011d). Mahdollisuus valita proseduraalisen ja oliopohjaisen ohjelmointitavan välillä voidaan tulkita sekä kielen vahvuudeksi, että heikkoudeksi. Proseduraalinen ohjelmointitapa toisaalta helpottaa ja nopeuttaa sovellusten tekoa, mutta samalla altistaa rakenteellisille ongelmille ja vaikeuttaa laajojen projektien hallintaa.

PHP mahdollistaa Matt Zandstraa (2004, 3) lainaten, ideoiden kokeilemisen liian helposti, ja imartelee tekijää antamalla hyvää palautetta. (Zandstra 2004, 3). Tällä Zandstra tarkoittaa, että PHP:lla on turhan helppo kirjoittaa proseduraalista koodia ilman minkäänlaista suunnitelmaa koko sovelluksen rakenteesta. Tällainen lähestymistapa yleensä kasaa paljon ongelmia sovelluksen kehityksen tulevaisuudelle, ja monesti pakottaa kirjoittamaan samaa koodia uudelleen.

Oliopohjainen ohjelmointitapa vastaavasti vaatii ainakin jonkin verran suunnittelua ohjelman rakenteesta, sekä järjestelmällistä toteutustapaa. Siinä missä oliopohjaisuus hidastaa hieman ohjelman rakentamisen aloittamista, vähentää se projektin tulevaisuudessa ilmeneviä ongelmia. Hyvin suunnitellussa oliopohjaisessa ohjelmassa samoja osia voidaan käyttää uudelleen, ilman koodin kopiointia tai uudelleenkirjoitusta. Lisäksi oliopohjainen koodaustapa houkuttelee rakentamaan ohjelman loogisesti, mikä vähentää koodia muokattaessa tulevia loogisia virheitä ja auttaa jäsentämään koodin selkeälukuisemmaksi ja helpommin ymmärrettäväksi.

2.4 Ajastetut toiminnot

Vaikka palvelinympäristössä ajastetut toiminnot ovatkin arkipäivää, ei webympäristössä ajastuksille ole ollut vielä pitkään käyttöä. Staattiset sivut ilman interaktiota eivät ole sellaisia tarvinneet, vaan tarvittavat päivitykset ovat liittyneet lähinnä sisällön päivitykseen. Webin kautta tarjottavien palvelujen lisääntyessä tarve automatisoida sivustojen toiminnan osia on tullut ajankohtaisemmaksi. Webpalvelimen tarjoamat toiminnot voivat olla hyvinkin laajoja, ja vaatia paljon ylläpitoa ja säännöllistä päivittämistä. Automatisoimalla toimintoja voidaan samalla vähentää ylläpitäjän työkuormaa, että pienentää riskiä inhimillisen erehdyksen aiheuttamiin virheisiin ja varmistaa suoritettujen toimintojen säännöllisyys.

Ajastamalla ja automatisoimalla sivuston toimintoja voidaan käyttäjille tarjota ajankohtaisempia ja siten hyödyllisempiä palveluita. Siinä missä staattisesti toteutetun sivun tiedot täytyy päivittää käsityönä, voidaan ajastettujen toimintojen ja dynaamisten webtekniikoiden avulla luoda itsestään päivittyvää sisältöä. Tällaisilla ajastuksilla voidaan asiakaskokemusten parantamisen lisäksi tehostaa sivuston seurantaa ja analysointia, jotka ovat ensisijaisen tärkeitä työkaluja asiakastyytyväisyyteen panostavan palveluntarjoajalla. Internetin kehitys mahdollistaa entistä enemmän sähköisesti toteutettavia palveluita, joiden toimivuus on riippuvainen tehokkaasti toteutetusta seurannasta.

Yksinkertaisin, mutta ei välttämättä toimivin tai edes helpoin tapa toteuttaa ajastuksia, on muuhun projektiin käytetyllä ohjelmointikielellä itsellään. Ajastuksen voi toteuttaa kielen itsensä ajastusfunktioilla, mikäli sellaisia on, tai kirjoittamalla itse koodin ajastuksen laskennalle. Tämän lisäksi on olemassa erillisiä ohjelmia, jotka ovat tarkoitettu yksinomaan komentojen ajastamiselle.

3 AJASTUSTEKNIIKAN VALINTA

3.1 Ajastustekniikan käyttökohteet

Webkehitykseen käytettyjen tekniikoiden ongelmana automaation kannalta on niiden suoritustapa: ohjelmakoodi ajetaan vain asiakkaan kutsusta. Opinnäytetyön tavoitteena on selvittää millaisia tapoja on toteuttaa PHP-kielellä tehdyille skripteille luotettavaa ajastusta ja sen avulla edelleen automaatiota. Yleisesti käytettyjä ajastustapoja tarkastellaan niiden vahvuuksien ja heikkouksien, kuten luotettavuuden, toiminnallisten rajoitusten, palvelinvaatimusten, toiminnallisten vaatimusten ja käyttöönoton vaikeuden, kannalta. Parhaaksi todettua tekniikkaa käytetään Futurality-oppimisympäristön ajastettujen toimintojen toteuttamiseen. Ajastamalla toistuvia toimintoja on tarkoitus vähentää ylläpitäjän tekemän monotonisen käsityön tarvetta.

Futurality-virtuaaliympäristössä on useita toistuvia ylläpito- ja päivitystoimintoja, joita ei ole kannattavaa ruveta tekemään käsin. Oppimisympäristöön on tarkoitus toteuttaa kuluttajamarkkinoiden simulointi, eli harjoitusyrityksille tulevaa ostoliikennettä. Ostoliikennettä on mahdollista tehdä käsityönäkin, mutta kymmenien tai satojen jälleenmyyjien simulointi vaatii kohtuuttoman määrän itseään toistavaa työtä. Tällaiset tehtävät on monesti mahdollista ulkoistaa ainakin osittain tietokoneen tehtäväksi.

Automatisoimalla kuluttajamarkkinat, päästään sekä ylläpidon, että koulutuksen kannalta sellaiseen ideaalitilanteeseen, jossa ympäristön tekninen toiminta ei vaadi juurikaan käsityötä, jolloin työaikaa vapautuu muihin tehtäviin. Ympäristöä käyttävät opiskelijat pystyvät perustamaan yrityksen, jonka liiketoiminta voidaan aloittaa täysimittaisesti heti, sen sijaan että yrityksen täytyisi ensin ruveta hankkimaan asiakkaita.

Tärkeä osa yritystoiminnan opiskelua on toiminnanohjausjärjestelmän eri osien ymmärtäminen, sekä niiden käytön hallitseminen. Yleisiä toiminnanohjausjärjestelmän osia ovat ainakin kirjanpito, palkanlaskenta ja osto- ja myyntireskontra. Lisäksi toiminnanohjausjärjestelmästä saattaa löytyä työkalut myös varastonhallintaan ja tuotannonohjaukseen. Näiden eri osien ymmärtämiseen saattaa riittää teoriaopinnot, mutta sujuvan käytön oppiminen

vaatii käytännön työtä: kokeilua, epäonnistumista ja onnistumista. Kokeilu vaatii kuitenkin yritystoiminnan normaaleja tapahtumia, kuten ostoja ja myyntiä. Koska harjoitusyritykset eivät myy reaalimaailman palveluita tai hyödykkeitä, ei reaalimaailmassa itsestäänselvyytenä olevia kuluttajamarkkinoitakaan ole virtuaaliympäristössä olemassa. Kuluttajamarkkinat, eli jälleenmyyjät ja tuotteiden loppukäyttäjät, täytyy harjoitusyritysympäristöön rakentaa keinotekoisesti.

Saumattomasti toimivan automatisoinnin tarkoitus on mahdollistaa pelin kaltaisen, mutta vahvasti todellisen maailman toimintoja vastaavan ympäristön helpon hallinnan ja jatkuvan päivittymisen. Tämä edelleen antaa mahdollisuuden kokeilla yrityksen perustamista ja sen pyörittämistä mahdollisimman realistisesti.

Simuloimalla liiketoimintaympäristö poistetaan oppimista haittaava odottelu, arvaamattomuus ja hallitsemattomuus. Puutteet voidaan korvata tarkkaan suunnitelluilla ja hallituilla mekanismeilla, kuten helposti ylläpidettävillä ja seurattavilla kuluttajamarkkinoilla. Futuralityn sisällä olevaan virtuaaliyritykseen tulevia tilauksia voidaan hallita hyvin tarkasti, ja yrittäjien, eli opiskelijoiden, työpanos voidaan ottaa huomioon yrityksen menestyksessä.

3.2 Ajastustekniikoiden vertailun kriteerit

Ajastuksen toteutukseen parhaiten sopiva tekniikka riippuu sekä toteutuksen vaatimuksista, että palvelimen käyttöjärjestelmästä ja tarjolla olevista ohjelmista. Ajastustekniikoiden sopivuutta voidaan vertailla kriteereillä, kuten varmuus, vikasietoisuus, säännöllisyys ja ylläpidon tarve. Lisäksi täytyy huomioida ajastuksen vaatimukset ja rajoitukset. Ajastuksen varmuudella tarkoitan ensisijaisesti sitä, kuinka todennäköistä on, että toteutettava komento jää syystä tai toisesta suorittamatta, mutta myös sitä, kuinka tarkasti komento on sidottavissa kellonaikaan.

Vaikka komennon varsinainen toteutuminen on usein vaatimus, voi tarkkuuden vaatimus vaihdella. Tekniikasta riippuen kelloon sidottu tarkkuus voi vaihdella sekunnin osista jopa useisiin päiviin. Vikasietoisuus on tärkeä osa

toimintavarmuutta, mutta olen erottanut sen omaksi osakseen tarkempaa vertailua varten. Siinä missä toimintavarmuudella tarkoitan enemmänkin tekniikan itsensä luotettavuutta, vikasietoisuudella haluan ottaa mahdolliset palvelinongelmat ja inhimilliset virheet tarkemmin esille. Vikasietoisuutta käytän tarkasteltaessa erilaisten ongelmatilanteiden vaikutusta tekniikan varmuuteen.

Säännöllisyys on myös toimintavarmuudesta irrotettu osa, jolla tarkastelen kunkin tekniikan toimintaa suhteessa ajastettujen toimintojen toivottuun toteutusaikaan. Ylläpidon tarpeella tarkoitan käsityön määrää, jota ajastetuista toiminnoista vastaava joutuu tekemään. Ylläpidon määrää on vaikea mitata absoluuttisesti, sillä kriteereinä voi olla joko tarvittavien ylläpitotoimien määrä, niihin kulunut aika, tai tehtävien monimutkaisuus ja vaadittu panostus.

Vertailtavuuden vuoksi voidaan ajatella, että täysin manuaalisena toteutettu ympäristö vaatii ajastuksen kannalta 100% ylläpitoa, ja täysin ajastettu ympäristö vaatii 0% ylläpitoa. Käytännössähän tarkastelu ei ole ihan näin suoraviivainen, sillä kokonaan ajastettuakin ympäristöä on hyvä valvoa, puhumattakaan toiminnan arvioinnista ja sen myötä toiminnan tehostamisesta. Tekniikoiden vaatimuksilla ja rajoituksilla tarkoitan vaadittuja ohjelmia ja toiminnallisia rajoituksia. Vaadituilla ohjelmilla otetaan kantaa sekä toteutusympäristöön, että toteutusta tekevän henkilön mahdollisesti rajallisiin käyttöoikeuksiin. Rajoituksista tutkitaan ajastusten minimi-intervallia, eli pienintä mahdollista aikaa joka kahden komennon suorituksen välillä voi olla.

Kriteereiden erottelu on jokseenkin pakotettua, sillä aiemmin mainitut varmuus, vikasietoisuus, säännöllisyys ja ylläpidon tarve ovat osittain päällekkäisiä. Varmuus, vikasietoisuus ja säännöllisyys ovat kaikki vain osia tekniikan luotettavuudesta, joka taas vaikuttaa merkittävästi ylläpidon tarpeeseen. Erottelu on tehty helpottamaan vertailua. Painottamalla näitä kriteereitä eri tavoilla voidaan tehdä vertailua ajastusten paremmuudesta tilanteesta riippuen.

Käytettävässä testiympäristössä vaaditut ohjelmat eivät ole rajoittava tekijä, sillä kaikkien tarvittavien ohjelmien ja oikeuksien asetus on mahdollista. Ylläpidon tarve on tärkeä kriteeri, sillä yhtenä osa toteutuksen tarkoitusta on juuri vaaditun käsityön vähentäminen. Varmuus ja säännöllisyys ovat kuitenkin ensisijaisina tutkittavina, sillä komentojen toteutusjärjestyksellä ja toteutumisen varmuudella on merkittävä osuus koko ympäristön toiminnan kannalta. Vikasietoisuus on

erittäin oleellista, mutta ei tässä toteutuksessa aivan ensimmäinen prioriteetti, sillä ympäristöä, johon toteutus tehdään, valvotaan jatkuvasti ja toiminta tarkastetaan säännöllisesti.

4 AJASTUKSEN RATKAISUVAIHTOEHDOT

4.1 Asiakaskoneiden laukaisemat ajot

Kaikkein yksinkertaisin tapa ajaa skriptejä lienee PHP:n normaali käyttötapa: asiakkaiden laukaisemat ajot. Näillä tarkoitan sellaista tilannetta, jossa käyttäjä (asiakas) navigoi itsensä palvelua tarjoavalle palvelimelle ja pyytää haluttua php-tiedostoa. Tämä tapahtuu yleisimmin selaimen osoitekenttään kirjoitetulla URL-osoitteella. Ajettujen tiedostojen määrä ei ole rajoitettu, vaan yhdellä kutsulla voidaan ajaa kaikki tarvittavat tiedostot. Jos asiakas kutsuu tiedostoa index.php, voidaan tiedostosta kutsua käyttäjän tietämättä muita tiedostoja ajastuksen tarpeisiin.

Jos sivuston haluttaisiin keräävän statistiikkaa eri navigaatioelementtien painalluksista ja lähettävän ne ylläpitäjän sähköpostiin, täytyisi sivustolle luoda siitä huolehtiva automaatio. Jokaisesta kutsusta laukeava ilmoitus aiheuttaisi järjettömän määrän hyödytöntä postia ylläpitäjälle, joten ilmoituksia täytyy ensin koota. Parempi tapa onkin lähettää ilmoitus aina kun painallusten määrä ylittää tietyn raja-arvon, mutta tällä lähestymistavalla raporttien lähetys on kaikkea muuta kuin säännöllistä.

Säännöllisesti ajastettu raportti voitaisiin tehdä tarkastamalla sivun latauksen yhteydessä onko raportin lähetykselle määritetty hetki ohitettu, ja lähettää raportti vain jos ehto toteutuu. PHP:lla tällaisen rakenteen toteuttaminen on suhteellisen helppoa. Ensin täytyy rakentaa toiminnallisuus, joka tallentaa jokaisen painalluksen. Painallukset voi tallentaa esimerkiksi XML-muotoiseen tekstitiedostoon tai tietokantaan. Tämän jälkeen tehdään skripti, joka tarkastaa onko painallukset ylitetty, ja tarvittaessa toimii halutulla tavalla. Yksinkertainen esimerkki tällaisesta rakenteesta:

```
<?php
$painallukset = haePainallukset();
if($painallukset >  1000){
        lahetaRaportti();
} else {
        lisaaPainallus();
}
?>
```

KOODIESIMERKKI 1. Painalluksiin sidonnainen automaatio PHP:lla

Näin toteutetussa ajastuksessa on paljon ongelmia. Sekä luotettavuus, säännöllisyys, että suorituskyky ovat huonoja. Luotettavuus kärsii, sillä skriptien ajo riippuu täysin käyttäjistä; jos yksikään käyttäjä ei kutsu skriptin sisältävää tiedostoa, ei skriptiä ajeta. Samasta syystä myös säännöllisyys kärsii. Ajatellaan että ylläpitäjä haluaisi joka maanantai klo 8:00 automaattisesti raportin sivuston käytöstä edellisellä viikolla. Raportin lähetys täytyisi ajastaa siten, että se lähetetään kun tämä kellonaika on ylitetty. Aikaan sidottu raportointi on toteutettavissa lähes yhtä helposti, kuin painallusten määrään sidottu raportointi. Vastaavasti täytyy tehdä toiminnallisuus, joka pitää kirjaa siitä, koska viimeisin raportti on lähetetty. Maanantaina klo 8 jälkeen lähetettävä raportti voitaisiin toteuttaa esimerkiksi:

```
<?php
$lahetetty = haeViimeisinLahetettyPVM();
if( $lahetetty != date(’Y-m-d’) and
date(’w’) == ’1’ and date(’H’) > 8 ){
        lahetaRaportti();
        paivitaViimeisinLahetettyPVM ();
}
?>
```

KOODIESIMERKKI 2. Aikaan sidonnainen automaatio PHP:lla

Jos sivun ensimmäinen lataus tuleekin kuitenkin vasta 10:30, joutuu ylläpitäjä odottamaan raporttiaan. Vilkkaasti liikennöidyllä sivulla tällaista ongelmaa ei ole, varsinkaan jos skriptin ajo suoritetaan aina usein ladattavaa tiedostoa, kuten juuri index.php:tä, ajaessa. Usein suoritettavassa tiedostossa ladattaessa törmätään uuteen ongelmaan: resurssien tuhlaaminen. Skripti, joka lähettää joka maanantai yhden raportin, halutaan käytännössä ajaa vain kerran viikossa, mutta aiemmin mainitussa toteutuksessa skriptin ajo tarkistetaan joka ikisellä latauskerralla.

Jos esimerkkiskriptin sisältävää sivua ladataan 10 000 kertaa maanantaina kello 8 jälkeen, tehdään tarkistus 99,99% kerroista turhaan. Asiakaskoneiden laukaisemana tehtävät skriptien ajot ovat toimiva toteutus jos automatisointi halutaan tehdä nopeasti, helposti ja ilman lisäohjelmien tarvetta, eivätkä varmuus ja säännöllisyyden tarkkuus ole kriittisiä. Suoraan ohjelmointikielellä itsellään toteutettu ajastus on myös helppo siirtää ympäristöstä toiseen ilman vaaraa yhteensopivuusongelmista. Oletuksena tietysti, että seuraavassa ympäristössä on vähintään yhtä uusi versio PHP-tulkista, kuin edellisessä.

4.2 Cron

Cron on *daemon*-tyyppinen UNIX-ohjelma, joka ajetaan sen päällä ollessa joka minuutti. Aina käynnistyessään Cron hakee sille ennalta määritetyistä kansioista (oletuksena */var/spool/cron/, /etc/cron.d/*) suoritettavia tiedostoja. (Cron man 2011) Näissä kansioissa on edelleen määritelty *crontab*-tiedostoja, joissa taas on erilaisia komentoja komentotulkin suoritettavaksi, sekä näiden komentojen suoritusajankohdat tai suoritusvälit. (crontab man 2011)

Komentoja voidaan määritellä suoritettavaksi minuutin, tunnin, päivän, kuukauden ja viikonpäivän perusteella. Suoritus voi olla siis useimmillaan kerran minuutissa, eli joka kerta kun Cron käynnistyy, ja harvimmillaan kerran vuodessa, eli tietyn kuukauden tiettynä päivänä. Eri komennoilla voi olla eri suoritusvälit, joten esimerkiksi yksi skripti voidaan ajaa joka tunti, toinen viikoittain tai kuukausittain. Cron on helppoutensa, toimintavarmuutensa ja tehokkuutensa ansiosta varsinkin UNIX-ympäristössä suosittu tapa toteuttaa ajastettuja toimintoja.

Suoritettavaksi määritetyt komennot suoritetaan luotettavasti juuri silloin kun ne on määrätty tehtäväksi. Windows-ympäristössä vastaava toiminto on *”Scheduled Tasks”*. Se toimii teknisesti eri tavalla kuin Cron, mutta molemmilla voidaan käytännössä luoda samankaltaisesti kellontarkasti ajastettuja toimintoja. PHP-skriptit voidaan ajaa helposti käyttämällä PHP-tulkkia itseään, mutta tämän käyttötavan ehtona on, että Cron pääsee käsiksi tarvittaviin tiedostoihin, ja että sillä on niihin suoritusoikeudet. Toinen tapa suorittaa php-tiedostoja on käyttää wget-ohjelmaa, joka hakee tiedoston HTTP-protokollan kautta samalla tavoin kuin asiakaskone tekisi.

Näin aiemmin mainittu statistiikkaraportin viikkolähetys voitaisiin ajastaa lähtemään varmasti joka maanantai tasan kello 8:00 riippumatta käyttäjien toimista ja sivujen latauksista. Toteutus on hiukan pelkällä PHP:lla tehtyä helpompi, sillä suoritusaikaa ei tarvitsisi erikseen tarkastaa. Cronilla maanantaina klo 8 ajettavan komennon voisi toteuttaa tarkastettavaan crontabiin kirjoitettavalla komennolla

```
00 8 * * 1 apache php /var/www/html/lahetaRaportti.php
lahetaRaportti
```

KOODIESIMERKKI 3. Raportin ajastus Cronilla

Komento vaatii toki lahetaRaportti.php-tiedoston, jossa varsinainen raportin koonti ja lähetys on toteutettu. Cron huolehtii tässä vain ajastuksesta. Vaikka Cron onkin periaatteessa erillinen ohjelma, on se useimmiten helppo ja tehokas tapa toteuttaa ajastuksia, sillä se on oletuksena asennettuna yleisimmissä UNIX-käyttöjärjestelmissä, eikä uusien crontabien lisäys suoritettaviin ajastustiedostoihin juurikaan lisää resurssien käyttöä.

Cronia käytettäessä tulee kuitenkin ottaa huomioon, että Cron olettaa sitä ajavan koneen olevan aina päällä. Jos ajastettu toiminto on määrätty maanantaille klo 8:00, eikä palvelin ole juuri tuolla hetkellä päällä, jää toiminto kokonaan suorittamatta. Cron on loistava tapa lähes mihin tahansa ajastuksen tarpeisiin, kunhan Cronin toimintalogiikka on hallussa, ja Cron on asennettuna tai mahdollista asentaa käytettävään palvelinkoneeseen.

4.3 Anacron

Anacron on Cronin tapainen ohjelma komentojen ajastamiseen ja automatisointiin. Cronin tavoin Anacron suoritetaan joka minuutti, jolloin Anacron-prosessi hakee ennalta määritetyistä kansiosta (oletuksena */etc/anacrontab*) *anacrontab*-tiedostoja, joissa voi olla suoritettavia komentoja ja niiden suoritusajankohtia. (Anacron man 2011) Anacronin toiminta on näiltä osin täysin identtinen Cronin kanssa. Merkittävin ero Anacronilla ja Cronilla on komentojen suorituslogiikassa. Siinä missä Cron tarkastaa vain onko kuluvalle minuutille määritetty suoritettavia komentoja, tarkastaa Anacron aina kaikki sille määritetyt tehtävät.

Jos Anacron huomaa että esimerkiksi edelliselle päivälle oli merkattu ajettavaksi komento, mutta komennon suoritukselle tallennettu *timestamp*-lokimerkintä ei täsmää suunniteltuun suoritusajankohtaan, komento suoritetaan ja sen

timestamp päivitetään. Anacron ei siis oleta, että palvelin olisi aina päällä, vaan osaa suorittaa palvelimen suljettuna ollessa lauenneet tehtävät takautuvasti, toisin kuin Cron. Anacronia käytettäessä on tiedostettava, että komentoja saatetaan suorittaa eri aikaan kun niitä on alunperin suunniteltu suoritettavaksi. Se ei siis sovellu käytettäväksi ajastuksiin, jotka täytyy suorittaa täsmällisinä aikoina. Toinen rajoite Anacronilla on ajastusten minimiaika.

Siinä missä Cron pystyy suorittamaan komennon vaikka joka käynnistymiskerrallaan, eli joka minuutti, voi Anacron suorittaa saman komennon korkeintaan kerran päivässä. Tämä johtuu aiemmin mainituista timestampeista, jotka tehdään vain päivän tarkkuudella. Kolmas Anacronin rajoite on sen suorituskyky: koska Anacron tarkastaa aina kaikkien sille määritettyjen komentojen suorituksen tilan, on se jonkin verran Cronia raskaampi ajettava jos tarkastettavia komentoja on paljon. Anacron on Cronin rinnalla tehokas valinta tarkkojen ajastusten toteutukselle silloinkin, kun palvelimen jatkuva päällä olo ei ole varmaa. Tekninen toteutus on täysin Cronin toteutusta vastaava, joskin toteutettava rivi täytyy kirjoittaa crontab-tiedoston sijaan asianmukaiseen anacrontab-tiedostoon.

4.4 PHP:n ajastusfunktiot

PHP:ssa itsessään on olemassa funktioita viiveen ja ajastuksen tekoon. Funktiolla *sleep()* voidaan viivästyttää skriptin tai sen osien ajoa. Lisäksi funktiolla *time_sleep_until()* voidaan ajastaa skriptin ajo. Näitä funktioita käyttämällä voitaisiin luoda yksi skriptitiedosto, joka täytyisi ajaa manuaalisesti esimerkiksi kerran viikossa. Ratkaisu ei ole kuitenkaan varsinainen automatisointi, sillä skripti täytyy joka tapauksessa laukaista määrätyin väliajoin käsin.

Asiakaskoneiden ajettavaksi näin tehtyä ajastusta ei voida antaa, sillä ajastuskriptin ajava sivu näytetään käyttäjälle vasta koko sivun latauduttua, eli vasta kun ajastettu prosessi laukeaa. Lisäksi toteutus vaatii yhden tai useamman pysäytetyn PHP-prosessin suoritettavana taustalla. Viikoittain ajettavan skriptin täytyy pahimmillaan olla seitsemän päivää sleep-tilassa, siinä missä *Cron*- ja *Anacron*-toteutuksissa skriptejä ei tarvitse ikinä jättää taustalle.

Taustalla odottava prosessi toimii kunhan palvelinta ei tarvitse käynnistää uudelleen skriptin odottaessa, eikä skriptin ajoa lopeteta muusta syystä. Toteutus on muuten vastaava kuin suoraan asiakaskoneiden laukaisemana tehdyt ajot, mutta ajastuksen toteutusta ei tarvitse tehdä käsin. Maanantaisin klo 8 laukeava toiminto voitaisiin tehdä esimerkiksi näin:

```
<?php
$suoritusaika = haeSeuraavanMaanantainTimestamp();
time_sleep_until( $suoritusaika );
lahetaRaportti();
?>
```

KOODIESIMERKKI 4. Raportin ajastus PHP:n ajastusfunktiolla

Tällaista toteutusta käytettäessä pitää kuitenkin toteuttaa käsin halutun ajankohdan timestampin haku, jotta PHP:n ajastusfunktio tietää kauanko sen on odotettava. PHP:n omia ajastusfunktioita käytettäessä täytyy huolehtia, ettei palvelimelle ole määritetty maksimiaikaa PHP-skriptien ajolle. Skriptien ajon maksimiajan poisto tosin altistaa palvelun tilanteille, jossa koodin logiikkavirhe aiheuttaa skriptin loputtoman toteuttamisen. Lisäksi täytyy huomioida, että skriptin ajaminen lukuisia kertoja voi saada palvelimen ylikuormitettua.

Tähän toteutukseen olisikin hyvä yhdistää asiakaskoneiden laukaisemissa ajoissa käytetty tarkastus komennon edellisestä ajokerrasta. PHP:n omia ajastusfunktioita ei voida hoitaa asiakaskoneen laukaisemina, sillä PHP:n säikeettömyys aiheuttaa sen, että sleep-komentoa odoteltaessa selain näyttää lataavan sivua. Lisäksi skriptin ajo keskeytetään, jos asiakas lopettaa sivun latauksen tai sulkee selaimen. Laukaisu täytyy siis tehdä sellaisella koneella, jolla selain voidaan jättää pysyvästi päälle.

Skriptiä kutsuvana koneena voi toimia palvelin itsekin, joskaan silloin ei teknisesti ottaen voidakaan puhua asiakkaasta. Selaimen, tai selainten, auki jättäminen syö kuitenkin turhaan palvelimen tehoja, eikä ole teknisesti kovin tyylikäs ratkaisu. Toteutustavan hyvänä puolena on sama kuin asiakaskoneiden

laukaisemilla ajoilla: erillisiä palvelinohjelmistoja PHP:n lisäksi ei tarvita. PHP:n ajastusfunktiot toimivat ajastuksen toteuttamiseen jos päällekkäisiä ajastuksia ei ole paljoa, ja toteutuksessa otetaan huomioon PHP:n nukkuvien prosessien vaatimukset.

Kovin laajamittaisiin ajastustarpeisiin en voi PHP:n omia ajastuksia suositella, eikä niitä ole selvästi sellaiseen suunniteltukaan. Jos PHP tukisi säikeitä, olisi sleep-funktioiden käyttö helpompaa, kun asiakkaalle voitaisiin näyttää sivun osia samalla, kun sleep-funktion laukeamista odotellaan. Säikeetkään eivät tosin mahdollistaisi sleep-funktioilla toteutettujen ajastusten laukaisun jättämistä asiakaskutsujen varaan, sillä säikeiden ajo loppuisi kuitenkin siinä vaiheessa jos asiakas poistuu ladattavalta sivulta.

4.5 Muut ajastustavat

Ajastuksessa voi hyödyntää myös JavaScriptiä tai HTML:n metakutsuja. Molemmilla voidaan toteuttaa sivu, joka lataa itsensä uudelleen säännöllisesti niin kauan kun koodin sisältävää tiedostoa kutsutaan. Käytännössä tämä toimisi niin, että ylläpitäjä avaa skriptiä ajavan tiedoston selaimeen, minkä jälkeen skripti lataa sivun uudelleen säännöllisin väliajoin, kuten viiden minuutin välein. Joka ajolla voidaan tarkastaa muiden skriptien toteutusehtoja.

Tämä tapa on sekoitus Cronin toimintatapaa tarkastaa ajettavat toiminnot säännöllisin väliajoin, ja asiakaskoneen laukaisemia ajoja. Ratkaisussa ajastus ulkoistetaan asiakaskoneen selaimelle, joten toimintavarmuus vaatii jatkuvan asiakasyhteyden. Asiakas voidaan toki simuloida palvelimelle itselleenkin, mikäli sillä on mahdollista pitää tarvittavia tekniikoita hyödyntävää selainta auki.

Yksi tapa on käyttää ulkoistettua ajastettua ajoa, kuten Webcron (Webcron 2011) ja Mywebcron (Mywebcron 2011). Webcronin ja Mywebcronin kaltaiset palvelut ovat palveluita, jotka tekevät asiakaskutsuja ajastetusti. Toteutus vastaa täysin asiakaskoneen laukaisemia ajoja sillä erotuksella, että skriptien ajo on säännöllistä. Näin ajettavan skriptin ei tarvitse myöskään olla usein ajettavassa tiedostossa, kuten index.php, mikä vähentää resurssihukkaa. Alla

yhteenveto (taulukko 1) kappaleessa mainittujen ajastustekniikoiden vertailusta, sekä vertailukohtana manuaalinen ajo.

TAULUKKO 1. Ajastustekniikoiden vertailua

PHP-skriptien ajastus					
	Anacron	**Cron**	**Asiakaskoneen laukaisemat ajot**	**PHP:n ajastusfunktiot**	**Manuaalinen ajo**
Säännöllisyys	Hyvä	Hyvä	Tyydyttävä	Hyvä	Huono
Varmuus	Hyvä	Hyvä	Huono	Tyydyttävä	Huono
Vikasietoisuus *	Hyvä	Tyydyttävä	Tyydyttävä	Huono	Huono
Käsityön määrä	Hyvä	Hyvä	Hyvä	Tyydyttävä	Huono
Ajastusten intervalli	Tyydyttävä (Päivä)	Hyvä (Minuutti)	Huono (ei varmuutta)	Hyvä (Sekunti)	Huono (ei varmuutta)
Vaaditut ohjelmat	Tyydyttävä (Anacron)	Tyydyttävä (Cron)	Hyvä (ei lisäohjelmia)	Hyvä (ei lisäohjelmia)	Hyvä (ei lisäohjelmia)
Yleisarvosana **	2,7	2,7	2,0	2,3	1,3
*** Palvelinongelmat, inhimilliset virheet**					
**** Hyvä = 3, Tyydyttävä = 2, Huono = 1**					

Vertailussa (taulukko 1) ei ole otettu huomioon kaikkia käytön rajoitteita, eikä palvelinrasitteen määrää, minkä takia tekniikka ”PHP:n ajastusfunktiot” saa suhteettoman korkean pistemäärän. PHP:n omat ajastusfunktiot toimivat selvästi muita tekniikoita huonommin pitkissä ajastuksissa, sekä tarvittaessa useita ajastuksia. Lisäksi PHP:n omien ajastusfunkioiden toimintalogiikan soveltuvuus ajastuksiin on jokseenkin kyseenalainen sen vaatiessa asiakaskutsun olevan aktiivinen koko odotusprosessin ajan.

5 VALITUT TEKNIIKAT

5.1 Ajastustekniikka

Ajastuksen olisi periaatteessa voinut toteuttaa millä tahansa tarkastelluista ajastustekniikoista, mutta parhaiten tehtävään sopivaksi osoittautui kuitenkin Cron. Skriptien ajo asiakaskoneiden laukaisemana ei olisi ollut riittävän säännöllistä, varmaa eikä optimoitua. Ajastettuja ajoja täytyy tehdä sekä viikoittain, että muutaman minuutin välein. Asiakaskutsujen säännöllisyydestä ei voi koskaan olla varma, jolloin ajoja saattaisi jäädä jopa kokonaan tekemättä. Kerran viikossa tai kerran päivässä ajettaville skripteille varmuuden taso saattaisi olla juuri ja juuri riittävä, mutta useammin kuin kerran päivässä tehtävissä ajastuksissa virhemarginaali alkaa olla niin pieni, että tämä tekniikka ei tule kysymykseen.

PHP:n omien ajastusfunktioiden hylkäämisen keskeisimpänä syynä oli toteutustavan aiheuttama rasite ylläpidolle, mutta osittain myös ajastusfunktioiden huono toimintavarmuus. Ajastuksen käynnistys täytyy tehdä kuitenkin manuaalisesti. Manuaaliajon voisi toki välttää yhdistämällä tekniikoita, ja jättämällä ajastuksen käynnistyksen asiakaskoneiden vastuulle. Asiantunteva ylläpitäjä ei kuitenkaan voi luottaa tällaiseen, joten skriptin käynnistyminen pitäisi kuitenkin tarkastaa erikseen, jolloin säästetty työmäärä joudutaan kuitenkin käyttämään ajastuksen varmistamiseen.

Vaikka ajastuksen käynnistyksen nähtäisiinkin olevan riittävän toimintavarma, on seuraava ongelma nukkuvissa skripteissä. Nukkuvat PHP-skriptit eivät toimi jos palvelinta tarvitsee käynnistää uudelleen ajon ollessa odottamassa. Lähtökohtaisesti palvelimia ei käynnistellä uudelleen muuta kuin suunnitellusti, jolloin ajastuksen käynnistämisen voisi vain ajoittaa palvelimen uudelleenkäynnistyksen jälkeen. Luotettava toiminta vaatisi näin että palvelinta ei käynnistettäisi koskaan muuta kuin suunniteltuna aikana, ja että ajastuksen käynnistys tehdään aina oikeaan aikaan. Vaihtoehto ei ole huonoin mahdollinen, mutta kaksi osittain epävarmaa vaihetta ajastuksessa sai minut hylkäämään tämänkin vaihtoehdon.

Anacron ja Cron ovat hyvin samankaltaisia ohjelmia, molemmat erityisesti tarkoitettu toimintojen ajastukseen. Ajastus on täysin automatisoitua, eikä vaadi ylläpitäjältä muuta kuin palvelimen normaalin toimintatarkastuksen ajoittain – tarkistus joka on tuotantoympäristön palvelimelle joka tapauksessa välttämätön. Cronin ja Anacronin ainoat puutteet liittyvät palvelimen uudelleenkäynnistyksiin. Jos palvelin on sammumassa, sammutettu tai käynnistymässä uudelleen juuri kun Cron- tai Anacron-ajo pitäisi tehdä, jää ajo tekemättä.

Anacronissa tämä ongelma korjautuu seuraavalla Anacronin käynnistymiskierroksella, mutta Cronissa toiminto ajetaan uudelleen vasta kun ajastus seuraavan kerran laukeaa; väliin jäänyttä tai jääneitä toimintoja ei ajeta ikinä. Anacronin tavassa ajaa väliin jääneet toiminnot on myös riski: jos palvelin on pitkään pois päältä, saattaa toiminnon toteuttaminen aiheuttaa yllättäviä ongelmia. Jos Anacronin toimintatapaa ei ole huomioitu ajettavien toimintojen suunnittelussa, voi tämä pahimmillaan aiheuttaa tehtävien toteutumista väärässä järjestyksessä.

Toteutustavaksi valitsin lopulta Cronin pitkälti siitä syystä, että Anacronin ajastusten minimi-intervalli on päivä, vaikka osaa ajastuksista täytyy ajaa useita kertoja tunnissa. Toteutuksessa voitaisiin käyttää Cronia usein laukeavien ajastusten toteutukseen, ja Anacronia päivittäin, viikoittain ja kuukausittain ajettaviin tehtäviin. Selkeyden vuoksi myös harvemmin ajettavat toiminnot hoidetaan Cronilla.

5.2 Tietokantapalvelinohjelma

Tietokannalla on hyvin suuri osa tilausautomaation toiminnassa. Kaikki tilauksen asetuksiin, luontiin, rakentamiseen ja lähettämiseen liittyvä data on kirjoitettu tietokantaan. Tämän lisäksi tehtävät laskut ovat itse asiassa rivejä Pupesoftin natiiveissa tauluissa, joita käyttämällä laskuja voidaan piirtää PDF-muotoon käyttäen Pupesoftista löytyviä työkaluja. Vaikka tilausautomaatin ensimmäisissä suunnitelmissa tietokantavaihtoehdoista löytyi myös PostgreSQL, oli MySQL:n valinta lopulliseksi ratkaisuksi melko selvä. MySQL on yksi käytetyimmistä relaatiotietokantaohjelmistoista, ja myös Pupesoft käyttää sitä. Lisäksi MySQL on Pupesoftin ja Futuralityn tavoin GPL-lisensoitu,

mikä selkeyttää koko järjestelmän käyttöoikeuksia. Valintaa tuki myös oma osaaminen, jota on eniten juuri MySQL:n käytöstä aina asennuksesta konfigurointiin ja käyttöön. PostgreSQL:n puolesta puhui lähinnä MySQL:n epävarma kohtalo sen päädyttyä Oraclen omistukseen (PostgreSQL 2009).

Toteutuksen kannalta pahin puute MySQL:n toiminnassa on se, ettei sarakkeiden oletusarvoksi voi antaa funktiota. MySQL-tietokannassa ei voida esimerkiksi määrittää DATE-tyyppisen ”luotu”-kentän oletusarvoksi ”CURRENT_DATE”, joka antaisi uuden rivin ”luotu”-kentän arvoksi automaattisesti luontihetken. MySQL:n käyttöohjeen (2011) mukaan funktioita voi käyttää ainoastaan TIMESTAMP-tyyppisten kenttien kanssa (Oracle Corporation 2011, 10.1.4). Aiemmin mainittu kenttä ”luotu” voitaisiin siis määrittää TIMESTAMP-tyyppiseksi, ja antaa oletusarvoksi ”CURRENT_TIMESTAMP”. Tämän kentän tietokanta osaisi täyttää luontiajalla automaattisesti uutta riviä tehdessä.

TIMESTAMP-datatyypin käytössä on tosin puute: TIMESTAMP-tyyppisiä kenttiä voi olla vain yksi MySQL-taulua kohden. Jos siis haluttaisiin aiemmin mainittu kenttä ”luotu”, sekä kenttä ”muokattu”, johon tallennettaisiin aina viimeisimmän muokkauksen ajankohta, ei TIMESTAMP-datatyypillä voitaisi toteuttaa kun toinen. Automaattisesti päivittyvä ”muokattu”-kenttä voidaan tehdä TIMESTAMP-tyyppisenä, antamalla sille oletusarvoksi ”ON UPDATE CURRENT_TIMESTAMP”. Koska kahta tai useampaa kenttää ei voida MySQL:n nykyisessä versiossa 5.6 määrittää päivittymään automaattisesti, on tämä puute otettava huomioon uusia rivejä luovassa ohjelmassa.

5.3 Käytettävät ohjelmointikielet

Tilausautomaatin ohjelmallinen puoli on hyvin perustason ohjelmointia, eikä vaadi koodaukseen käytettävältä kieleltä mitään erityisominaisuuksia. Ohjelman olisi voinut toteuttaa yhtä hyvin melkein millä tahansa muullakin kielellä. PHP:n valitsin pitkälti sen takia, että Pupesoft, toiminnanohjausjärjestelmä johon Futurality-virtuaaliympäristö pohjautuu, on toteutettu PHP:llä. Olisi epäloogista vaihtaa ohjelmointikieltä ilman pakottavaa syytä. Mahdollisista kielistä PHP on järkevä ratkaisu myös sen takia, että tilausautomaatin hallinta- ja

raportointipaneeli tehdään selainpohjaiseksi käyttäen PHP-, XHTML, JavaScript (JQuery) ja CSS-kieliä. Yhden ohjelmointikielen käyttö kaikissa Futuralityn osissa mahdollistaa koodin uudelleenkäytön ilman muutoksia. Cronin kannalta ei ole suurta merkitystä, millä kielellä ohjelmat on toteutettu, sillä Cronin avulla pystytään helposti ajamaan ohjelmia eri komentotulkkien kautta.

Varteenotettavimpana vaihtoehtona PHP:lle oli Shell script, joka on suoraan UNIX-komentotulkille kirjoitettavien ohjelmien ohjelmointikieli. Sillä voidaan kirjoittaa nopeasti ja suhteellisen helposti erilaisia työkaluja helpottamaan tehtävien suorittamista. Esimerkiksi usein toistuvia komentoja voidaan hoitaa käsityön sijaan ohjelmallisesti. Jos järjestelmästä tarvitsisi toistuvasti hakea kaikki kuvatiedostot, siirtää ne tiettyyn sijaintiin, ja nimetä ne sisältämään pelkkiä pienaakkosia, voitaisiin monen manuaalisen komennon toistamisen sijaan kirjoittaa yksinkertainen skripti, joka hoitaa kaikki tarvittavat toiminnot. Tämän jälkeen koko tehtävän pystyisi hoitamaan yhdellä komennolla, tai jopa automatisoimaan kokonaan käyttäen Cron- tai Anacron-ohjelmaa.

Pääasiassa kaikki, mitä Shell scriptillä voidaan tehdä, voitaisiin kirjoittaa myös jollain toisella ohjelmointikielellä, kuten Perl tai Python. Shell script on kuitenkin monesti nopein mahdollinen tapa tehdä yksinkertaista tarvetta vastaava koodi. Shell scriptin etuna on myös se, että se toimii yleensä kaikissa samaa shelliä käyttävissä ympäristöissä ilman erillistä tulkkia. Suoraan käyttöjärjestelmän komentotulkille ajettava koodi yleensä on erillisen tulkin läpi ajettua koodia nopeampi suorittaa.

Shell script on kuitenkin komentotulkkiin pohjautuessaan jokseenkin rajoittunut, verrattuna uudempiin kieliin, eikä sitä siksi ole yleensä mielekästä käyttää suuremmissa projekteissa. Lisäksi suoraan komentotulkille ajettavassa koodissa on riskinsä: tulkki tekee täsmälleen kuten sitä käsketään, eikä varaa koodausvirheelle ole. Lyöntivirhe, kuten väärä kirjain, tai välilyönti väärässä paikassa, saattaa pahimmassa tapauksessa aiheuttaa jopa koko järjestelmän rikkoutumisen.

Shell scriptin rajoitusten takia sitä ei päätetty käyttää toteutuksessa varsinaisen ohjelman tekoon, mutta lyhyiden ajastukseen liittyvissä komentoriveissä sitä on käytössä. Varsinkin tiedostojen siirtämiseen ja poistamiseen liittyvissä tehtävissä totesin Shell scriptin olevan hyödyllinen.

5.4 Muut käytetyt ohjelmat

5.4.1 Toiminnanohjausjärjestelmä (ERP)

Devlab Oy:n kehittämä Pupesoft on avoimen lähdekoodin GPL-lisensoitu toiminnanohjausjärjestelmä. Ohjelma on kirjoitettu pääasiassa PHP:lla, joskin osaa automaatioista ohjataan pienehköillä shell scripteillä. Käyttöliittymä on selainpohjainen, ja HTML- ja CSS-kielillä.

Markkinoilla on kymmeniä avoimen lähdekoodin toiminnanohjausjärjestelmiä, kuten Compiere, OpenBravo, OpenERP ja WebERP (Toiminnanohjaus 2010). Pupesoft valittiin tilausautomaatin rinnalle sen avoimen lähdekoodin, selainpohjaisen käyttöliittymän sekä Suomalaisen kehittäjäyrityksen takia. Valintaa helpotti myös se, että Pupesoft on käytössä monilla suomalaisilla yrityksillä. Harjoitteluun käytettävä työkalu on käytössä reaalimaailman yrityksillä, joten siirtymä virtuaaliyrityksestä varsinaiseen yritykseen on mahdollisimman helppo.

Avoin lähdekoodi on projektissa tärkeää, koska parhaimman mahdollisuuden yhteensopivuuden takaamiseksi on oleellista nähdä, miten ohjelman eri osat toimivat. Lisäksi tilausautomaatti vaatii pieniä muutoksia joihinkin Pupesoftin tiedostoihin. Näiden muokkausten tekeminen ei olisi mahdollista ilman avoimen lähdekoodin lisenssiä. Toisena tärkeimmistä kriteereistä on selainpohjaisuus, sillä sen avulla oppimisympäristöä voidaan tarjota helposti palveluna, ilman että loppukäyttäjän tarvitsee asentaa koneelleen mitään selaimen lisäksi. Lähes kaikki ohjelman osat toimii ilman JavaScriptiä, joten sekään ei ole varsinaisesti vaatimus ohjelman toiminnalle.

Tilausautomaatin ei ole tarkoitus olla täysin itsenäinen ohjelma, vaan se tukeutuu vahvasti Pupesoftiin, siksi myös ohjelmointikielet täsmäävät. Pupesoft tallentaa kaikki ohjelmassa tehdyt tapahtumat ja toiminnot MySQL-tietokantaan, mistä ne ovat täysin tilausautomaatin luettavissa. Automaatti lukee yritysten toimintoja, kuten yhteistyökumppaneiden määrää, markkinointipanosta, toimipisteen kuluja, asiakassuhteita, henkilöstöpanosta ja työaikoja. Näiden tietojen perusteella lasketaan yritykselle suhteellinen tilausarvon kerroin. Mitä

enemmän yritystoimintaan on panostettu, sitä suurempi on yritykselle tulevien tilausten arvo.

5.4.2 Sähköpostin välitysohjelma (MTA)

Tilausautomaatin toiminnan kannalta on välttämätöntä, että sitä ajava palvelin on kykenevä lähettämään sähköpostia. Tämä toiminnallisuus on toteutettu Postfix-sähköpostinvälitysohjelmalla (mail transfer agent). Postfix on muiden Futuralityn osien tapaan avointa lähdekoodia. Vaikka Fedora Linuxin oletusohjelma sähköpostin välitykseen onkin sendmail, on se tilausautomaattia varten vaihdettu Postfixiin. Syynä vaihdokseen on osittain Sendmailin tietoturvaongelmat (Cnet 2006; Cert 2003), mutta pääsyynä on omakohtaiset huonot kokemukset Sendmailin luotettavuudesta.

Sendmaililla toteutettu sähköpostin välitys on kahdessa ylläpitämässäni ympäristössä jumiutunut siten, että lähetetyt viestit on jääneet lähtevien postien jonoon, eivätkä ole päässeet sieltä eteen päin. Lähtevän sähköpostin jonoa on joutunut manuaalisesti muokkaamaan, joko lähettämällä satoja jonoon jääneitä posteja uudelleen, tai poistamalla jonoon jumiutuneita viestejä. Kumpikaan ratkaisuista ei ole kovin käyttäjäystävällinen, eikä ylläpitäjän kannalta toivottava tilanne. Postfixin kanssa en ole törmännyt ongelmiin, eikä sen tietoturvaongelmista löydy samanlailla uutisointia kuin Sendmailista. Koska ajastetut toiminnot saattavat varsinkin käyttäjämäärän kasvaessa aiheuttaa sähköpostipalvelimella suurtakin rasitetta, ei sen toiminnassa voida tehdä kompromisseja

6 AJASTUSTEN TOTEUTUS

6.1 Ajastuksen ympäristö

Käytettävänä ympäristönä on Futurable Oy:n testipalvelin, joka on Linux Fedora 13 –käyttöjärjestelmällä toimiva virtuaalipalvelin. Toteutus tehtiin alustavasti tuotantopalvelimen sijaan testipalvelimella lähinnä siksi, että pystyin toimimaan rajoittamattomilla oikeuksilla erilaisia ratkaisuja kokeiltaessa: asentamaan ja poistamaan haluamiani ohjelmia ja ajamaan ajastuksia parhaaksi näkemilläni käyttöoikeuksilla. Toteutuksen kannalta välttämättömät asennetut osat ovat käyttöjärjestelmän lisäksi Apache 2 httpd –webpalvelinohjelmisto, Postfix-MTA, PHP-tulkki, MySQL-tietokantaohjelma ja Cron-ajastusohjelma. Analysointia ja työprosesseja helpottamassa käytettiin Linuxin komentotulkki bashin lisäksi myös Webmin-ohjelmaa palvelimen eri toimintojen seurantaan, phpMyAdmin-ohjelmaa tietokantojen hallintaan, sekä htop-ohjelmaa prosessien työkuorman seurantaan.

Virtuaalinen oppimisympäristö Futurality, jonka osaksi ajastus tehdään, on kooste erilaisia liiketoiminnan opetukseen käytettäviä ohjelmia ja työkaluja. Toteutetut ajastukset hoitavat näiden ohjelmien ja niiden osien automaattisia toimintoja, sekä ohjelmien välistä tiedonsiirtoa. Ohjelmat sijaitsevat samalla palvelimella, ja käyttävät osittain yhteisiä tietokantoja, ja jopa yhteisiä tietokannan tauluja. Kaikkia eri ohjelmia ja niiden osia ei ole kuitenkaan voitu yhdistää tietokantatasolla, minkä takia tiedon synkronointia on täytyy tehdä säännöllisin väliajoin ajastetuilla toiminnoilla. Osa ohjelmista vaatii myös itsenäiseen toimintaansa liittyviä ajastuksia, kuten ostotilausautomaatin tilausten luonti ja lähetys. Lisäksi osaa tietokannan tauluista päivitetään automaattisesti julkisista RSS-syötteistä. Oppimisympäristön eri osat ovat rakennettu käyttäen pääasiassa XHTML-, PHP- ja MySQL-kieliä.

6.2 Ajastettavat osat

Tilausautomaattiin liittyviä automaatioita on kaksi: tilausten luonti viikoittain ja tilausten lähetys 29 minuutin välein. Tämän lisäksi Futuralityssa on myös kuukausittainen ja päivittäinen automaatio virtuaaliympäristön pankin ja verottajan toimintoja varten. Kaikki ajastukset on toteutettu Cronilla. Seuraavassa taulukossa (taulukko 2) toteutettavat ajastukset ja niiden tämänhetkiset funktiot.

TAULUKKO 2. Käytetyt ajastukset

Aikaväli	Kuukausittain	Viikoittain	Päivittäin	29 min. välein
Ajankohta	Kuukauden ensimmäinen päivä	Maanantaisin	Kello 2:45	29 min. välein arkipäivisin
Tehtävä(t)	Korkojen laskenta, Lainojen lyhennys	Tilausten luonti	Euribor-kurssien päivitys, verotilien täsmäys	Tilausten lähetys

Tilausautomaatin viikoittain, joka maanantai, laukeavan ajastuksen tehtävänä on lukea tilausasetukset ja luoda niiden perusteella tilaukset asianmukaisille virtuaaliyrityksille. 29 minuutin välein laukeavan ajastus vastaavasti tarkastaa aina lauetessaan, onko sillä tehtäviä tilauksia. Jos tilauksia löytyy, lukee skripti tilaukseen liittyvät kerroinasetukset ja tarvittavat yhtiötiedot, laskee niiden perusteella tilauksen ja lähettää sen asianmukaiselle yhtiölle. Jos tehtäviä tilauksia ei ole, skripti ei tee mitään. Ajastuksen laukeamisväli on juuri 29 minuuttia, eikä esimerkiksi tasan 30 minuuttia, jotta lähetettävien ostotilausten lähetysaika vaihtelee. Jos ajastuksen laukeamisväli olisi tasan 30 minuuttia, tulisi ostotilaukset virtuaaliyrityksille aina joko kellon ollessa tasan (xx:00), tai 30 minuuttia yli (xx:30).

Erikoisen ajastusvälin ainoa tarkoitus on luoda realistisuutta tilausten lähetykseen. Tilausautomaatti ei luo tilauksia kuin arkipäiville, joten ajastustakaan ei ajeta kuin maanantaista perjantaihin. Sekä tilausautomaatti että Cron toimivat arkipäivien kohdalla sillä yksinkertaisella logiikalla, että päivät

maanantaista perjantaihin ovat arkipäiviä ja muut päivät ei. Tilauksia voi näin tulla näille päiville osuville pyhäpäiville. Ostotilauksia ei tarvitse kuitenkaan käsitellä heti, joten tästä ei pitäisi muodostua ongelmaa.

6.3 Cronin käyttö

Cronin ajastetut toiminnot kirjoitetaan taulukkomuotoisiin crontab-tiedostoihin, joita Cron lauetessaan lukee. Crontab-tiedostoissa on kerrottu Cronille kaikki ajastuksen ajamiseen tarvittavat tiedot, eli käynnistysten ajankohdat ja ajettava skripti. Lisäksi ainakin Red Hat Linuxista ja Debian Linuxista löytyy crontab-tiedosto, jossa myös skriptin ajajan voi määrittää. Tiedosto sijaitsee oletusarvoisesti sijainnissa /etc/crontab. Lisäksi Cron hakee oletuksena crontab-tiedostoja sijainnista /etc/cron.d/. Tässä sijainnissa voi olla /etc/crontab-tiedoston kaltaisia tiedostoja haluttu määrä.

Tiedostojen käyttötapa on molemmissa sijainneissa sama. Erotuksena näillä kahdella on se, että /etc/crontab-tiedostossa kaikkien ajettavien komentojen on oltava samassa tiedostossa, kun taas /etc/cron.d/-kansiosta ajettavat komennot voivat olla ryhmiteltyinä eri tiedostoihin. Vaikka käyttäjä voi mieltymyksistään riippuen käyttää kumpaa tahansa tapaa, on suositeltavaa valita käyttöön vain toinen. Jos molempia sijainteja aikoo käyttää, täytyy olla tarkkaan selvillä, mitä mistäkin ajetaan. Crontabin manuaalista (2011, 5) voidaan tarkastaa crontab-tiedoston syntaksi.

```
# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12)
# |  |  |  |  .---- day of week (0 - 6)
# |  |  |  |  |
# *  *  *  *  * user-name  command to be executed
```

KOODIESIMERKKI 5. crontab-tiedoston syntaksi (crontab man 2011)

Tiedostoissa taulukon eri solut on eroteltu välilyönnillä. Ensimmäinen solu kertoo ajettavan minuutin, toinen solu ajettavan tunnin, kolmas solu ajettavan päivän, neljäs solu ajettavan kuukauden ja viides solu ajettavan viikonpäivän. Minuuteissa voidaan käyttää arvoja 0-59, tunneissa voidaan käyttää arvoja 0-23, päivissä voidaan käyttää arvoja 1-31, kuukausissa voidaan käyttää arvoja 1-12 tai englanninkielisiä lyhenteitä jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, viikonpäivissä numeroarvoja 0-6 (ja 7) tai englanninkielisiä lyhenteitä sun,mon,tue,wed,thu,fri,sat. Viikonpäivissä sekä 0 että 7 tarkoittavat sunnuntaita. Kuudennessa solussa on joko komennon ajava käyttäjä tai ajettava komento, riippuen crontab-tiedoston sijainnista. Jos kuudes solu on komennon ajava käyttäjä, on seitsemäs solu ajettava komento.

Soluihin voidaan myös antaa niin sanottuja wildcard-arvoja, joilla voidaan kuvata montaa eri arvoa. Edeltävässä kuvaajassa näkyy yleisin wildcard-merkki ’*’. Tämä merkki kuvaa arvoa ”mikä tahansa”. Minuutin kohdalla tämä tarkoittaa, että komento ajetaan joka minuutti, tunnin kohdalla, että komento ajetaan joka tunti, päivän kohdalla, että joka päivä jne. Tämän merkin kanssa täytyy olla tarkkana varsinkin minuutin kohdalla, sillä komennon voi määrittää ajettavaksi vaikka joka minuutti. Wildcard-arvoihin voidaan antaa myös aikavälejä, kuten ”*/5”. Minuuttia kuvaavassa solussa ”*/5” tarkoittaisi ”joka viides minuutti”. Joka viides minuutti ajettava komento ajastettaisiin:

```
*/5 * * * * {käyttäjä} {komento}
```

KOODIESIMERKKI 6. Cron-ajastus 5 minuutin välein

Aikoihin voidaan määrittää myös monta arvoa tai aikavälejä. Monta arvoa erotellaan pilkuilla. Viikonpäivää kuvaavassa solussa ”0,6” tarkoittaisi ”sunnuntaisin ja lauantaisin”. Pilkun jälkeen ei saa tulla välilyöntiä, tai Cron tulkitsee solun loppuvan. Lisäksi crontabin viikonpäiviä käyttäessä täytyy huomata, että viikonpäivien lasku alkaa sunnuntaista, joka on sekä 0, että 7. Tämän ajastuksen voisi siis tehdä kummalla tahansa seuraavista esimerkeistä (koodiesimerkki 7; koodiesimerkki 8), ja lopputulos olisi sama.

```
0 2 * * 0,6 {käyttäjä} {komento}
```

KOODIESIMERKKI 7. Cron-ajastus joka sunnuntai, tapa 1

```
0 2 * * sun,sat {käyttäjä} {komento}
```

KOODIESIMERKKI 8. Cron-ajastus joka sunnuntai, tapa 2

Aikavälin määrittäminen tapahtuu ”-”-merkillä. Kello 6:00-8:00 välillä tunneittain tehtävä ajastus kirjoitettaisiin crontabiin

```
0 6-8 * * * {käyttäjä} {komento}
```

KOODIESIMERKKI 9. Aikavälin määrittäminen crontab-tiedostossa

Ajastusten määrityksessä voi käyttää viiden numeroarvon sijaan myös crontabin manuaalisivuilta (2011) löytyviä erikoiskomentoja:

```
komento         tarkoitus
------          -------
@reboot         Aja kerran käynnistyksen yhteydessä
@yearly         Aja kerran vuodessa, "0 0 1 1 *".
@annually       (sama kuin @yearly)
@monthly        Aja kuukausittain, "0 0 1 * *".
@weekly         Aja viikoittain, "0 0 * * 0".
@daily          Run once a day, "0 0 * * *".
@midnight       (sama kuin @daily)
@hourly         Aja tunneittain, "0 * * * *".
```

KOODIESIMERKKI 10. crontabin ajastuksen erikoiskomennot (Crontab man 2011)

Lisäksi on olemassa myös crontab-tiedostoja, joissa skriptin ajajaa ei erikseen määritetä, vaan ajaja on tiedoston omistaja. Esimerkki tällaisesta crontab-tiedostosta on jokaisen käyttäjän henkilökohtainen crontab, jossa määritetään vain suoritusaika ja suoritettava komento. Käyttäjäkohtaista crontabia pääsee muokkaamaan komennolla crontab -e. Tämä crontab-tiedosto sijaitsee oletusarvoisesti sijainnissa /var/spool/cron/$USER. Esimerkiksi root-käyttäjään henkilökohtainen crontab-tiedosto sijaitsee oletuksena sijainnissa /var/spool/cron/root. Joissain UNIX- ja Linux-järjestelmissä sijainti voi olla myös /var/spool/cron/crontabs/root.

Cronille voidaan myös määritellä ylimääräisiä kansioita, joista ajettavia crontab-tiedostoja haetaan. Useimmissa Debian Linuxiin ja Red Hat Linuxiin pohjautuvissa Linux-jakeluissa on oletusarvoisesti kansiot /etc/cron.hourly/, /etc/cron.daily/, /etc/cron.weekly/ ja /etc/cron.monthly/. Jokainen sijainti on määritelty Cronin tarkastettavaksi kansioiden nimien mukaan: cron.hourly tarkistetaan tunneittain, cron.daily päivittäin, cron.weekly viikoittain ja cron.monthly kuukausittain. Jos kansioista löytyy skriptejä, ne ajetaan aina tarkastuksen yhteydessä. Kansioiden varsinaiset tarkastusajankohdat on

määritettävä jossakin Cronin jo ajamassa crontab-tiedostossa, kuten /etc/crontab:issa. Esimerkkinä /etc/crontab-tiedoston sisällöstä voisi olla:

```
08 * * * * root run-parts /etc/cron.hourly
45 2 * * * root run-parts /etc/cron.daily
15 3 * * 1 root run-parts /etc/cron.weekly
30 3 1 * * root run-parts /etc/cron.monthly
```

KOODIESIMERKKI 11. crontab-esimerkkitiedosto

Kyseisen crontab-tiedoston tehtävinä ei ole muuta kuin ajaa mainitussa kansioissa olevia skriptejä. Komento ”run-parts” ajaa kaikki määritellyssä sijainnissa olevat skriptit. Ensimmäisellä rivillä (koodiesimerkki 12) määritetään kaikki kansiossa /etc/cron.hourly olevat tiedostot ajettavaksi joka tunti minuuttien ollessa 08 (00:08, 01:08, 02:08, ..., 23:08) joka päivä.

```
08 * * * * root run-parts /etc/cron.hourly
```

KOODIESIMERKKI 12. crontab-esimerkkitiedoston ensimmäinen rivi

Toisella rivillä (koodiesimerkki 13) määritetään kaikki kansiossa /etc/cron.daily olevat tiedostot ajettavaksi joka päivä kello 02:45.

```
45 2 * * * root run-parts /etc/cron.daily
```

KOODIESIMERKKI 13. crontab-esimerkkitiedoston toinen rivi

Kolmannella rivillä (koodiesimerkki 14) määritellään kaikki kansiossa /etc/cron.weekly olevat tiedostot ajettavaksi joka viikko maanantaina kello 03:15.

```
15 3 * * 1 root run-parts /etc/cron.weekly
```

KOODIESIMERKKI 14. crontab-esimerkkitiedoston kolmas rivi

Neljännellä rivillä (koodiesimerkki 15) määritellään kaikki kansiossa /etc/cron.monthly olevat tiedostot ajettavaksi joka kuukauden 1. päivänä kello 3:30.

```
30 3 1 * * root run-parts /etc/cron.monthly
```

KOODIESIMERKKI 15. crontab-esimerkkitiedoston neljäs rivi

Käyttäjä voi myös tehdä tarvittaessa lisää vastaavia kansioita, kuten cron.annually tai cron.weekends. Nämä kansiot voisi määrittää tarkastettavaksi vuosittain tai joka lauantai ja sunnuntai. Vuosittainen ajo tammikuun ensimmäisenä päivänä kello 3:00 tehtäisiin seuraavan esimerkin (koodiesimerkki 16) mukaan.

```
0 3 1 1 * root run-parts /etc/cron.annually
```

KOODIESIMERKKI 16. Cron-ajo vuosittain

Lauantaina ja sunnuntaina kello 3:00 toteutuva ajo voitaisiin tehdä esimerkiksi seuraavalla tavalla (koodiesimerkki 17).

```
0 3 * * 0,6 root run-parts /etc/cron.weekends
```

KOODIESIMERKKI 17. Cron-ajo lauantaisin ja sunnuntaisin

Crontabeihin määriteltävät suoritusajat voidaan yleensä määrittää myös käyttämällä crontabin erikoiskomentoja, joista lisää myöhemmin.

Kaikille ajankohdille ei ole tietenkään järkevää tehdä omia sijainteja, vaan pelkästään usein käytettävät ajankohdat on järkevää ajaa näin. Komentoja

voidaan ajaa myös suoraan natiiveista crontab-tiedostoista, joten tällaisia aikaväliin sidottuja sijainteja ei ole millään lailla välttämätöntä käyttää. Cronia voi käyttää monella eri tavalla riippuen käyttäjän omista mieltymyksistä. Kaikki ajettavat komennot voidaan laittaa yhteen tiedostoon, kaikki ajettavat komennot voidaan laittaa omiin tiedostoihinsa, tai mitä tahansa näiden väliltä. Paljon ajastuksia tekevällä palvelimella komentoja kannattaa ryhmitellä eri tiedostoihin tai kansioihin selkeyden vuoksi.

6.4 Toteutetut ajastukset

Toteutuksessa ryhmittelyä on käytetty hyväksi tekemällä jokaiseen erilliseen toimintoon liittyville komennoille oma tiedosto. Tarvittavia tiedostoja on näin ollen kolme: osatiedostojen ajon ajastava /etc/crontab, virtuaalipankin toimintoja ajastava /etc/cron.d/pankki, tilausten luonnista ja lähetyksestä huolehtiva /etc/cron.d/orderAutomation. Osatiedostoja ajavan /etc/crontab-tiedoston sisältö esitetty alla (koodiesimerkki 18).

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
08 * * * * root run-parts /etc/cron.hourly
45 2 * * * root run-parts /etc/cron.daily
15 3 * * 1 root run-parts /etc/cron.weekly
30 3 1 * * root run-parts /etc/cron.monthly
```

KOODIESIMERKKI 18. /etc/crontab-esimerkkitiedosto

Tiedoston tehtävänä on tarkastaa tunneittain, päivittäin, viikoittain ja kuukausittain vastaavien kansioiden skriptit. Tiedoston alussa olevat määritykset ovat käytettävä komentotulkki (SHELL=/bin/bash), tulkin sijainti (PATH=/sbin:/bin:/usr/sbin:/usr/bin), sekä virheilmoitusten raportoinnin saaja (MAILTO=root). Ajastuskenttien jälkeen tuleva käyttäjännimi ”root” tarkoittaa,

että toiminnot ajetaan root-käyttäjän oikeuksilla. root-käyttäjän käyttöoikeudet palvelimella on lähes rajattomat, joten tällä käyttäjätunnuksella ajettavien komentojen kanssa täytyy olla huolellinen.

Ajastetuilla Cron-ajoilla on täysin mahdollista ajaa komentoja, jotka tekevät peruuttamatonta vahinkoa järjestelmälle. Ajettaessa run-parts -komentoa root-oikeudet ovat monesti kuitenkin välttämättömiä riittävien luku-, suoritus- ja kirjoitusoikeuksien saamiseksi. Kaikissa muissa esiteltävissä crontab-tiedostoissa käyttäjänä onkin käytetty webpalvelinohjelma apachen käyttäjäoikeuksia. Virtuaalipankin toimintoja ajastava tiedosto sisältää seuraavat rivit (koodiesimerkki 19).

```
15 3 * * 1 apache php /automation/LoanAutomation.php
updateEuribor >> /automation/euribor_log

30 3 1 * * apache php /automation/LoanAutomation.php
makeLoanRepayments >> /automation/repayment_log

30 3 1 * * apache php /automation/EventFeeAutomation.php >>
/automation/eventFee_log
```

KOODIESIMERKKI 19. Virtuaalipankin ajastuksia

Ensimmäinen komento ajetaan viikoittain, ja sen funktio on päivittää Euribor-korkoprosentit RSS-feedistä tietokantaan. Rivin lopussa oleva rivi (koodiesimerkki 20) on yksinkertainen Shell script-komento, jolla tiedoston antama palaute ohjataan euribor_log-lokitiedostoon.

```
”-- >> /automation/euribor_log”
```

KOODIESIMERKKI 20. Palautteen ohjaaminen lokitiedostoon

Toinen ja kolmas komento ajetaan kuukausittain, ja niiden tarkoituksena on laskea kuukausittaiset korot, sekä tapahtumamaksut. Ensimmäisen komennon tapaan niiden antama palaute ohjataan kullekin toiminnolle ominaiseen lokitiedostoon. Komentojen ajastukset ovat identtisiä edellisen crontab-tiedoston run-parts-ajastusten kanssa. Nämä skriptit voitaisiin siis myös pilkkoa omiksi

tiedostoikseen ja asettaa ne /etc/cron.weekly/- ja /etc/cron.monthly/-kansioihin, jolloin toteutusajat ja toiminta olisi täysin nykyistä tilannetta vastaava. Komennot ovat ylläpidon helpottamiseksi koottu yhteen tiedostoon. Seuraava crontab-tiedosto /etc/cron.d/orderAutomation (koodiesimerkki 21) huolehtii tilausautomaation toteutuksesta.

```
15 3 * * 1 apache php /automation/OrderAutomation.php
createOrders >> /automation/automation_log

*/29 * * * * apache php /automation/OrderAutomation.php
sendOrders >> /automation/automation_log
```

KOODIESIMERKKI 21. Tilausautomaatin ajastuksia

Tässä tiedostossa on ajastukset tilausten luonnille, sekä lähetykselle. Tilauksia luova rivi ajetaan kerran viikossa, ja se voitaisiinkin jälleen sijoittaa myös cron.weekly-kansioon. Pankin ajastusten tapaan tiedoston antama palaute tallennetaan lokitiedostoon. Tiedoston toinen komento ajetaan 29 minuutin välein, ja sen tarkoitus on tarkastaa onko tehtäviä tilauksia odottamassa, ja tarvittaessa lähettää odottava tilaus. Näin usein ajettavalla tiedostolla virheraportointi ja muun lokitiedon tallennus kannattaa suunnitella hyvin, sillä tiedoston ajokertoja tulee päivässä kymmeniä.

7 TYÖN ARVIOINTI

Ajastustekniikoiden laajasta kirjosta huolimatta varteenotettavien vaihtoehtojen rajaus oli odotettua selkeämpää. Toimivalta vaikuttavia tekniikoita ei tarvinnut rajata pois. Pois rajatut vaihtoehdot olivat pääasiassa opinnäytetyössä esiteltyihin tekniikoihin pohjautuvia toteutustapoja, poikkeuksena Windows-käyttöjärjestelmän Scheduled tasks, jonka rajasin luonnollisesti jo käyttöjärjestelmän takia.

Esitellyt tekniikat ovat yleisesti käytössä ja toteutettavissa erilaisiin ympäristöihin. Vaihtoehdoissa on myös mukavasti vaihtelua toteutustavan teknisyydessä. Tekniikat lisäksi skaalautuvat yksinkertaisesta, nopeasti kokoonpantavasta, toteutustavasta monimutkaisiin ratkaisuvaihtoehtoihin, joissa osien toiminta-ajat ja toteutusjärjestys täytyy olla tarkkaan määritetty. Futurality-virtuaaliympäristöön toteutetut ajastukset ovat toimineet odotusten mukaan hyvin, eikä mitään ongelmia ole varsinaisesti tullut.

Huomionarvoinen seikka kuitenkin on, että hyvin tiheään ajastettavien toimintojen virheidenkäsittely kannattaa suunnitella etukäteen. Tilausten lähetyksestä vastaava ajastus oli alunperin 7 minuutin välein, mikä aiheutti testauksen aikana pienen ongelman: muokkasin tilausten lähetykseen käytettävää skriptiä siten, että tehtävien tilausten tarkastus ei toiminut oikein, vaan antoi tietyssä tilanteessa virheilmoituksen. Testasin muutoksia tietysti manuaalisesti, mutta en testannut kaikkia mahdollisia tilanteita. Virheellistä skriptiä koitettiin ajaa koko yön, mikä aiheutti kymmeniä virheilmoituksia. Virheilmoitukset olin ohjannut sähköpostiini, ja jokaisesta virheilmoituksesta oltiin määritetty lähetettäväksi ilmoitus.

Tiheään ajettavat skriptit on testattava hyvin, ja ajojen tiheyttä ei kannata määrittää tarpeettoman lyhyin väliajoin. 7 minuutin välein tehtävällä ajolla ei ollut merkittävää etua 29 minuutin välein ajoon, mutta en osannut aluksi ennustaa lyhyen aikavälin tarkastuksen riskejä. Jos vastaava virhe tapahtuisi esimerkiksi sellaisen skriptin kanssa, joka kuittaa laajalle sähköpostilistalle lähetetyn massaviestin lähetyksen tehdyksi, mutta ei ikinä saisi kuittausta tehtyä onnistuneesti, lähettäisi palvelin 7 minuutin välein saman massapostin niin kauan, että virhe korjataan.

Toteutettu tilausautomaatti on testikäytössä Futurality-virtuaaliympäristön testipalvelimella. Seuraan ohjelman toimintaa loppuvuoden, ja teen tarvittaessa korjauksia ja lisäyksiä. Automaatti toimii kuitenkin tähänastisen testauksen perusteella niin kuin pitäisikin, joskaan laajempaa testausta ja monipuolisempia tapahtumaketjuja päästään testaamaan vasta pilottivaiheen alkaessa. Pilottivaiheessa testaajia on enemmän ja testaajilla on erilaisia lähtökohtia osaamisen ja käyttötottumusten suhteen, joten erilaisia käyttötilanteita, ja niiden myötä mahdollisesti erilaisia virhetilanteita, syntyy enemmän. Kaikkia mahdollisia virhekäyttötilanteita ja vikoja on todella hankala saada aikaiseksi ilman varsinaisia loppukäyttäjiä.

Vaikka olen tehnyt aiemminkin ajastuksia niin webympäristöön, kuin UNIX- ja Microsoft Windows –käyttöjärjestelmille, ovat ajastusten varsinainen toimintalogiikka ja eri ajastustekniikoiden ominaisuudet olleet jokseenkin huonolla teoriapohjalla. Opinnäytetyön kautta olen oppinut ajastustekniikoista paljon aiemman käytännön osaamisen tueksi, ja pystyn nyt paremmin hyödyntämään erilaisia ajastustekniikoita juuri niille sopivassa ympäristössä.

LÄHTEET

Adobe. 2011. Adobe – Flash Player
http://www.adobe.com/software/flash/about/
Tarkistettu 26.10.2011

Anacron man. 2011. Linux anacron manual
http://linuxmanpages.com/man8/anacron.8.php
Tarkistettu 26.09.2011

Cert. 2003. CERT® Advisory CA-2003-25 Buffer Overflow in Sendmail
http://www.cert.org/advisories/CA-2003-25.html
Tarkistettu 10.10.2011

Cnet. 2006. Sendmail flaw opens door to intruders
http://news.cnet.com/2100-1002_3-6052758.html
Tarkistettu 10.10.2011

Cron man. 2011. Linux cron manual
http://linuxmanpages.com/man8/cron.8.php
Tarkistettu 26.09.2011

crontab man. 2011. Linux crontab manual
http://linuxmanpages.com/man1/crontab.1.php
Tarkistettu 26.09.2011

Futurable Oy. 2011. Futurality - simuloitu liiketoimintaympäristö.
http://www.futurable.fi/Index.php?page=Ymparisto
Tarkistettu 26.09.2011

Langpop 2011. Programming Language Popularity
http://langpop.com/
Tarkistettu 07.10.2011

Oracle Corporation. 2011. MySQL :: MySQL 5.6 Reference Manual
http://dev.mysql.com/doc/refman/5.6/en/
Tarkistettu 19.10.2011

Microsoft. 2011. About | Microsoft Silverlight
http://www.microsoft.com/silverlight/what-is-silverlight/
Tarkistettu 26.10.2011

Netcraft. 2011. Internet Research, Anti-Phishing and PCI-security Services.
http://news.netcraft.com/archives/2011/09/06/september-2011-web-server-survey.html
Tarkistettu 03.10.2011

Oracle 2011. Java Web Start Overview

http://www.oracle.com/technetwork/java/javase/overview-137531.html
Tarkistettu 26.10.2011

php.net. 2011a. PHP: Hypertext Preprocessor.
http://www.php.net/
Tarkistettu 07.10.2011

php.net. 2011b. PHP: General Information - Manual
http://fi.php.net/manual/en/faq.general.php
Tarkistettu 18.10.2011

php.net. 2011c. PHP: PHP Usage Stats
http://www.php.net/usage.php
Tarkistettu 07.10.2011

php.net 2011d. PHP: Introduction – Manual
http://www.php.net/manual/en/oop5.intro.php
Tarkistettu 07.10.2011

PostgreSQL. 2009. Cross Compare of PostgreSQL 8.4, SQL Server 2008, MySQL 5.1 - Postgres OnLine Journal.
http://www.postgresonline.com/journal/index.php?/archives/130-Cross-Compare-of-PostgreSQL-8.4,-SQL-Server-2008,-MySQL-5.1.html
Tarkistettu 10.10.2011

qmail. 2011. qmail: Second most popular MTA on the Internet
http://qmail.org/top.html
Tarkistettu 10.10.2011

ServerSchool. 2011. Popular Server Operating Systems.
http://www.serverschool.com/server-software/popular-web-servers/
Tarkistettu 07.10.2011

Shearer 2007. MTA Comparison
http://shearer.org/MTA_Comparison
Tarkistettu 10.10.2011

Statowl, 2011. Web Browser Plugin Market Share / Global Usage
http://www.statowl.com/plugin_overview.php
Tarkistettu 07.10.2011

Toiminnanohjaus, 2010. ERP Toiminnanohjaus – Open source ERP
http://www.toiminnanohjaus.fi/index.php?option=com_content&task=view&id=20&Itemid=45
Tarkistettu 10.10.2011

W3Schools. 2011. Browser statistics.
http://www.w3schools.com/browsers/browsers_stats.asp
Tarkistettu 07.10.2011

Webcron. 2011. Webcron Server Monitoring & Online Cron
http://www.webcron.org/
Tarkistettu 10.11.2011

Mywebcron. 2011. My Free Web Cron Schedule Service
http://www.mywebcron.com/
Tarkistettu 10.11.2011

Zandstra, M. 2004. PHP 5 Objects, Patterns, and Practice.
California: Apress.