Metropolia Ammattikorkeakoulu
Tietotekniikan koulutusohjelma

Arto Iijalainen

Tiedonsiirtomoduulin toteuttaminen Drupal-sisällönhallintajärjestelmälle

Insinöörityö 17.3.2009

Ohjaaja: projektipäällikkö Santeri Lindgren
Ohjaava opettaja: lehtori Kimmo Saurén

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Arto Iijalainen Tiedonsiirtomoduulin toteuttaminen Drupal-sisällönhallintajärjestelmälle 44 sivua 17.3.2009
Koulutusohjelma	tietotekniikka
Tutkinto	insinööri (AMK)
Ohjaaja Ohjaava opettaja	projektipäällikkö Santeri Lindgren lehtori Kimmo Saurén

Insinöörityössä oli tavoitteena toteuttaa erillinen tiedonsiirtomoduuli asiakasjärjestelmään, joka oli Drupal-sisällönhallintajärjestelmään pohjautuva web-palvelu. Tiedonsiirtomoduulin päämäärä oli ensisijaisesti tuottaa palvelun tietorakenteista tiedostoja, joiden avulla olisi mahdollista siirtää informaatiota asiakasjärjestelmän ja ulkoisten järjestelmien välillä. Myöhemmässä vaiheessa asiakas lisäsi toissijaiseksi tavoitteeksi raporttien tuottamisen ja viemisen järjestelmästä.

Moduuli toteutettiin Drupal-järjestelmän määrittämien moduulivaatimusten mukaisesti PHP-ohjelmointikielellä. Sen lisäksi ohjelmakoodia tehostettiin käyttämällä oliorakenteita moduulin pääominaisuuksien pohjana. Suurin osa ominaisuuksista toteutettiin itse, mutta muutamassa erikoistapauksessa päädyttiin käyttämään kolmannen osapuolen ratkaisuja tiukan arviointiprosessin jälkeen.

Insinöörityön lopputuloksena syntyi moduuli, joka vastasi tarkasti asiakkaan asettamia lopullisia tavoitteita, mutta ei ollut täysin linjassa alkuperäisten tavoitteiden kanssa. Tämä johtui asiakasprojektin kehitysprosessin aikana tapahtuneista muutoksista, jotka johtivat tiedonsiirtomoduulin roolin vaihtumiseen ydinjärjestelmästä konseptidemoksi ja yleisemmäksi kehitysrungoksi.

Tulevaisuuden kehitystarpeet määräytyvät moduulin roolin mukaan. Mikäli moduulia halutaan käyttää isoissa järjestelmienvälisissä tiedonsiirtoajoissa, tarvitaan erillinen aliohjelma, joka käyttää jo olemassa olevia luokkatoteutuksia, mutta toimii varsinaisen siirron aikana itsenäisesti. Lisäksi informaation tuominen järjestelmään vaatii kyseistä esitysmuotoa vastaavan tuontiluokan toteuttamista.

Hakusanat	tiedonsiirto, Drupal, PHP, sisällönhallintajärjestelmä

Helsinki Metropolia University of Applied Sciences **Abstract**

Author	Arto Iijalainen
Title	Implementation of a data transfer module for a Drupal content management system
Number of Pages	44
Date	17 March 2009
Degree Programme	Information Technology
Degree	Bachelor of Engineering
Instructor Supervisor	Santeri Lindgren, Project Manager Kimmo Saurén, Senior Lecturer

The purpose of this project was to design a data transfer module to a client project, which was a Drupal-based web service. The data transfer module's primary objective was to generate external files from the service's data structures. The files were to provide a means to transfer information between the client system and external systems. Later on the client requested that report generation and export should be added as a second objective for the data transfer module.

The module was implemented according to the Drupal's definitions using the PHP programming language. In addition, the program code was optimized by using object oriented programming as a base for the main features. Most of the features were self-implemented but in some special cases it was decided to use third-party solutions after a critical assessment of the different possibilities.

The solution of the data transfer module was obtained as the result of this project. The outcome fulfills the final objectives set by the customer. However, the outcome does not meet all the original requirements. This is because of the shift change of the module's role from a key system to a concept demo and a general framework, originating from the changes in the development process of the client system.

The future development needs of the module will be defined by the module's role. If large amounts of data are required to be transferred between systems, external subprogram will be required to execute the process. The subprogram would use the existing program classes but would work independently. Also importing data to a client system requires an import class specialized to handle the given representation form of the incoming data stream.

Keywords	data transfer, Drupal, PHP, content management system

Sisällys

1 Johdanto

Internet-aikana on pikkuhiljaa päästy eroon laitteistopohjaisesta ajattelusta. Palvelinpuolelta saattaa löytyä kuinka paljon kapasiteettia tahansa, mutta se ei välity käyttäjälle asti – Googlen hakusivu näyttää yhtä yksinkertaiselta, vaikka palvelimena toimisi supertietokone tai antiikkinen kotikone. Toki yhtäaikaisten käyttäjien määrä ja tietokannan laajuus vaikuttavat tarvittavaan kapasiteettiin, mutta sitäkään ei voi järkevästi kertoa käyttäjälle muuten kuin raa'alla numerodatalla, jonka massiivisuutta käyttäjä ei pysty käsittämään. Toinen naula laitteistopohjaisen ajattelun arkkuun ovat asiakaspäätteet. Jokaisesta modernista matkapuhelimesta löytyy Internet-valmius, eli isonkaan tietomäärän selaamiseen ei näennäisesti tarvita kuin vähän konetehoa.

Nykyaikana vallitsee järjestelmä- eli palvelupohjainen ajattelumalli. Kaikkialta käytettävät, yleensä tarkasti tietyn asian ympärille rakentuneet Internet-palvelut keräävät huomiota ja käyttäjiä. Yleensä näiden järjestelmien tietorakenteet ovat yksinkertaisia ja ne noudattavat mielellään yhtä ”Web 2.0:n” tärkeimpänä pidettyä teemaa: käyttäjät luovat sisällön [1, s. 2]. Tämä tarkoittaa käytännössä sitä, että laadun sijaan luotetaan määrään. Kukaan ei varsinaisesti odota, että pelkällä innostuksella voi luoda mestariteoksia, mutta kun tietty kriittinen sisältömassa ylitetään, keskikertaisuuden seasta alkaa löytyä kultahippujakin. Periaatteessa tämä on nerokas tapa saada tuotettua paljon sisältöä halvalla ja eritoten käyttäjien ehdoilla. Toisaalta se johtaa väistämättä informaation keskimääräisen laadun romahtamiseen.

Järjestelmäpohjainen ajattelumalli on kaikesta huolimatta askel kohti entistä informaatio-orientuneempaa ajattelutapaa, jossa laitteiston lisäksi myös ohjelmistopalvelujen rajat hämärtyvät ja itse informaation arvo korostuu. Jo nykyään on olemassa palveluita, kuten Yahoo Pipes, jossa dataa voidaan hakea useammasta palvelusta ja yhdistää uudenlaisiksi kokonaisuuksiksi. Informaatio voi toimia pelkkänä uuden informaation pohjana tai sitä voidaan hyödyntää sellaisenaan eri järjestelmissä, kuten esimerkiksi selainten kirjanmerkkien välittäminen.

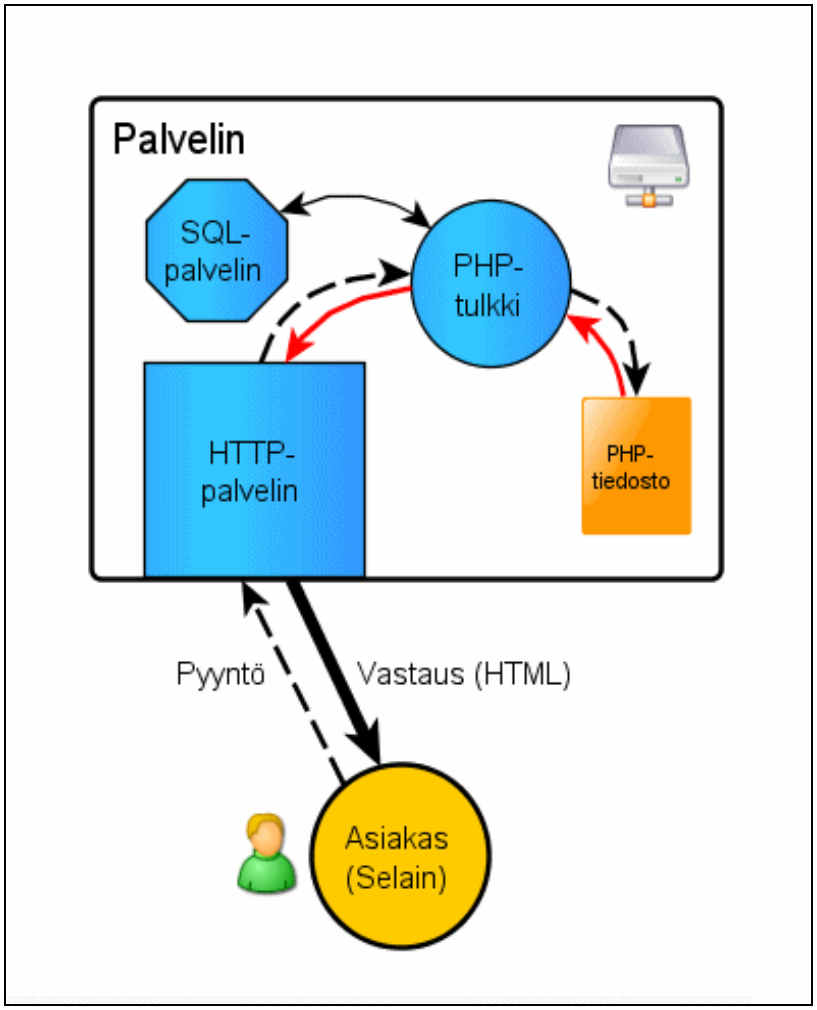
Luonnollisesti informaation siirtäminen järjestelmästä toiseen vaatii, että järjestelmien on pysyttävä kommunikoimaan keskenään. Kommunikaation ei välttämättä tarvitse olla synkronista tai edes kaksisuuntaista – usein on riittävää, että lähdejärjestelmä tarjoaa ylipäänsä jonkinlaisen tiedonsiirtorajapinnan, jota kohdejärjestelmä voi hyödyntää. Hyvänä esimerkkinä tästä on blogeihin vakioitunut RSS-syöte.

Tämän työn tavoitteena oli vastata edellä esitettyihin haasteisiin kehittämällä erillinen tiedonsiirtomoduuli asiakasjärjestelmään. Moduuli mahdollisti järjestelmään tallennetun informaation viemisen ja jatkokäsittelyn ulkoisissa järjestelmissä sekä informaation tuomisen ulkoisista lähteistä.

2 Käytetyt tekniikat

2.1 PHP-kieli

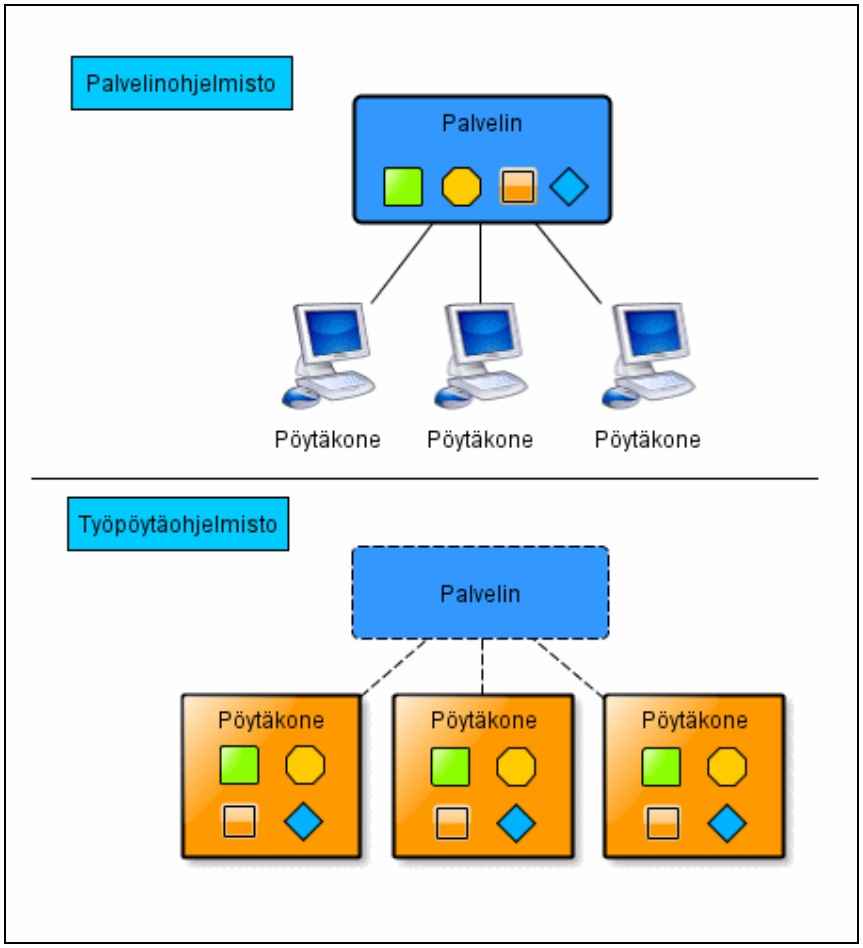
PHP on erityisesti dynaamisten verkkopalvelujen kehittämiseen tarkoitettu tulkattava ohjelmointikieli, joka on sittemmin laajentunut myös muille alueille. PHP on erittäin suosittu: nykyisten laskelmien mukaan se pyörittää 20 miljoonaa verkkosivustoa [2] miljoonalla palvelimella [3]. Suurin osa palveluista on pieniä, mutta myös suuret jättiläiset, kuten Facebook, Yahoo! ja Wikipedia (MediaWiki) ovat joko osittain tai kokonaan rakennettu PHP:n päälle [3].



Kuva 1. Palvelimen sisäinen toimintakaavio.

Verkkopalveluiden pohjaratkaisuna PHP-ohjelmia ajetaan yleensä verkkopalvelinympäristössä. Kuvassa 1 on esitetty palvelimen toimintatapa. Asiakas lähettää sivupyynnön HTTP-palvelimelle, joka huomaa, että kyseinen sivupyyntö koskee PHP-sivua staattisen HTML-sivun sijaan. Palvelin siirtää pyynnön edelleen PHP-tulkille. Tulkki käy läpi kohdesivun PHP-koodia sisältävät kohdat ja palauttaa (yleensä HTML-muotoisen) vastauksen takaisin HTTP-palvelimelle, joka lähettää sen eteenpäin asiakkaalle. PHP-sivun monimutkaisuus riippuu täysin sen tarkoitusperästä. Yksinkertainen esimerkkisivu voi olla täysin staattinen tai sisältää ainoastaan yksinkertaisen kellonajan tulostamisen, kun taas laaja palvelu voi sisältää useampia PHP-sivuja, joiden sisältö muodostuu täysin dynaamisesti sivukutsujen mukana lähetettävien parametrien pohjalta.

Kuvassa 2 vertaillaan palvelin- ja työpöytäsovelluksen eroja. Palvelinpohjaisten ohjelmistojen etu tavallisiin työpöytäsovelluksiin nähden on se, että asiakaskoneen ohjelmiston ei tarvitse olla monimutkainen – riittää, että se osaa lähettää asiakkaan syötteen palvelimelle ja vastavuoroisesti näyttää palvelinohjelmiston tuottaman tulosteen. Kaikki järjestelmän kannalta tärkeät tapahtumat hoidetaan keskitetysti palvelimella. HTML-pohjaiset verkkopalvelut edustavat yleiskäyttöisyytensä vuoksi tämän ajattelumallin laajimmin levinnyttä kärkeä niin asiakasohjelmistojen (selaimet) kuin varsinaisten palveluidenkin määrässä.



Kuva 2. Palvelinohjelmiston ja työpöytäohjelmiston eroavaisuudet.

PHP:n juuret ovat vahvasti avoimen lähdekoodin puolella. Pääkehittäjä, Rasmus Lerdorf loi kielen 1994 helpottaakseen oman verkkosivustonsa ylläpitoa. Vuotta myöhemmin hän julkaisi kielen vapaaseen käyttöön GPL-lisenssin alaisena. [3.] PHP:n nykyinen pääversionumero on viisi, ja sen tulkki on kirjoitettu uusiksi melkein jokaisessa pääversiopäivityksessä [4]. Perussyntaksi on sen sijaan pysynyt suhteellisen samanlaisena koko kehityskaaren ajan.

PHP on alustariippumaton, ja se on käännetty monille eri järjestelmille, mikä mahdollistaa laajan ajoympäristötuen. Normaalikäytössä, verkkosivujen taustamoottorina, PHP-ohjelma tarvitsee PHP-tulkin lisäksi HTTP-palvelimen. Lisäksi kaikki vähänkään järeämpää tiedontallennusta vaativat ohjelmat käyttävät jonkinlaista tietokantasovellusta, koska oman, laadukkaan tallennusratkaisun kehittäminen ei ole huonon hyöty-aikasuhteen takia perusteltua. Yleensä ylläpitäjät päätyvät Apache-HTTP

-palvelimeen sekä MySQL-tietokantaan**.** Molemmat ovat avoimia ja ilmaisia. Tämä ohjelmistojen muodostama kokonaisuus on niin yleinen, että esimerkiksi useimmissa Linux-jakeluissa ohjelmien paketit asennetaan automaattisesti, kunhan tietokoneen roolin määrittelee WWW-palvelimeksi. Windows-järjestelmille on puolestaan olemassa useita erilaisia verkko-ohjelmistokehityspaketteja, kuten WAMP (Windows, Apache, MySQL, PHP), jotka asentavat edellä mainitut palvelinohjelmistot sekä erillisen hallintaohjelmiston.

Kielellisesti PHP on helppo omaksua useista syistä. Ensinnäkin muuttujien tyypitys on heikkoa eli vapaata. Tämä tarkoittaa käytännössä sitä, että muuttujan tyyppi määräytyy lennosta sen mukaan, minkälaista dataa se sisältää. Heikon tyypityksen etuna on se, että erityyppisiä muuttujia voidaan huolettomasti yhdistellä tai esimerkiksi tallentaa samaan taulukkoon. Toinen PHP:n etu on laaja funktiovalikoima**,** jonka käyttöönotto ei vaadi mitään lisäkirjastoja. Etenkin nopeampien kielten kanssa uuden ohjelman kehittäminen aloitetaan kirjoittamalla itse tarvittavat perusrutiinit, mikä luonnollisesti hidastaa ja hankaloittaa varsinaisen ohjelmalogiikan kehittämistä. PHP:n tapauksessa tämä esiohjelmointivaihe voidaan suoraan sivuuttaa.

Lisäksi PHP:lle on tarjolla kielen kehittäjien ylläpitämä kattava digitaalinen ohjekirja sekä verkkopalveluna toteutettu funktioreferenssi suorine esimerkkeineen. Isompaan avuntarpeeseen vastaa PHP:n ympärille kehittynyt laaja ja aktiivinen yhteisö, joka tarjoaa vertaistuen lisäksi esimerkkejä, oppaita, valmiita ohjelmakomponentteja sekä muuta lisämateriaalia. [5.]

Kaikkien mielestä PHP ei kuitenkaan ansaitse saamaansa suosiota. [6;7;8;9;10;11;12] Koko kieli on ollut alun perinkin pelkkä nopeasti kirjoitettu pienen projektin apuväline eikä sen perusfilosofiaa ole missään vaiheessa mietitty uudestaan [13]. Ominaisuuksia on toki lisätty, mutta nekin yleensä reilusti jälkijunassa, suuren vaatimusmyrskyn saattelemana. Tässä mielessä moni raskaan sarjan kieli on huomattavasti PHP:tä edellä.

Suurin ongelma ei kuitenkaan piile itse kielessä, vaan sen käyttäjissä. PHP on erinomainen kieli aloittelevalle ohjelmoijalle: syntaksi on samanlaista kuin muissa

yleisesti käytetyissä ohjelmointikielissä, mutta hitusen helpompaa. Muuttujien käsittely on huomattavasti vahvan tyypityksen kieliä yksinkertaisempaa ja perusoperaatioihin on olemassa valmiit toteutukset.

Helppo alku takaa, että jo perustaidoilla saa nopeasti kehitettyä vähintäänkin toimivia konseptidemoja sekä pienimuotoisia projekteja. Sen sijaan se ei opeta nöyryyttä ohjelmointia ja ohjelmistokehitystä kohtaan. Monille PHP-kehittäjille uusien ohjelmointikielien opettelemisessa nousee nopeasti seinä vastaan, kun uusi kieli ei ole yhtä miellyttävä ja anteeksiantava kuin PHP. Voidaan todeta, että pelkästään PHP:n ja sen yhteisön parissa kehittynyt ohjelmoija kärsii samanlaisista ongelmista kuin kieli itse: perusasiat ovat kunnossa ja niiden avulla voi kehittyä suhteellisen pitkälle, mutta tarvitaan reilusti ulkoa tulevaa painostusta, jotta kehitys saadaan nostettua kokonaan uudelle tasolle, kuten siirryttäessä proseduraalisesta ohjelmoinnista olio-ohjelmointiin.

Näistä kehittäjistä tulee ennen pitkää osa yhteisön tukirankaa, ja huonot ohjelmointitavat periytyvät seuraaville ohjelmoijasukupolville. Ongelma on todellinen, sillä vaikka PHP-tulkki itsessään on varsin turvallinen [14;15;16], PHP:llä kirjoitetut ohjelmistot sijoittuvat haavoittuvuustilastojen kärkipäähän [3;17]. Tilannetta vielä pahentaa se tosiseikka, että PHP-palvelut sijaitsevat yleensä Internetiin kytketyillä palvelimilla, joten palveluiden aukkojen etsiminen ja hyödyntäminen on helppoa ja nopeaa. Etenkin vialliset, mutta laajasti käytetyt järjestelmät, kuten useimmat PHP-pohjaiset keskustelualueet ovat jatkuvasti hyökkäysten kohteena [18;19;20;21;22]. Myös yritykset ovat saaneet tästä osansa, sillä kiireen ja kustannustehokkuuden nimissä näitä ilmaispalveluita on otettu käyttöön ilman minkäänlaista tietoturvaselvitystä.

2.2 Drupal-sisällönhallintajärjestelmä

PHP:llä on kuitenkin mahdollista tehdä myös turvallista koodia, kunhan noudattaa hyvän ohjelmointitavan periaatteita. Yksi tärkeimmistä ohjenuorista on välttää toteutusten monistamista: pyörää ei kannata keksiä uudelleen. Suuremmassa mittakaavassa se pätee myös muiden tekemään koodiin – jos joku toinen on keskittynyt yhden osakokonaisuuden hiomiseen ja toteutuksen ulkoiset ominaisuudet (lisenssi,

saatavuus, ylläpidettävyys) ovat kunnossa, komponentti kannattaa sisällyttää omaan ohjelmistoon itse toteutetun sijaan.

Tätä silmällä pitäen PHP:lle on kehitetty joukko kolmannen osapuolen ohjelmistorunkoja (engl. framework) ja muita valmisratkaisuja, kuten sisällönhallintajärjestelmät. Ohjelmistorungon tarkoitus on tarjota kattava komponenttikirjasto, jolloin ohjelmoijan tarvitsee keskittyä ainoastaan uuden ydinlogiikan kehittämiseen. Myös perusjärjestelmän ohjelmoiminen jää kehittäjän vastuulle.

Sisällönhallintajärjestelmä (engl. CMS, Content Management System) sen sijaan toimii perustarkoituksessaan suoraan. Nimensä mukaisesti sen päätehtävä on mahdollistaa erilaisen käyttäjäsisällön lisääminen sivustolle. Järjestelmät sisältävät yleensä sisäisen editorin uusien sivujen luomista varten, käyttäjähallinnan sekä työkalut sisältömateriaalin, kuten kuva- ja liitetiedostojen ylläpitoon sekä lisäämiseen.

Asiakasprojektin pohjaksi oli valittu valmis, avoimen koodin sisällönhallintajärjestelmä, Drupal, joka on tällä hetkellä ilmaisista sisällönhallintajärjestelmistä paras ja kattavin [23]. Drupalin perusfilosofia on modulaarisuus – järjestelmää laajennetaan ja räätälöidään erilaisilla moduuleilla, joita on saatavilla runsaasti. Myös omien moduulien kehittäminen onnistuu kivuttomasti. Drupal tarvitsee toimiakseen PHP-tulkin, Apache- tai IIS- HTTP-palvelimen sekä MySQL- tai PostgreSQL-tietokantapalvelimen. Versiovaatimukset ovat hieman tavallista PHP-järjestelmää vaativammat, mutta tarvittavat ohjelmistot ovat saatavissa useille eri käyttöjärjestelmille (Windows, Mac OS X, Linux, FreeBSD, OpenBSD, Solaris 10, OpenSolaris), joten Drupalia voidaan ajaa kaikesta huolimatta hyvin laajalla järjestelmäpohjalla. Kuvassa 3 on esitetty Drupalin etusivu.

Kuva 3. Drupal-sisällönhallintajärjestelmän etusivu.

Drupalin historia on samanlainen kuin PHP:llä: alussa oli yksi pääkehittäjä, Dries Buytaert ja tuoreen ohjelmiston päätehtävänä oli helpottaa oman sivuston ylläpitoa. Drupalin esiaste aloitti BBS-järjestelmänä ja kehittyi ajan saatossa nykyaikaiseksi sisällönhallintajärjestelmäksi. Kuten PHP:n kanssa kävi, myös Drupalista tehtiin (vuonna 2001) nopeasti opensource-projekti. Nykyisin sitä levitetään GPL-lisenssin alaisuudessa. [32.]

Drupalin yhdistetty käyttäjä- ja kehittäjämäärä on ylittänyt sen kriittisen massan, joka tarvitaan jatkuvan kehityksen ja tuen ylläpitämiseksi. Yritysympäristössä, jossa järjestelmien pitää toimia katkoitta koko käyttöikänsä ajan, tämä on erittäin tärkeä kriteeri. Normaalin avoimen lähdekoodin periaatteen mukaan tukea voi myös ostaa kattoyrityksiltä, jotka ylläpitävät hallinnoimansa ohjelmiston peruspakettia ja ratkovat maksusta ilmenneet haasteet, kuten Novell SUSE Linuxin tapauksessa. Vastaavaa tukea Drupalille tarjoaa Aquia-niminen yritys. [33.]

Yksi jatkuvan toiminnan perusperiaatteista on järjestelmän eri komponenttien pitäminen

ajan tasalla. Drupal valvoo itsenäisesti sekä ytimen että asennettujen moduulien päivitystilannetta ja ilmoittaa, kun uusia versioita on saatavilla. Tietoturvasyistä uudet versiot on asennettava käsin, mutta se tapahtuu yksinkertaisesti purkamalla ladattu paketti vanhan version päälle ja ajamalla sisäinen päivityskäsky. Päivitystilanteessa vanhan järjestelmän sisältämän informaation pitää luonnollisesti siirtyä ongelmitta uuteen. Moni, huonommin suunniteltu järjestelmä jättää käyttäjänsä tässä kohdassa pulaan, jolloin joudutaan elämään pitkän aikaa kahden eri version välisessä loukussa. Tämän estämiseksi Drupal ja sen moduulit sisältävät erilliset tietojenpäivitysrutiinit, jolloin päivityksessä tapahtuva tietokantamallin muutos ei tuhoa vanhan informaation sisäistä tietorakennetta.

Päivitysideologia saa risuja ainoastaan Drupalin pääversion päivittämisestä. Uusi pääversio sisältää aina rajapintamuutoksia, mikä on välttämätöntä, jotta järjestelmä pääsee kehittymään ilman vanhojen versioiden tukemisen aiheuttamaa painolastia. Tämä kuitenkin aiheuttaa isoja ongelmia, sillä rajapintamuutokset ovat yleensä radikaaleja ja vaikuttavat melkein kaikkiin moduuleihin. Useat moduulit vaativat vielä toimiakseen muita moduuleita, joten etenkin riippuvuusketjun loppupäässä olevien moduulien päivittyminen hidastuu huomattavasti.

Ongelmien kokoluokkaa on saatu pienennettyä parantamalla rajapintamuutosten dokumentaatiota ja siirtämällä tärkeimpiä moduuleita Drupalin ytimeen, jolloin pääversion päivittyessä elintärkeimmistä moduuleista julkaistaan samalla uudet versiot. Hyvistä yrityksistä huolimatta Drupal-järjestelmäylläpitäjät pitäytyvät mielellään vanhassa pääversiossa, mikä vaikuttaa noidankehämäisesti uuden pääversion kehittämiseen. Useimpia moduuleja kehitetään vielä rintarinnan molemmille pääversioille, mikä vastaavasti hidastaa moduulikehitystä entisestään.

2.3 Drupal-moduulien kehittäminen

Drupalissa oman moduulin – joka oli myös tiedonsiirtotoiminnallisuuden pohjalla – kehittäminen alkaa siitä, että johonkin Drupalin moduulihakemistoista luodaan uusi alihakemisto, jolle annetaan moduulin koneellisesti luettava nimi (engl. machine

readable name). Koneellisesti luettava nimi kulkee järjestelmän sisällä moduulin identifikaationa, joten sen nimeämiselle on asetettu tiettyjä rajoituksia perustuen lähinnä käytetyn ohjelmointikielen, eli PHP:n, rajoituksiin. Luonnollisesti järjestelmän sisällä ei myöskään voi olla useampaa samannimistä moduulia.

Moduulin omaan hakemistoon luodaan seuraavaksi kaksi tiedostoa: ’<`moduulin nimi`>.info’ sekä ’<`moduulin nimi`>.module’. Info-tiedostoon kirjoitetaan yleistietoa moduulista: käyttäjälle näytettävä moduulin nimi, lyhyt selkokielinen kuvaus, versionumero, ulkoiset riippuvuussuhteet jne. Tietoja käytetään hallinnassa lisätarkenteina sekä yleisesti tarvittavien oheismoduulien tarkistamisessa. Module-tiedosto puolestaan sisältää moduulin ytimen, johon Drupal-rajapinnan toteutukset sijoitetaan. Koodipohjaa voi tarpeen tullen myös hajauttaa useampaan tiedostoon, mikä on suositeltavaa etenkin isojen moduulien tapauksessa.

Drupal on lähtökohtaisesti funktionaalinen järjestelmä, ja sen ohjelmistorajapinnat toteuttavat samaa ideologiaa. Moduulit kytketään järjestelmään niin sanottuja koukku-funktiota (engl. hooks) käyttämällä. Koukut toimivat samalla tavalla kuin tapahtumapohjainen viestintä olio-ohjelmoinnissa: tietty tapahtuma laukaisee kyseiseen tapahtumaan liittyvät koukkufunktiot, jolloin koukut voivat vaikuttaa tapahtuman suorittamiseen. Esimerkiksi tiedonsiirtomoduulin tapauksessa tarvittiin hallintapuolelle uusi valikko, josta tiedon tuontia ja vientiä voitiin hallita, joten moduulille oli lisättävä uusi menu-koukku. Kun hallintavalikkoa ollaan luomassa, kutsutaan kaikkia moduuleilla olevia menu-koukkuja, jotka palauttavat oman valikkonsa tietorakenteen. Olioiden sijasta Drupal käyttää valikoiden määrittelemiseen sisäkkäisiä taulukoita, joiden perusrakenne on sisällytetty valmiiksi järjestelmään.

Erilaisia koukkuja on olemassa runsaasti, ja niiden hyödyntäminen on vapaaehtoista, joten käytössä oleva rajapinta on laaja eikä pakota moduuleja toimimaan minkään ennalta määrätyn muotin mukaisesti. Laajennusvaraa tarvitaan, sillä Drupalin ydinjärjestelmä on kehitetty toimimaan ainoastaan perusalustana, jonka toiminnallisuutta moduulit laajentavat. Näin järjestelmästä karsiutuvat automaattisesti haluttuun käyttötarkoitukseen nähden ylimääräiset osat ja toisaalta puuttuvien

toiminnallisuuksien lisääminen on helppoa.

2.4 Kuvauskielen valitseminen

Järjestelmästä tuodulle informaatiolle tarvittiin yleinen esitysmuoto, jota kautta varsinainen tietosisältö voitaisiin siirtää muihin järjestelmiin. Käytännössä sen tehtävänä oli siis toimia eräänlaisena välivarastona, jota muut järjestelmät pystyisivät ymmärtämään joko suoraan omien valmisominaisuuksiensa avulla tai erillisen lisäkehityksen jälkeen. Tämä asetti vaatimuksia esitysmuodon olemukselle. Tärkein periaatepäätös tehtiin rajaamalla pois erilliset tallennusjärjestelmät, kuten tietokannat ja muut, suuria lisäkirjastoja tai -ohjelmistoja vaativat tallennusmuodot.

Nopeudestaan ja tiedon kuvaamisen tarkkuudestaan huolimatta niiden käyttö ei ollut perustelua – yleiskäyttöön tarkoitettu väliaikainen esitysmuoto ei voi vaatia toimiakseen isoja resursseja. Järjestelmillä, johon informaatiota ollaan siirtämässä, on yleensä oma, optimoitu tapa tallentaa informaatio sisäiseen käyttöön. Jos siirrettävää informaatiota on paljon ja siirtoja ajetaan usein, on kannattavampaa muuntaa siirrettävä informaatio suoraan lähtöjärjestelmästä kohdejärjestelmään ilman välivarastointia.

Varsinaisten tallennusjärjestelmien poisrajaamisen jälkeen jäljelle jäivät erilaiset kuvauskielet, joista piti valita sopivin. Kuvauskieliä ei varsinaisesti suoriteta kuten ohjelmointikieliä, vaan niitä luetaan ja informaatio käännetään käytetyn ohjelmointikielen tai järjestelmän sisäisiksi tietorakenteiksi. Tämä määritelmä pitää luonnollisesti sisällään sen ehdon, että kuvauskielen on oltava koneellisesti luettavissa. Eroja eri kuvauskielten välille saadaan vertailemalla muita ominaisuuksia: kuinka laajasti se on tuettu eri ohjelmointikielissä ja valmisjärjestelmissä, miten tarkasti tietorakenteita voidaan kuvata, kuinka helppolukuista kieli on ihmiselle ja niin edelleen. Kuvauskielistä valittiin vertailuun kolme nykypäivänä yleisesti käytettyä kieltä: JSON, YAML ja XML [24].

2.4.1 JSON

JSON (JavaScript Object Notation) on ryhmän nouseva tähti. Web 2.0 -aikakaudella ydinlogiikka pidetään palvelinpäässä, mutta käyttäjälle voidaan tarjota interaktiivisempaa sisältöä käyttämällä selaimessa tulkattavaa JavaScriptiä, jolloin selain ja palvelin voivat siirtää tietoa molempiin suuntiin ilman, että varsinaista sivua tarvitsee päivittää. JSON perustuu suoraan JavaScriptin tietorakenteeseen, jolloin palvelimen JSON-muodossa lähettämä informaatio on selaimen käytettävissä sellaisenaan. Syntaksi on helppolukuista myös ihmiselle. Helppokäyttöisyydestään johtuen sitä on alettu käyttää myös JavaScriptiin perustumattomien järjestelmien välisessä tiedonsiirrossa.

Koska JSON on suoraan JavaScript-koodia, se sisältää kaikki ohjelmoinnin perustietotyypit, mukaan lukien taulukot sekä objektit. Tästä johtuen informaation rakennetta ei tarvitse laajentaa yksinkertaisen perustietotyyppituen takia, kuten esimerkiksi XML:n tapauksessa.

```
{
     "muuttuja1": "muuttujan1 arvo",
     "muuttuja2": "muuttujan2 arvo",
     "muuttuja3": "muuttujan3 arvo",

     "taulukko": [
         "taulukon arvo 1",
         "taulukon arvo 2",
         "taulukon arvo 3",
     ]
     "olio": {
         "olioMuuttuja1": "olioMuuttujan1 arvo",
         "olioMuuttuja2": "olioMuuttujan2 arvo",
         "olioMuuttuja3": "olioMuuttujan3 arvo",
         "olioMuuttuja4": 12345
     },
 }
```

Kuva 4. Esimerkki JSON-kielen syntaksista.

Kuvassa 4 esitellään JSON:n syntaksia. Esitetyn informaation ympärillä olevat aaltosulkeet tarkoittavat, että kuvan informaatio edustaa yhtä oliota. Tekstimuotoinen informaatio, myös muuttujien nimet, eristetään lainausmerkkien väliin. Muuttujan nimen ja arvon välistä riippuvuutta kuvataan kaksoispisteellä. Muuttujat erotetaan toisistaan pilkulla. Taulukon esittämiseen käytetään hakasulkuja, joiden väliin taulukon arvot sijoitellaan. Avain-arvotaulukoille ei ole erillistä määritelmää JSON-määrityksessä, sillä ne esitetään olioina. Muuttujat sijoitetaan arvoineen aaltosulkujen väliin kuvassa esiintyvän ’olio’-esimerkin mukaisesti [25].

Koska JSON on JavaScript-koodia, sen voi tulkata JavaScript-ohjelmassa eval() -funktiolla. Tähän sisältyy kuitenkin tietoturvariski, sillä JSON-muotoiseen informaatioon voidaan sisällyttää JavaScript-komentoja, jotka ajetaan tulkkausvaiheessa. Tilanteen korjaamiseksi suositellaan käyttämään erillistä JSON-tulkkia myös JavaScriptin tapauksessa, jolloin menetetään suora JavaScript-yhteensopivuus, mutta ohjelmia ei voi murtaa vihamielisellä syötteellä. Yleisen siirtomuodon tapauksessa tilanteeseen joudutaan puuttumaan ainoastaan siinä tapauksessa, että kohdejärjestelmä on JavaScript-pohjainen.

2.4.2 YAML

YAML (rekursiivinen akronyymi, YAML Ain't a Markup Language) on JSON:n tapaan tietorakenteiden kuvauskieli. Kielet ovat erittäin samankaltaisia - rakenteellisesti JSON on YAML 1.2:n osajoukko ja YAML-tulkit lukevat suoraan suurinta osaa JSON-dokumenteista [26]. YAML on myös kehitetty varta vasten ihmisten luettavaksi. Tarjolla on muitakin erikoisominaisuuksia, kuten identtisen informaation esittäminen osoittimilla monistamisen sijaan.

Lisäksi YAML ei voi sisältää tulkkauksen aikana ajettavia komentoja toisin kuin suoraan käytettävä JSON, mikä on tärkeä ominaisuus turvallisuuden kannalta. Koska yleiskäyttöinen esitysmuoto ei tarvitse suoraa JavaScript-tukea, YAML korvaa kattavasti JSON:n. JSON:n ainoa etu YAML:ään nähden on sisäinen informaation oikeellisuuden tarkistaminen (engl. validation). Tähän ei kuitenkaan ole suurta tarvetta,

sillä täysin turvallinen ohjelma ei voi luottaa ulkoa tulevaan syötteeseen eikä täten sen mukana saapuvaan vahvistusinformaatioonkaan.

Kuvassa 5 esitetään YAML-syntaksi perusmuodossaan. YAML-dokumentti alkaa kolmella viilalla ’---’ ja lopetetaan kolmeen pisteeseen ’…’. Mikäli YAML-muotoista dataa otetaan vastaan virtana, voidaan lähettää useita dokumentteja sijoittamalla aloitus- ja lopetusmerkit dokumenttien väliin. YAML:n syntaksissa on kaksi tapaa merkitä elementtejä: sisennetty ja suljettu. Suljettu merkintätapa toimii täsmälleen samalla tavalla kuin JSON, mikä selittää, miten JSON voidaan lukea YAML 1.2 -standardin osajoukoksi. YAML-määritys kuitenkin suosittelee käyttämään ennemmin sisennettyä esitystapaa, joka on ihmisen kannalta luettavampaa. Suljettua merkintätapaa suositellaan ainoastaan tilanteisiin, jossa sen ytimekkyydellä voidaan parantaa luettavuutta, kuten useiden sisäkkäisten elementtien tapauksessa.

Sisennetyn merkinnän tapauksessa elementtien välinen hierarkia esitetään välilyönneillä sisentämällä. Määritys ei ota kantaa tarvittavien välilyöntien määrään. Riittää että samalla tasolla olevat elementit on sisennetty samalla tavalla ja vastaavasti eri tasolle kuuluvat elementit ovat selkeästi omalla tasollaan. Merkintätapojen erot selvenevät parhaiten kuvan 5 ’suljettuja taulukoita’- sekä ’sisennettyjä taulukoita’ -kohtia vertaamalla. Tämä merkintätavan lisäksi YAML-määritykseen on sisällytetty muita, inhimillistä lukijaa helpottavia seikkoja. Sekä muuttujien nimet että arvot voidaan kirjoittaa ilman ylimääräisiä heittomerkkejä. Muuttujan nimi ja arvo erotetaan toisistaan JSON:n tapaan kaksoispisteellä. Merkkijonoja ei tarvitse erikseen lopettaa, vaikka ne olisivatkin usealla rivillä. Tässä tapauksessa kielen tulkki määrittää merkkijonon alkamis- ja loppumiskohdan sisennysten perusteella. YAML tarjoaa eri tulkkausvaihtoehtoja usealle riville jaoteltujen merkkijonojen rivinvaihdoille, jolloin luettavuus ei aiheuta ongelmia informaation käytettävyydelle. Esimerkkejä kuvan 5 kohdassa ’muu syntaksi’.

Inhimillisen luettavuuden lisäksi YAML sisältää käyttökelpoisia ominaisuuksia myös koneellista lukua varten. Kaikki JSON:n sisältämät tietotyypit ovat luonnollisesti käytössä sillä erotuksella, että oliot ja avain-arvotaulukot on jaoteltu erillisiksi

kokonaisuuksiksi. Ero on molempien kielien osalta perusteltu: JSON on tehty mahdollisimman kevyeksi ja yksinkertaiseksi; YAML on kehitetty yleispäteväksi ratkaisuksi informaation välittämiseen, jolloin sen on oltava jo syntaksiltaan semanttisesti moitteeton.

YAML-määritys ei itsessään sisällä mitään suositusta datan pakkaamisesta, jolloin se on aina ihmisen luettavissa. Tekstimuotoinen data vie kuitenkin huomattavasti enemmän tilaa binäärimuotoiseen dataan verrattuna. Pakkaamisen sijaan YAML sisältää tuen ankkureille, joilla toistuvat elementit voidaan yksinkertaistaa, mistä on hyötyä sekä koneellisesti että inhimillisesti luettuna. Esimerkki ankkureiden käytöstä löytyy kuvasta 5, kohdasta ’ankkuriesittely’. Tila1 määritellään lähteeksi luomalla siitä ankkuri. Vastaavasti tila2 määritellään ankkurin kohteeksi, jolloin se perii tila1:n elementit, uusio ja uusio2. Tila2 kuitenkin sisältää jo elementin uusio2, jolloin peritty arvo ylikirjoitetaan. Tämä johtaa lopputulokseen, jossa tila2:lla on kaksi alielementtiä: uusio sekä uusio2, joiden arvot ovat peritty ’3’ sekä ylikirjoitettu ’a’.

```
---
suljettuja taulukoita:
    - {muuttuja: arvo, muuttuja2: arvo2}
    - lista: [riisi, ohra, vehnä]

sisennettyjä taulukoita:
    - muuttuja1.1: abcd
      muuttuja1.2: efgh

    - muuttuja2.1: 1234
      muuttuja2.2: 5678

ankkuriesittely:
    - tila1:  &ankkuri
        uusio:  3
        uusio2: 5

    - tila2:
        <<: *ankkuri
        uusio2: a

muu syntaksi:
    - olio:  !luokka { data: 3, data2: 4}
    - !!str 52525
    --- |
      Pitkä teksti
      usealla rivillä.
      Merkintä säilyttää rivinvaihdot.
    --- >
      Pitkä teksti
      usealla rivillä.
      Merkintä ei säilytä rivinvaihtoja.

...
```

Kuva 5. Esimerkki YAML-kielen syntaksista.

2.4.3 XML

Viimeinen vertailluista kuvauskielistä, XML (Extensible Markup Language), lähtee täysin eri lähtökohdista. XML on kuvauskieli muiden kuvauskielien määrittelemiseen. Sen on suunnitellut ja toteuttanut W3C, joka vastaa yleisesti virallisista Internet-määrityksistä, joten kieli on ollut syntyhetkestään asti standardi. [27.] Tästä syystä XML on erittäin käytetty informaationesitysmuoto myös yritysmaailmassa, joka suhtautuu – osittain aiheesta – epäilevästi lupaaviin, mutta epävirallisiin tekniikoihin. Lisäksi sitä käytetään pohjana useissa muissa standardeissa: erilaisissa tiedostomuodoissa (SVG-vektorigrafiikkatiedostot, OpenOffice-dokumentit), muissa verkkostandardeissa (XHTML), palvelujen välisessä tiedonsiirrossa (XML-RPC,

SOAP) ja niin edelleen. Sen sijaan toisin kuin yleensä luullaan, HTML-kieli ei perustu XML:ään vaan sen esiasteeseen, SGML:ään.

XML on SGML-osajoukko ja kehitetty korvaamaan alkuperäinen SGML, jolloin kielen tulkkien kehittäminen helpottui huomattavasti [28]. Mammutin korvaajasta on ajan saatossa kasvanut vastaavanlainen kankea jättiläinen, jota vastaan muut yleiset kuvauskielet kilpailevat. Historiansa puolesta XML ei ole koskaan pyrkinytkään vastaamaan muihin haasteisiin kuin koneellisen lukemisen helpottamiseen – muilla osa-alueilla nojataan suoraan SGML:n ominaisuuksiin. XML ei sisällä muita tietorakenteita kuin sisäkkäin järjestettävät elementit, jotka voivat sisältää tietoa ja erilaisia attribuutteja. Yksinkertaisuudessaan tämä on riittävä tarkkuus kuvaamaan minkälaista informaatiota hyvänsä, mutta useissa tapauksissa ohjelmointikielten sisältämät tietorakenteet selkeyttäisivät informaation kuvaamista huomattavasti.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<esimerkkipuu>
   <elementti>
      elementin data
   </elementti>


   <elementti attribuutti=”attribuutin arvo”>
      <elementti>
         elementin data
      </elementti>

      <tyyppi2>
          elementin data
      </tyyppi2>
   </elementti>
</esimerkkipuu>
```

Kuva 6. XML- esimerkki.

XML-spesifikaation mukaan jokaisen luettavan XML-dokumentin rakenteen pitää olla oikein muotoiltu (engl. well-formed), eli se ei saa sisältää kuvauskielen kannalta virheellistä merkkausta ja spesifikaatioon määrättyjen ehtojen on täytyttävä. Jos ehdot eivät täyty, spesifikaationmukaisen tulkin on kieltäydyttävä lukemasta dokumenttia. Kuvassa 6 on esitetty esimerkki XML-dokumentista. Jokaisen XML-dokumentin pitää sisältää heti ensimmäisellä rivillä XML-määritys, joka kertoo käytettävän XML-

määrityksen version sekä merkistökoodauksen. XML-dokumentin rakenne koostuu elementeistä, jotka on koottu yhden puurakenteen sisään. Juuria, jota kuvassa edustaa ’esimerkkipuu’-elementti, ei saa olla enempää kuin yksi.

Elementit määritellään aloitus- ja lopetustageilla. Aloitustagi voi sisältää yhden tai useampia attribuutteja arvoineen. Itse elementti voi sisältää datan lisäksi sisäkkäisiä elementtejä, jotka muodostavat XML:n puurakenteen. Esimerkinmukainen elementtien sisentäminen ja erottelu toisistaan ei ole välttämätöntä. XML:n vaatima dokumenttien rakenteen oikeellisuus mahdollistaa sen, että kieli ei hajaudu omiin, epästandardeihin tulkkaustapoihin, kuten HTML:lle on käynyt. Toisaalta on ongelmallista, jos XML-dokumentti ei täytä määriteltyjä ehtoja ja tulkki kieltäytyy kokonaan lukemasta sitä. Joissain lukijoissa on erillinen tila, joka lukee dokumentin virheistä huolimatta. Näin elintärkeä tieto on mahdollista ottaa talteen, vaikka dokumentti olisi vahingoittunut tai muuten väärin muotoiltu.

Informaation rakenteen määrittelemiseksi XML sisältää kaksi eri vaihtoehtoa: DTD:n (Document Type Definition) [29] sekä XML Scheman / XSD:n (XML Schema Definition) [30]. Kyseisiä järjestelmiä käyttämällä voidaan kuvata dokumentissa esiintyvän informaation rakenne, jolloin XML-ohjelmisto voi tarkistaa, onko tallennettu informaatio oikeassa muodossa. DTD sisältyi jo SGML-kuvauskielen määrityksiin ja periytyi siitä eteenpäin XML:lle. DTD on XSD:tä helppokäyttöisempi – se voidaan esimerkiksi sisällyttää suoraan XML-dokumenttiin – mutta häviää XSD:lle ominaisuuksiensa puolesta. Isoimmat erot muodostuvat siitä, että DTD ei tue ollenkaan uusia XML:n ominaisuuksia, kuten nimiavaruuden määrittämistä eikä sillä voi kuvata tarkasti dokumentin rakennetta. [27.]

Kuva 7 sisältää DTD-esimerkkirakenteen: ’juuri’-elementin toteutuksen sekä DTD-määrittelyn, jota vastaan toteutuksen rakenne tarkistetaan. DTD-rakennemääritelmä sijoitetaan !DOCTYPE-elementin sisään. XML-standardista poiketen DTD-elementti saa muodostaa uuden juuren varsinaisen informaatiokuvauksen lisäksi. Esimerkissä ’juuri’-elementti määritellään sisältämään kaksi muuta elementtiä: ’lohko1’ sekä

’lohko2’. Lohkot on määritelty dataelementeiksi (#PCDATA), jolloin ne voivat sisältää varsinaista informaatiota.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE juuri [
  <!ELEMENT juuri (lohko1, lohko2)>
  <!ELEMENT lohko1  (#PCDATA)>
  <!ELEMENT lohko2  (#PCDATA)>
]>
<juuri>
    <lohko1>data1</lohko1>
    <lohko2>data2</lohko2>
</juuri>
```

Kuva 7. DTD-esimerkki.

XSD on W3C:n kehittämä määritelmä, jonka tarkoituksena on korvata DTD XML-dokumenttien rakenteiden kuvaamisessa. Toisin kuin DTD, XSD:n syntaksi on XML-pohjainen ja tässä mielessä edeltäjäänsä standardinmukaisempi. Tämä tarkoittaa myös sitä, että XSD-rakennemääritystä ei voi liittää suoraan tarkastettavaan XML-dokumenttiin, koska siinä tapauksessa itse määritys muodostaisi, DTD:n tavoin, ylimääräisen elementtipuun, mikä ei ole XML-standardin mukaista. Nimiavaruuksien lisäksi XSD:ssä voi suoraan määritellä XML-dokumentissa käytettävän elementin tietotyypin sekä sen, kuinka monta samanlaista elementtiä isäntä voi sisältää.

Kuvassa 8 on esitetty XSD-esimerkkidokumentti. Esimerkkidokumentti toimii tarkisteena elementille ’Kurssi’. Kurssi sisältää ’xs:complexType’-määritystagin, jolloin ’Kurssi’ saa sisältää muita elementtejä. ’xs:sequence’-tagi määrittelee, että alielementtien on oltava järjestyksessä ja koska se on esitetty ilman attribuuttimäärityksiä ’minOccurs’ ja ’maxOccurs’, alielementtien esiintymismäärää ei ole rajoitettu. Suurin osa alielementeistä (Nimi, Kuvaus, Opettaja, jne) on määritelty merkkijonoiksi antamalla attribuutille ’type’ arvoksi ’xs:string’. Lisäksi Nimi-elementin on esiinnyttävä vähintään kerran, koska sille on määritelty ’minOccurs’-attribuutti arvolla ’1’. ’Kurssi’-elementti saa tekstikenttien lisäksi sisältää erikoiskentän ’Opetuskieli’, joka on myös tekstimuotoinen, mutta se saa sisältää ainoastaan ’xs:restriction’-elementin määrittämät arvot ”FI”, ”EN” ja ”SE”.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Kurssi">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Nimi" type="xs:string" minOccurs="1"/>
        <xs:element name="Kuvaus" type="xs:string" />
        <xs:element name="Opettaja" type="xs:string" />
        <xs:element name="Luokkatila" type="xs:string" />
        <xs:element name="MuutaTietoa" type="xs:string" />
        <xs:element name="Opetuskielet">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="FI" />
              <xs:enumeration value="EN" />
              <xs:enumeration value="SE" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Kuva 8. XSD-esimerkki.

2.4.4 Valintaprosessi

Edellisten tietojen pohjalta lähdettiin valitsemaan sopivinta kuvauskieltä. Koska YAML tukee suoraan JSON:a, pelkän JSON:n käyttäminen ei ollut enää perusteltua ja se voitiin jättää vertailun ulkopuolelle. Valinta oli siis tehtävä YAML:n ja XML:n välillä. Molemmilla kielillä on hyvät ja huonot puolensa. YAML on uusi kieli, eikä sillä ole taakkanaan edellisiltä kieliltä perittyä ajattelutapaa. Lisäksi YAML:a ei ole suunniteltu yleiseksi standardiksi, jonka on sovittava kaikkiin tarkoituksiin. Sen sijaan se on kehitetty toimimaan mahdollisimman yksinkertaisesti erilaisten ohjelmointikielten ja niillä kirjoitettujen palveluiden rajapintakielenä, joka on helposti luettavissa niin koneen kuin ihmisenkin toimesta. Tätä varten on toteutettu kaikki ohjelmointikielten perustietotyypit, mutta syntaksia on yksinkertaistettu ihmislukijaa varten.

Vastaavasti XML on jo vakauttanut asemansa yleisenä kuvauskielenä. Se on monessa suhteessa raskastekoinen, jolloin se on ilmaisuvoimaltaan YAML:a heikompi, mutta samalla sen pohjimmaisena ideana on toimia yksinkertaisen kuvauskielen sijaan tekstimuotoisena tietokantana, johon sisältyy dokumenttien rakenteen oikeellisuuden

tarkistaminen sekä kattavat käännöstyökalut, kuten XSLT ja XSLT-FO, sekä osiontikieli Xpath. Tietokanta-ajatteluun sisältyy myös tarve säilöä tallennettua informaatiota pitkään. Tämän tarpeen täyttämiseen kuvauskieleltä edellytetään standardinmukaisuutta sekä tarpeeksi pitkää ja vakaata historiaa, jotta sen voidaan todeta kestävän aikaa.

Lopullinen päätös tehtiin kartoittamalla loppuasiakkaiden tarpeet ja selvittämällä, kumpi kuvauskielistä soveltuisi paremmin heillä jo käytössä olevien järjestelmien integroimiseen. Tässä vertailussa YAML hävisi selvästi ja takasi XML:n valinnan. Järjestelmästä yleisessä XML-muodossa tuotu informaatio ei kaikissa tapauksissa ole suoraan yhteensopiva muiden järjestelmien kanssa, mutta se toimii hyvänä pohjana järjestelmiä varten mukautetuille XML-malleille. Tulevaisuudessa saattaa ilmetä tarvetta myös YAML- tai JSON-muotoiselle informaatiolle, mikäli kielet kypsyvät epävirallisiksi standardeiksi. Etenkin JavaScript-ohjelmien tapauksessa on harkittava tarkkaan, mitä muotoa kannattaa käyttää. Tiedonsiirtomoduuli on toteutettu sillä periaatteella, että standardien vaihtuessa uusia siirtomuotoja voidaan ottaa käyttöön nopeasti.

3 Projektin toteutus

Tiedonsiirtomoduulin toteuttaminen oli osaprojekti suuremmassa asiakasprojektissa. Projektin tilaaja on työnantajani, Brain Alliance Oy:n, asiakas. Salassapitosopimus velvoittaa, että asiakkaan nimeä eikä pääjärjestelmän ydinlogiikkaa tai tarkkoja tietorakenteita saa esittää julkisesti, joten tietojen tarkkuutta on jouduttu näiltä osin rajoittamaan tässä dokumentissa. Sen sijaan on sallittua kertoa yleisesti järjestelmästä, toimintatavoista ja luonnollisesti itse moduulista.

Asiakasjärjestelmä on verkkopalvelu, joka on suunnattu yritys- ja instituutiokäyttöön. Sisältö on tekstimuotoista, ja sen luomisesta vastaavat käyttäjät, mutta tavallisesta ’Web 2.0’ -palvelusta poiketen keskitytään tiedon määrän sijaan sen laatuun. Tämä toteutuu osittain automaattisesti kohderyhmän valinnan kautta, joka rajaa käyttäjäjoukon pienempään ja koherentimpaan yhteisöön. Lisäksi yhteisön osana toimivat järjestelmän pääkäyttäjät kantavat suuremman vastuun prosessin ohjaamisessa.

Palvelun päästrategiana on tarjota perusjärjestelmä, jota räätälöidään loppuasiakkaiden toiveiden mukaisesti. Samalla toteutettuja lisäpalveluita voidaan käyttää suoraan tai pohjana uusien loppuasiakkaiden esittämien ominaisuuspyyntöjen toteutuksina. Hyvänä esimerkkinä tästä web-widget -komponentti, jolla loppuasiakas voi lisätä omille tai kumppaniensa sivuistoille minitoteutuksen järjestelmästä, jonka tarkoituksena on aktivoida käyttäjiä jo alkuvaiheessa osallistumaan sisällön luomiseen ja sitouttaa heidät varsinaiseen järjestelmään.

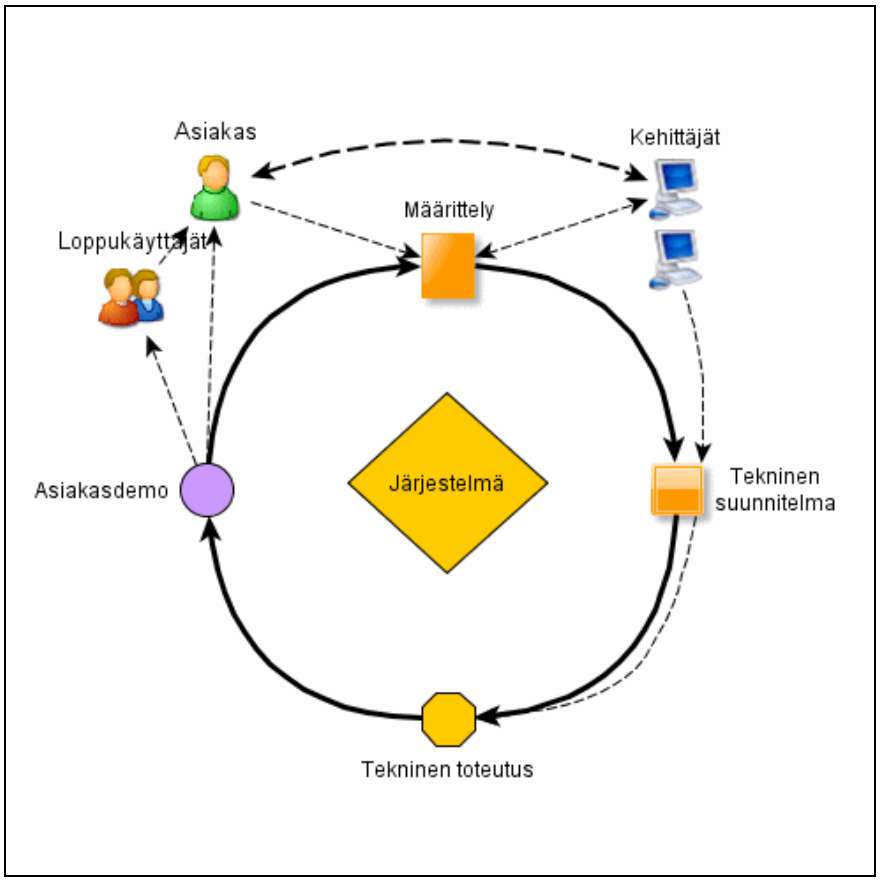
Lisäksi järjestelmään on pyritty lisäämään etukäteen valmiita lisäpalveluita erilaisten skenaarioiden esittelemiseksi loppuasiakkaille. Tiedonsiirtomoduuli edustaa tämänkaltaista lisäpalvelua. Järjestelmän osana sen tehtävä on tarjota kanava kerätyn informaation hyödyntämiseen ja jatkokehittämiseen, joka sisältö- ja käyttäjämäärien kasvattamisen sijaan lisää informaation arvoa ja mahdollistaa uusien käyttötapojen ja ajattelumallien yhdistämistä järjestelmään ulkoisten toiminnallisuuksien kautta.

3.1 Suunnitteluprosessi

Tiedonsiirtomoduulin suunnitteluprosessissa asiakas vastasi ideoinnista, jonka tuloksista hiottiin uusia ominaisuuksia. Valmiin määrittelyn jälkeen seurasi tekninen suunnittelu ja toteutus. Tekniikan osalta asiakas antoi vapaat kädet sillä perusedellytyksellä, että tehdyn toteutuksen taso pysyisi tarpeeksi korkealla. Laadunvalvonta suoritettiin asiakkaan kanssa yhdessä tapahtuvana testaamisena ulkoisen auditoinnin tai mittareiden sijaan.

Suunnittelun kannalta haasteellisinta oli, että itse pääasiakasprojektia kehitettiin tiedonsiirtomoduulin kanssa rinnakkain, joten moduuli jäi osittain tärkeämpien asioiden varjoon. Toinen vaikuttava tekijä oli se, että ei oltu täysin varmoja, mitä tarpeita ja vaatimuksia loppukäyttäjällä tulisi olemaan. Määritelmää tarkennettiinkin yleensä johtoryhmän ja asiakkaan omille asiakkailleen järjestämien demojen jälkeen. Tästä johtuen kehityksestä jäi puuttumaan selkeä linja, kuinka edetä ja mikä olisi tavoiteltu lopputulos.

Suunnittelu ja kehitys muuttuivat sykliseksi – moduulille ei annettu lopullista valmistumismääräaikaa eikä -kriteerejä, vaan määriteltiin ainoastaan seuraavan demoon tarvittavat lisäominaisuudet ja toteutettiin ne ennen varsinaista esityspäivää. Syklinen suunnittelutapa on mallinnettu kuvassa 9. Hyvänä puolena voidaan nähdä se, että teknisessä suunnittelussa piti ottaa heti alusta lähtien huomioon laajennettavuus ja luokkien yleisyys. Moduulin ydin ja peruslogiikka säilyivätkin lähes muuttumattomana koko projektin ajan. Lisäominaisuudet kehitettiin niiden ympärille omina luokkakokonaisuuksinaan.



Kuva 9. Syklinen suunnittelu- ja kehitysmalli.

Ensimmäinen kehitysvaihe oli ytimenä toimivan Drupal-moduulin luominen, jonka yhteyteen rakennettiin Drupal-tuontiluokka järjestelmän tietojen hakemista varten. Tämä oli ensimmäinen tekninen ”proof-of-concept”, jolla testattiin, että perusasiat olivat kunnossa. Seuraavaksi moduulille luotiin oma hallintatyökalu, jolla pystyi valitsemaan, mitkä tietorakenteet käyttäjä halusi viedä järjestelmästä ja missä muodossa. Koska tässä vaiheessa ei ollut vielä selvillä, minkälaisille järjestelmille viety data suunnattaisiin, päätettiin kehittää yleinen, XML-pohjainen siirtoformaatti.

Seuraavaksi asiakas näki tarpeelliseksi luoda raportteja järjestelmän tietorakenteista ja toiminnallisuus päätettiin yhdistää osaksi tiedonsiirtomoduulia. Pohjalla oli edelleen ajatus datan viemisestä ulos järjestelmästä, mutta generoituna suoran tietorakennesiirron sijaan. Tiukan käsitteellisesti ajatellen tämä ei kuuluisi tiedonsiirtomoduulin tehtäviin – raporttien luomiseen pitäisi olla oma työkalu, jonka lopputuloksia voisi siirtää muihin järjestelmiin tiedonsiirtomoduulin kautta. Kuitenkin reaalimaailmassa jo pelkkä

samanlaisten komponenttien tarve ja niiden eriyttämisestä syntyvät ongelmat riittivät takaamaan sen, että moduulin toimenkuvaa laajennettiin.

Raporttien käsittelystä haluttiin tehdä käyttäjän kannalta mahdollisimman helppoa ja standardinmukaista, joten lopputuloksena raporttien muodoksi valikoitui PDF. Formaatin suurimpana etuna on se, että dokumentit näyttävät kaikkialla samanlaisilta – myös tulosteissa, joita raporteista yleensä tehdään. Ainoa vaatimus on ilmainen lukijaohjelma, Adobe Acrobat Reader, joka on tiedostomuodon levinneisyydestä johtuen helposti saatavilla alustalle kuin alustalle.

PDF on hyvä formaatti valmiille raporteille, mutta lisäksi ilmeni tarvetta myös erilaisten tunnuslukujen hakemiseen järjestelmästä. Koska tunnuslukuja harvemmin esitetään tai tulostetaan suoraan, PDF:n käyttäminen ei ollut perusteltua. Tunnuslukujen yleisin jatkomuokkausväline on ehdottomasti taulukkolaskentaohjelma, jolla saaduista tunnusluvuista loppukäyttäjän on helppo valmistaa haluamiaan kaavioita ja tarkempia analyysejä laajempien raporttien pohjaksi.

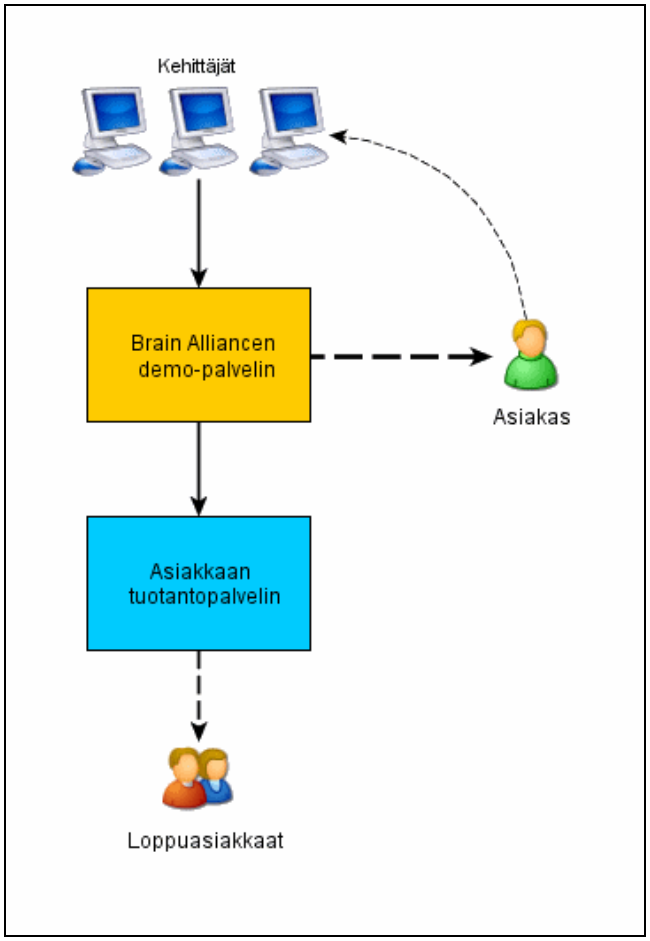
Valmiin Excel-muuntimen rakentaminen olisi vienyt liian pitkään, joten sen sijaan tiedot käännettiin Excelin ymmärtämään CSV-muotoon (Comma Separated Values), jossa rivi vastaa suoraan taulukkolaskentaohjelman riviä ja solut erotellaan tosistaan joko – kuten formaatin nimikin kertoo – pilkulla tai Excelin tapauksessa puolipisteellä. CSV-käännöksen tekeminen on helppoa, ja se on raportointipuolen lähin vastine tiedonsiirtomoduulin varsinaiselle käyttötavalle – pohjajärjestelmän tiedoista käännetään siirtoformaatti (CSV), joka lähetetään toiselle järjestelmälle (taulukkolaskentaohjelma) jatkokäsittelyä varten. Lopputulos voidaan sen jälkeen liittää kolmanteen järjestelmään, kuten tekstinkäsittelyohjelmaan, jolla luodaan esityskelpoinen raportti käsitellyn informaation ympärille.

3.2 Kehitysmenetelmät

Koko asiakasprojektin kehittämisen kanssa toimittiin siten, että kehittäjät pystyivät työskentelemään omillaan, mutta samalla tarjoten asiakkaalle mahdollisuuden tutustua

valmisteilla oleviin muutoksiin heille sopivana ajanhetkenä. Käytännössä tämä tarkoitti sitä, että jokaisella kehittäjällä oli tietokantaa lukuun ottamatta täydellinen kopio järjestelmästä omalla työkoneellaan. Kopiot pidettiin yhdenmukaisena versiohallinnan kautta. Versionhallintakonfliktit saatiin pidettyä minimissä jakamalla kehityskohteet osioittain – tiedonsiirtomoduuli yhtenä esimerkkinä – eri kehittäjille.

Asiakasta varten myös Brain Alliancen demopalvelimelle luotiin oma kopionsa järjestelmästä, joka päivitti itseensä automaattisesti versionhallintaan lähetetyt muutokset. Näin asiakas oli aina ajan hermolla ja pystyi tarvittaessa vaikuttamaan kehitykseen. Tämän lisäksi asiakkaalla oli vielä oma tuotantopalvelin, jonka järjestelmäkopiota päivitettiin hitaammalla tahdilla. Tällä tavalla mikään muutos ei koskaan päässyt testaamattomana suoraan tuotantoon. Asiakasprojektin kehitysprosessin toimintakaavio on esitetty kuvassa 10.



Kuva 10. Kehitysprosessin toimintakaavio.

Tiedonsiirtomoduulin tapauksessa noudatettiin pitkälti samaa kehitysparadigmaa. Tietokantaan ei tarvinnut tehdä mitään muutoksia, joten siirto versionhallinnan kautta toisiin järjestelmiin ei vaatinut muita perustoimenpiteitä. Ainoastaan kolmannen osapuolen HTML-PDF-kääntäjä piti konfiguroida jokaisen asennuksen yhteydessä, mutta sitä varten kääntäjässä oli sisäänrakennettu toiminnallisuus.

4 Asiakkaan Drupal-moduulin tekninen toteutus

Pääasiakasjärjestelmä oli rakennettu PHP-kielen ja Drupal-sisällönhallintajärjestelmän päälle, joten pohjaan ei voinut vaikuttaa ja tiedonsiirtomoduuli oli rakennettava tämän perusratkaisun yhteyteen. Asiakkaan hyväksynnän lisäksi tämä malli tarjosi suhteellisen laajat mahdollisuudet moduulin toteutukselle – PHP ei kielenä asettanut suuria rajoitteita, ja Drupalia varten piti toteuttaa ainoastaan erillinen hallintajärjestelmä, jolla moduulin ydinlogiikkaa saatiin ohjattua Drupalin ylläpitopuolelta.

Teoriassa olisi ollut mahdollista toteuttaa koko tiedonsiirtomoduulin toiminnallisuus jollain paremmin datan kääntämiseen sopivalla kielellä tai järjestelmällä, kunhan sen olisi yhdistänyt Drupalin kanssa, mutta hyödyn suhde käytettyyn aikaan olisi jäänyt erittäin pieneksi, joten idea jäi ajatustasolle. Raporttien osalta käännöstyössä joudutaan muutenkin ottamaan huomioon seikkoja, jotka tekevät suoran kääntämisen mahdottomaksi. Tässä mielessä PHP:n tapaisen yleiskielen käyttäminen on perusteltua.

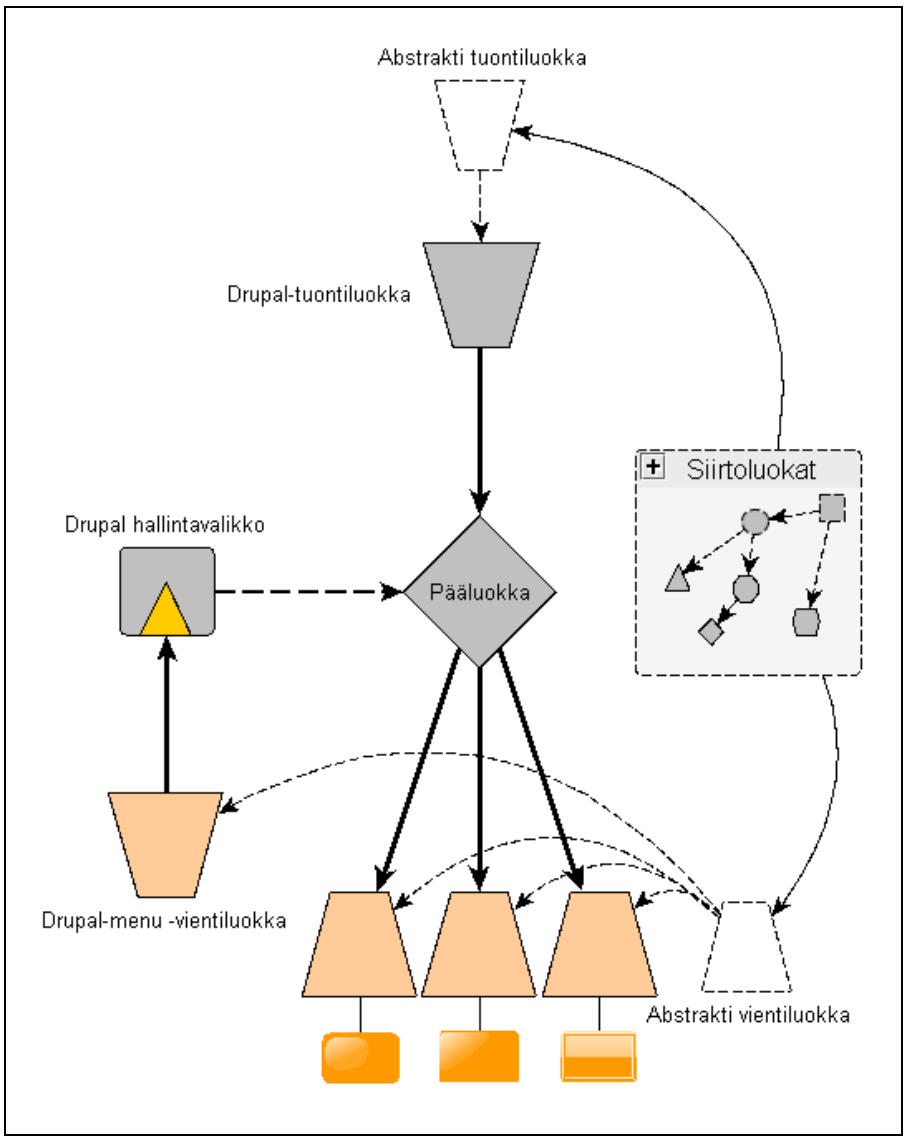
4.1 Tiedonsiirtomoduulin pohja

Tiedonsiirtomoduuliin ei tarvittu muita Drupal-koukkuja kuin oman hallintavalikon tulostaminen ja kontrollointi – loppuosa koodista käsittelee varsinaista ydinlogiikkaa. Tiedon vieminen ja tuominen ovat käytännössä saman kolikon kaksi puolta: molemmissa siirrettävä informaatio on luettava lähteestä ja sen jälkeen muutettava kohteen ymmärtämään muotoon. Oikeanlaisella suunnittelulla molemmat operaatiot oli mahdollista toteuttaa samalla pohjalogiikalla.

Drupal ei juurikaan hyödynnä olio-ohjelmoinnin tuomia etuja. Suurimpana syynä tähän on se, että Drupalin ydin perustuu PHP 4 -pohjaiseen ohjelmistoarkkitehtuuriin, joka ei vielä sisältänyt kunnollista oliotukea. Sen sijaan mikään ei estänyt toteuttamasta moduulia olio- ja PHP5-pohjaisena. Siispä kaikki mahdollinen, monessa paikassa käytettävä koodi, pakattiin erillisiin pääluokkiin, joita alaluokat perivät. Olioiden avulla ohjelmakoodia on helppo jakaa pienempiin osiin, mikä helpottaa koodin ymmärtämistä ja parantaa uudelleenkäytettävyyttä. Moduulin rakenne on esitetty kuvassa 11.

4.2 Pääluokka

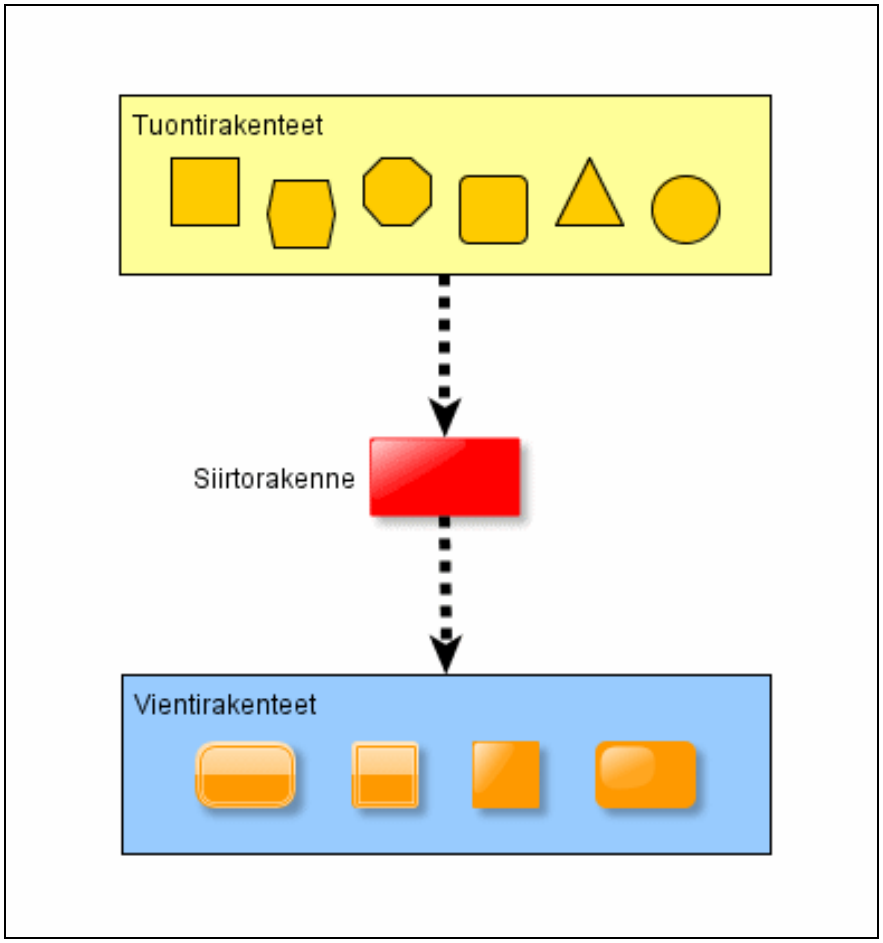
Ensimmäiseksi oli luotava operaation pääluokka, joka sisälsi koko prosessin päälogiikan. Informaation tuominen ja vieminen on hyvin suoraviivaista, joten pääluokasta tuli yksinkertainen: luokan instanssille annetaan tuonti- ja vientiobjekti sekä lisäksi lista siirrettävistä tietorakenteista. Tämän jälkeen tarvitaan ainoastaan yksi metodi, joka ohjaa informaation tuontiobjektilta vientiobjektille ja käskee vientiobjektin tuottamaan annetusta informaatiosta halutunlaista dataa.



Kuva 11. I/E-moduulin rakenne.

4.3 Sisäinen tietorakenne

Jotta informaation tuomisessa ja viemisessä olisi järkeä, lähdeinformaation on oltava eri muodossa kuin kohdeinformaatio, vaikka niiden esittämä tieto pysyykin samana. Prosessin dilemma on siinä, että jokaisen vientiluokan on ymmärrettävä jokaisen tuontiluokan tuottamaa informaatiota, jotta informaation muotoa voidaan muuttaa vapaasti. Ongelman ratkaisu on hyvin inhimillinen: eri kieliä puhuvat ihmiset käyttävät kommunikaatioon yhtä yleiskieltä, jota kaikki ymmärtävät. Tiedonsiirtomoduulin tapauksessa otettiin käyttöön yhteinen siirtorakenne, jota kaikki siirtoluokat pystyvät tulkitsemaan. Käytäntö on esitetty kuvassa 12. Kuvassa 11 esitetyssä moduulin kokonaisrakenteessa siirtorakenne on esitetty lohkona ’Siirtoluokat’.



Kuva 12. Yleinen siirtorakenne.

Ensimmäinen vaihtoehto oli Drupalin oma tietorakenne, jonka ympärille koko pääjärjestelmä rakentuu. Tällä saavutettaisiin sama hyöty kuin ihmisillä, jotka puhuvat yleiskieltä äidinkielenään: tulkkausvaihe yleiskielestä äidinkielelle jää välistä. Sisäinen tietorakenne ei kuitenkaan taipunut tarpeeksi hyvin siirtoprosessin tarpeisiin, ja näin ollen oli helpompaa aloittaa kokonaan alusta.

Informaatio on tallennettu – verkkopalvelulle hyvin tyypillisesti – vain muutamalla erilaisella tietotyypillä, joten tietorakenteen kehittämiseen ei tarvinnut käyttää paljoakaan ylimääräistä aikaa. Lisäksi eri tietotyypeillä on tarkka keskinäinen hierarkia, joten luokkia periyttämällä tietorakenteeseen tarvittavan koodin määrää saatiin dramaattisesti pienennettyä. Luonnollisesti tämä yksinkertaisti myös tietorakenteen käsittelyä itse siirtoprosessissa.

4.4 Tuonti- ja vientiluokat

Sekä tuonti- että vientiluokkien kanssa noudatettiin perinteistä olio-ohjelmoinnin paradigmaa: molemmista luotiin omat abstraktit pohjaluokat ja sen jälkeen lähdettiin toteuttamaan varsinaisia reaaliluokkia. Abstrakteihin luokkiin säilöttiin perusmetodit tietorakenteen läpikäyntiin sekä käyttäjän valitseman informaatiosisällön rajaaminen.

Tuontiluokista toteutettiin ainoastaan Drupal-versio, jotta informaation vienti järjestelmästä onnistuisi. Vientiluokkia kehitettiin enemmän ja laajin, varsinaisesti vientityökaluksi tarkoitettu luokka, oli XML-kokoaja (engl. aggregator). Vientiluokkien yleisin tehtävä oli erilaisten raporttien luominen, joka ei teknisesti poikennut tavallisesta vientiprosessista millään tavalla: Drupal-tuontiluokalla haettiin käyttäjän valitsema informaatio, muutettiin se siirtomuotoon ja annettiin ne valitulle vientiluokalle, joka tässä tapauksessa loi siirtomuodossa olevasta informaatiosta raportin.

Moduulin optimointivaiheessa havaittiin, että Drupalin hallintapuolella käytettävä valikko, josta valittiin siirrettävät tietorakenteet, voitiin myös toteuttaa omana vientiluokkanaan – kaikki muut prosessin osaset paitsi Drupal-menu -vientiluokka oli jo valmiiksi toteutettu ja oli turha pitää kahta samankaltaista toteutusta rinnakkain. Tämä

on tärkeää etenkin koodipohjan ylläpidon kannalta, jolloin mahdolliset muutokset saadaan ulotettua kerralla kaikkiin alaluokkiin. Erikoistilanteissa toteutukset voidaan eriyttää omiksi sisarluokikseen, sillä jos ne eivät toimi samalla tavalla, ne eivät ole saman yleiskäsitteen esiintymiä.

4.5 Asiakasmoduulin ulkoiset komponentit

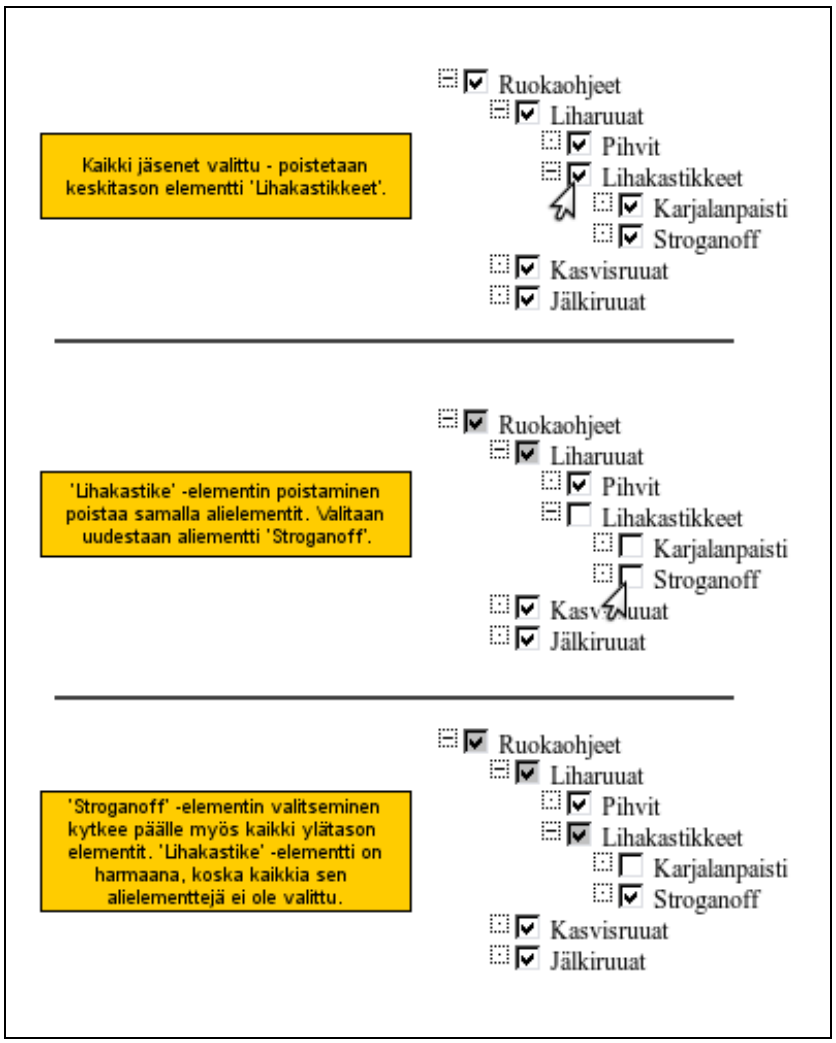
Samalla tavalla kuin Drupalinkin kanssa, moduulin ydinlogiikan ulkopuolelle jäävälle koodille kannattaa ensimmäiseksi etsiä jokin kolmannen osapuolen tarjoama valmistoteutus ja vasta sen jälkeen lähteä itse tekemään. Yleensä tällä tavalla säästää aikaa ja vaivaa ja lopputulos on silti parempi kuin hätäisesti itse tehty vastine.

Moduulin sisällä tämä tilanne tuli vastaan PDF-moottorin valinnassa sekä kahteen otteeseen JavaScript-toteutuksissa. HTML-PDF-kääntäjän kanssa asiassa ei ollut mitään epäselvää: toteuttaminen olisi vienyt kauemmin kuin koko moduulin rakentaminen yhteensä. HTML-PDF-kääntöprosessissa on lisäksi vielä useita sudenkuoppia, kuten CSS-tuki, omien fonttien sisällyttäminen sekä tietysti PDF-formaatin omat nyanssit ja niiden tukeminen. Onneksi saatavilla on useita erilaisia vaihtoehtoja, joista käyttöön valikoitui LGPL-lisensoitu dompdf. Vapaasti käytettävä moottori on teknisesti huippuluokkaa ja sisältää suoraan kaikki asettelun ja ulkoasun vaatimat ominaisuudet. LGPL-lisenssillä dompdf-komponenttia saa käyttää suoraan myös suljettujen ohjelmistojen osana, joten sen käyttäminen ei tuo mukanaan juridisia ongelmia, mikä on mahdollista GPL-lisensoitujen ohjelmistokomponenttien kanssa.

JavaScript on toinen alue, jossa käytetään usein kolmannen osapuolen valmistamia kirjastoja. Tiedonsiirtomoduulissa tarvittiin puurakenteinen valikko, josta vietävät tietorakenteet valittiin. Se oli mahdollista toteuttaa pelkällä HTML:llä. Sen sijaan puun haarojen laskostumiseen, valintojen päivittämiseen ja muuhun vastaavaan dynaamiseen toimintaan JavaScript oli ainoa järkevä vaihtoehto. JavaScript sisältää kuitenkin joukon ongelmia, jotka liittyvät lähinnä eri selainten välisiin tulkintaeroihin sekä siihen, että JavaScript on ainoastaan kieli eikä sisällä juurikaan valmiita komponentteja. Siksi myös JavaScriptille on olemassa framework-kirjastoja. Drupal sisältää yhden tunnetuimmista

kirjastoista, jQueryn, vakiona, joten ensimmäinen päätös ’Mitä pohjakirjastoa tulisi käyttää, vai käyttääkö mitään?’ oli jo tehty valmiiksi. [31.]

jQuery minimoi selaintenväliset ongelmat, koska sen ominaisuudet on toteutettu jokaiselle selaimelle erikseen. Alustusvaiheessa kirjasto tunnistaa käytetyn selaimen ja käyttää sille yksilöityä toteutusta. Kehittäjän kirjoittama ydinlogiikka yksinkertaistuu ja toimii luotettavammin. Lisäksi jQuery mahdollistaa HTML-elementtien dynaamisen muokkauksen suoraan luokka- ja ID-määrityksiä käyttämällä, mikä on vallankumouksellinen tapa ratkaista elementtien käsitteleminen – koodista tulee yksinkertaista, selkeää ja helposti muokattavaa.



Kuva 13. Puurakenteen ohjauslogiikka – elementtien valitseminen.
HUOM! Esitetty tietosisältö ei liity varsinaiseen asiakasprojektiin.

Myös varsinainen puurakenteen ohjauslogiikka oli tarkoitus hakea muualta, ja valmiita toteutuksia oli vastassa läjäpäin. Suurin osa karsiutui puutteellisten ominaisuuksien takia, sillä melkein kaikista puuttui kuvassa 13 esitetty alahaarojen päivittäminen ylemmän haaran muuttuessa – jos hierarkiaan nojautuvan tietorakenteen päänoodin ottaa pois käytöstä, ei alempia voi enää valita tai päänoodi on valittava samalla uudelleen. Jäljelle jäi vain muutama maksullinen vaihtoehto, mutta ostamisen jälkeen niihinkin olisi pitänyt vielä lisätä erilaiset vientityypit, joten tässä kohtaa oli järkevämpää toteuttaa kokonaisuus itse. Suunnitteluun, koodin kirjoittamiseen ja testaamiseen meni reilu päivä, joka olisi hyvinkin voinut vierähtää vieraiden toteutusten mukauttamiseen, eikä lopputulos jäänyt jälkeen millään osa-alueella verrattuna valmisratkaisuihin.

Valinta oman ja kolmannen osapuolen toteutusten välillä ei välttämättä ole aina yksinkertaista. Etenkin tilanteissa, joissa tarjolla olevat ulkoiset toteutukset ovat tuntemattomia ja vähän käytettyjä, on perusteltua käyttää aikaa oman vaihtoehdon suunnitteluun ja kartoittamiseen. Valmiin, mutta epävakaalle pohjalle kasatun järjestelmän päivittäminen ja laajentaminen jälkikäteen on yleensä hankalaa sekä kallista, ja operatiivisella johdolla on harvoin uskallusta rakentaa koko järjestelmää alusta asti uudelleen. Siksi kriittiset päätökset, kuten tärkeiden toteutusten valitseminen, pitäisi tehdä harkitusti eikä kehityspalaverin sivulauseessa, kuten yleensä käy.

5 Yhteenveto

Koko moduulin kehitys on edennyt pelkästään asiakkaan tarpeiden pohjalta ja annetut vaatimukset on täytetty, joten voidaan sanoa, että asiakkaan kannalta moduulista tuli juuri sellainen kuin oli tilattu. Kehitystavasta johtuen määritykset olivat jatkuvassa käymistilassa, mikä oli odotettavaa, sillä koko pääjärjestelmän konseptin tarkentuessa aukeni myös tiedonsiirtomoduulin kannalta uusia kehityssuuntia. Tilanteeseen vaikuttivat myös osaltaan loppuasiakkaiden mielipiteet.

Teknisellä tasolla moduuli pysyi koko ajan yhtenäisenä. Uudet asiakasmääritykset pyrkivät lähinnä lisäämään uusia toiminnallisuuksia eivätkä olleet ristiriidassa edellisten kanssa. Suurin kehitystyö tehtiin muutenkin moduulin ytimen toteutuksessa, jonka tavoitteena oli saada itse siirtoprosessista niin yleispätevä kuin mahdollista. Tämä menettelytapa todistettiin toimivaksi useaan otteeseen projektin aikana.

Kehityssuunnan kääntyessä varsinaisesta tiedonsiirtämisestä raporttien luomiseen jätettiin kuitenkin toteuttamatta kaksi ominaisuutta. Ensinnäkin tiedon tuomiseksi järjestelmään tarvitaan erillinen Drupal-tuontiluokka, joka siivottiin pois ominaisuuslistalta, jotta saatiin tilaa raporttiluokille. Toiseksi itse siirtoprosessi on tällä hetkellä toteutettu siten, että sen kautta saa suoraan tuotua ulos raportin tai valitussa formaatissa olevan vientitiedoston. Sen sijaan suuremmat järjestelmien väliset ajot, jotka joudutaan tekemään taustaprosessina, eivät toimi suoraan. Järjestelmä on edelleenkin mahdollista laajentaa kattamaan molemmat ominaisuudet, ja päätökset niiden poisjättämisestä on tehty tältä pohjalta, asiakkaan ja kehitysryhmän yhteisymmärryksessä. Jos tiedonsiirron kehittäminen nähdään tarpeelliseksi, ominaisuudet otetaan takaisin toteutuslistalle.

Drupal-tuontiluokan rakentaminen on suoraviivainen operaatio, eikä se vaadi järjestelmältä muita uusia komponentteja. Sen sijaan tausta-ajoja varten vaaditaan suurempia uudistuksia. Tiedonsiirto voidaan edelleenkin hoitaa olemassa olevilla perusrakenteilla, mutta sen lisäksi tarvitaan Drupal-pohjainen hallintajärjestelmä operaatioiden koordinoimiseen sekä kattavalla rajapinnalla ohjattava erillinen PHP-

prosessi, joka hoitaa varsinaisen siirtotyön. Suunnittelussa ja toteutuksessa tulee ottaa erityisesti huomioon, etteivät työnsä suorittaneet järjestelmäprosessit jää tausta-ajoon siirron päättyessä tai virhetilanteessa. Etenkin säännöllisesti ajastettuna ja ilman kunnollista valvontaa tilanne voi riistäytyä nopeasti käsistä, kun ylimääräiset prosessit jäävät kuormittamaan järjestelmää. Pahimmassa tapauksessa koko palvelin saattaa jumiutua.

Loppusanat

Suuremmassa mittakaavassa ajatellen tiedonsiirtomoduuli ei välttämättä ole oman lajinsa valioyksilö. Teknisenä suoritteena se täyttää kaikki sille asetetut velvoitteet, mutta käytännön kannalta sen todellinen rooli jäi huomattavasti alkuperäisiä tavoitteita pienemmäksi. Tiedonsiirron näkökulmasta nykyinen sovellus toimii lähinnä runkona, jonka varaan voidaan rakentaa varsinaisia siirtototeutuksia.

Kaikki on kuitenkin kiinni kehitystarpeista. Esimerkiksi tilanne olisi muuttunut täysin, jos joku asiakkaan asiakas olisi halunnut integroida pääprojektin omaan toiminnanohjausjärjestelmäänsä – koko käyttöprosessi olisi mennyt uusiksi ja tarkoittanut täysin uudenlaista asemaa tiedonsiirtomoduulille. Vastaavanlaisten tilanteiden varalle on tärkeää, että jo nykyisellä toteutuksella tiedonsiirto järjestelmästä on perustasolla mahdollista ja asiakas voi esitellä sitä omille asiakkailleen. Tällä tavalla voidaan kenties rohkaista moduulin jatkokehitystä. Pelkkänä teknologiademona tiedonsiirtomoduulin hyödyt valuvat kuitenkin auttamatta hukkaan.

Tiedonsiirtomoduulin tulevaisuudennäkymät ovat epäselvät. Kehitys on toistaiseksi keskeytetty, mutta moduulin tilaa ei kuitenkaan ole virallisesti julistettu lopulliseksi. Moduulissa on selkeitä kehityskohteita, mutta niiden toteuttaminen ei ole kannattavaa, ennen kuin niillä on käytön kannalta jotain merkitystä. Voidaan siis todeta, että tilanne on loppukäyttäjien päätösvallassa. Myös itse pääjärjestelmän muutospaineet saattavat vaikuttaa tavalla tai toisella I/E-moduulinkin kehitykseen.

Lähteet

1 O'Reilly, Tim: What is Web 2.0. (WWW-dokumentti). O'Reilly Media, Inc. <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html?page=2>. 30.9.2005. Luettu 6.1.2009.

2 PHP: PHP Usage Stats. (WWW-dokumentti). <http://www.php.net/usage.php>. Luettu 6.1.2009.

3 PHP. (WWW-dokumentti). Wikipedia. <http://en.wikipedia.org/wiki/Php>. Luettu 6.1.2009.

4 PHP: History of PHP. (WWW-dokumentti). <http://fi.php.net/manual/en/history.php.php>. Luettu 6.1.2009.

5 The PHP Resource Index: Community. (WWW-dokumentti). <http://php.resourceindex.com/Community/>. Luettu 6.1.2009.

6 van der Stock, Andrew: PHP Insecurity: Failure of Leadership. (WWW-dokumentti). <http://www.greebo.net/2006/01/04/php-insecurity-failure-of-leadership/>.4.1.2006. Luettu 6.1.2009.

7 Crane, Aaron: Experiences of Using PHP in Large Websites. (WWW-dokumentti). GBdirect. <http://www.ukuug.org/events/linux2002/papers/html/php/index.html>. Luettu 6.1.2009.

8 Martin, Edwin: What I don't like about PHP. (WWW-dokumentti). <http://www.bitstorm.org/edwin/en/php/>. 6.8.2004. Luettu 6.1.2009.

9 PHP in contrast to Perl. (WWW-dokumentti). <http://tnx.nl/php.html>. Luettu 6.1.2009.

10 On PHP. (WWW-dokumentti). <http://www.tbray.org/ongoing/When/200x/2006/02/17/PHP>. Luettu 6.1.2009.

11 Maurus, Jonas: I'm sorry, but PHP sucks. (WWW-dokumentti). <http://maurus.net/resources/programming-languages/php/>. Luettu 6.1.2009.

12 Pagaltzis, Aristotle: Why PHP is good but bad. (WWW-dokumentti). <http://plasmasturm.org/log/393/>. 21.2.2006. Luettu 6.1.2009.

13 Ledorf, Rasmus: The Web needs an overhaul. (WWW-dokumentti). <http://www.builderau.com.au/strategy/architecture/soa/The-Web-needs-an-overhaul/0,339028264,339284351,00.htm?feed=pt_php>. Luettu 6.1.2009.

14 Tung, Liam: PHP, Python, Samba get security tick of approval. (WWW-dokumentti). ZDNet.com.au. http://www.builderau.com.au/news/soa/PHP-Python-Samba-get-security-tick-of-approval/0,339028227,339289351,00.htm?feed=pt_php>. 28.5.2008. Luettu 6.1.2009.

15 Vamosi, Robert: PHP, Perl and Python pass Homeland Security test. (WWW-dokumentti). CNET News.com. <http://www.zdnet.com.au/news/security/soa/PHP-Perl-and-Python-pass-Homeland-Security-test/0,130061744,339284949,00.htm>. 9.1.2008. Luettu 6.1.2009.

16 Open Source Report 2008. (WWW-dokumentti). Coverity. <http://scan.coverity.com/report/>. Luettu 6.1.2009.

17 Coelho, Fabien: PHP-related vulnerabilities on the National Vulnerability Database. (WWW-dokumentti). <http://www.coelho.net/php_cve.html>. Luettu 6.1.2009.

18 Half-Million Sites Mostly Running PHPBB Forum Software Hacked In Latest Attack. (WWW-dokumentti). CyberInsecure.com <http://cyberinsecure.com/half-million-sites-mostly-running-phpbb-forum-software-hacked-in-latest-attack/>. 12.5.2008. Luettu 6.1.2009.

19 Common PHP-Nuke security vulnerabilities. (WWW-dokumentti). <http://www.karakas-online.de/EN-Book/common-php-nuke-security-vulnerabilities.html>. Luettu 6.1.2009.

20 PHP-Nuke. (WWW-dokumentti). Wikipedia. <http://en.wikipedia.org/wiki/PHPNuke>. Luettu 6.1.2009.

21 phpBB. (WWW-dokumentti). Wikipedia. <http://en.wikipedia.org/wiki/Phpbb>. Luettu 6.1.2009.

22 The PHP App Insecurity Top 20. (WWW-dokumentti). <http://funkatron.com/site/comments/the_php_app_insecurity_top_20/>. 18.4.2007. Luettu 6.1.2009.

23 Drupal Wins Best Overall 2008 Open Source CMS Award for Second Year in a Row. (WWW-dokumentti). <http://drupal.org/Drupal-Wins-Best-Overall-2008-Open-Source-CMS-Award-Packt>. 31.11.2008. Luettu 1.2.2009.

24 Cram, Jon: XML vs YAML vs JSON: A Study To Find Answers. (WWW-dokumentti). <http://webignition.net/articles/xml-vs-yaml-vs-json-a-study-to-find-answers/>. 5.11.2008. Luettu 1.2.2009.

25 JSON. (WWW-dokumentti). Wikipedia. <http://en.wikipedia.org/wiki/Json>. Luettu 6.1.2009.

26 YAML. (WWW-dokumentti). Wikipedia.
<http://en.wikipedia.org/wiki/Yaml>. Luettu 6.1.2009.

27 XML. (WWW-dokumentti). Wikipedia.
<http://en.wikipedia.org/wiki/Xml>. Luettu 6.1.2009.

28 SGML. (WWW-dokumentti). Wikipedia.
<http://en.wikipedia.org/wiki/Sgml>. Luettu 6.1.2009.

29 Document Type Definition. (WWW-dokumentti). Wikipedia.
<http://en.wikipedia.org/wiki/Document_Type_Definition. Luettu 6.1.2009.

30 XML schema. (WWW-dokumentti). Wikipedia.
<http://en.wikipedia.org/wiki/Xml_schema>. Luettu 6.1.2009.

31 Javascript framework usage among top websites. (WWW-dokumentti).
<http://royal.pingdom.com/2008/06/11/javascript-framework-usage-among-top-websites/>. 11.6.2008. Luettu: 1.2.2009.

32 Drupal. (WWW-dokumentti). Wikipedia.
<http://en.wikipedia.org/wiki/Drupal>. Luettu 6.1.2009.

33 Overview. (WWW-dokumentti). Acquia.
<http://acquia.com/about-us>. Luettu 6.1.2009.