VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Matti Ritala

Joomla -komponentin kehitys

Liiketalous ja matkailu
2011

VAASAN AMMATTIKORKEAKOULU
Tietojenkäsittely

TIIVISTELMÄ

Tekijä	Matti Ritala
Opinnäytetyön nimi	Joomla -komponentin kehitys
Vuosi	2011
Kieli	suomi
Sivumäärä	35+0 liitettä
Ohjaaja	Kimmo Paulaharju

Opinnäytetyössä tarkastellaan avoimenlähdekoodin sisällönhallintajärjestelmän Joomla 1.5:n komponenttiohjelmointia PHP -ohjelmointikielellä.

Työssä käsitellään komponentin kehitykselle olennaisia PHP:n olio-ohjelmointiominaisuuksia ja Joomlan ohjelmistoarkkitehtuuria. Näkökulma painottuu niin sanotun malli-ohjain-luokka -arkkitehtuurimallin (MVC) toiminnan kuvaamiseen. MVC-mallin toiminnan ymmärtäminen on ensisijaista, jotta komponentin kehitys olisi tehokasta ja suunnitelmallista.

Joomlan ohjelmistokehyksen toimintaa ja komponentin ohjelmointia havainnollistetaan ohjelmakoodilistauksilla ja koodin kommentoinnilla Lopuksi tarkastellaan ohjelmointiin liittyviä seikkoja ja esitetään lähestymistapa, jolla komponentin kehitys aloittaa.

Työssä havaittiin, että tärkeä osa komponentin kehitystä ja suunnittelua ovat vaatimusmäärittely ja vaatimusten toteuttamismahdollisuuksien punnitseminen Joomlan sovelluskehyksen puitteissa. Havaittiin myös, että MVC -mallin mukainen koodin lohkominen parantaa koodin laatua ja ymmärrettävyyttä.

Avainsanat Joomla, laajennusohjelmointi, MVC, PHP

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietojenkäsittely

ABSTRACT

Author	Matti Ritala
Title	Joomla Component Development
Year	2011
Language	Finnish
Pages	35+ 0 Appendices
Name of Supervisor	Kimmo Paulaharju

The thesis examined the extension development of Open Source content management system (CMS) Joomla 1.5 using PHP programming language.

The work discussed the themes of object-oriented programming with PHP and Joomla's software architecture, issues relevant to Joomla component development. The emphasis was on describing the functioning of Model-View-Controller -architectural pattern which Joomla employs. Grasping the operation of MVC-model is essential for efficient and well designed development.

The operation of Joomla framework and component programming were exemplified with code snippets and by commentary of the code. Lastly, the problems encountered during the development were reflected and a solution and general advice for approaching and beginning the component development were offered.

It was observed that requirement specification and weighting the possibilities to implement the requirements within in the Joomla framework are essential parts of the component development and design. It was further observed that the division of the code according to the MVC -pattern greatly improves the quality of the code and intelligibility.

Keywords Joomla, Extension Development, MVC, PHP

SISÄLLYS

KUVA, KUVIO- JA LISTAUSLUETTELO

Kuvat

Kuviot

Listaukset

1 JOHDANTO

1.1 Aiheen valinta

Idean aiheekseni Joomla! -komponentin (tästä lähdin Joomla) kehityksestä sain ollessani harjoittelijana Vaasan korkeakoulukonsortion alaisuudessa. Vastuullani oli toteuttaa websivut yhdelle konsortion yhteistyöryhmistä. Päädyin tällöin käyttämään websivujen luomisessa Joomla -sisällönhallintajärjestelmää, koska siihen löytyy mittavasti erilaisia laajennuksia ja hyvä monikielisyyden hallintatyökalu.

Monikielisyys oli tärkeä seikka tälle korkeakoulukonsortion alaisuudessa toimivalle ryhmälle, sillä ryhmän jäsenet ovat niin suomen- kuin ruotsinkielisistä oppilaitoksista ja luonnollisesti ryhmä halusi tarjota tietoa itsestään ainakin näillä kahdella kielellä. Kiinnostuin päästä syvemmälle Joomlan sielunelämään ja saada näkemystä muutenkin sisällönhallintajärjestelmien toiminnasta teknisellä tasolla.

Erilaiset sisällönhallintajärjestelmät websivujen alustoina yleistyvät jatkuvasti. Tämän vuoksi tekninen osaaminen niiden ohjelmoinnissa ja käytössä on arvokasta tietopääomaa. Tällöin voidaan helpommin arvioida erilaisten järjestelmien ominaisuuksia

1.2 Tutkimuskysymys ja opinnäytetyönraportin sisältö

Tutkimuskysymyksen voisi muotoilla seuraavasti: Kuinka Joomla -komponentin kehitys ja ohjelmointi tapahtuu?

Tutkimuskysymyksen käytännönläheisyyden vuoksi myös tämän opinnäytetyöraportin sisältö on varsin käytännönläheinen. Luvussa kaksi kerron lyhyesti Joomlasta, jonka jälkeen kerron syvemmin Joomla-kehityksen pääpiirteistä. Lisäksi kerron hieman yleistä teoriaa olio-ohjelmoinnista ja PHP -palvelinkielestä.

Esittelen myös Joomlan ohjelmistoarkkitehtuurin pääperiaatteita, joiden ymmärtäminen on osa menestyksekästä kehitystyötä. Palaan näihin seikkoihin myös myöhemmin raportissani kommentoiden niitä suunnittelu- ja ohjelmointityöskentelyä kuvatessani.

Luvussa 3. kerron toteuttamani Joomla-komponentin suunnittelusta sanallisesti ja kaavioin. Kuvaan komponenttia käyttötapauskaavioin ja käyttöliittymähahmotelmin.

Neljännessä luvussa selostan miten komponentin ohjelmointi tapahtuu. Havainnollistan asiaa muutamin ohjelmakoodikatkelmin ja kuvailemalla Joomlan sovelluskehyksen toimintaa, painottaen arkkitehtuurin yleistä toimintaa. Tarkoitukseni ei ole kädestä pitäen näyttää kuinka kaikki tapahtuu, vaan kuvailla pääpiirteittäin mitä komponentin ohjelmakoodi pitää sisällään.

Luvussa 5. arvioin ohjelmoimani komponentin toimintaa ja esittelen parannusehdotuksia. Mietin myös miten olisin voinut paremmin onnistua työssäni. Lisäksi arvioin hieman Joomlan sovelluskehystä ja kuinka yleisesti toteuttaa hyviä sovellusarkkitehtuurisuosituksia. Luon myös katsauksen siihen mitä olen opinnäyteyöni tekemisenä oppinut ja kuinka kehittää taitojani tulevaisuudessa.

2 JOOMLA

2.1 Joomlasta yleisesti

Joomla on avoimen lähdekoodin sisällönhallintajärjestelmä. Joomla toimii palvelinpuolen skriptikieli PHP:n ja MySQL-tietokannanhallintajärjestelmän päällä. Web-palvelimena voi toimia joko Apache tai Microsoftin IIS. (Wikipedia 2011.)

Joomlalla voidaan luoda monipuolisia webpalveluita niin internetiin kuin intranetiinkin. Se soveltuu portaalimaisten ja verkkolehtimäisten sivujen rakentamiseen tai yksinkertaisen blogin hallinnointiin. Joomlalla voidaan myös rakentaa esimerkiksi toimiva webkauppa monipuolisten laajennusominaisuuksien ansioista.

Joomlalla on toteutettu miljoonia webpalveluita ympäri maailmaa. Joomlan kotisivut väittävät, että noin 2,6 %:n osuus internetin websivuista käyttää alustanaan Joomlan ohjelmistokehystä. Joomlaa käyttävät niin pienet yhdistykset ja Pk-yritykset kuin suuret oppilaitoksetkin. Hyvänä esimerkkinä tästä on Harvardin yliopisto. (Joomla 2011.)

2.2 Historiaa

Joomlan ensimmäinen versio 1.0 julkaistiin syyskuussa 2005. Tämä Joomlan aikainen versio oli kehityshaara toisesta sisällönhallintajärjestelmästä nimeltä Mambo. Joomlan kehittäjätiimi erkaantui tästä projektista, koska Mambo -tavaramerkin omistava yhtiö ei toiminut vapaan lähdekoodin kehityksen periaatteita kunnioittaen. (Wikipedia 2011.)

Versio 1.5 julkaistiin tammikuussa 2008, jonka viimeisin julkaisu 1.5.22 on marraskuulta 2010. Viimeisin Joomla-versio 1.6 julkaistiin 10. tammikuuta 2011 (Wikipedia 2011).

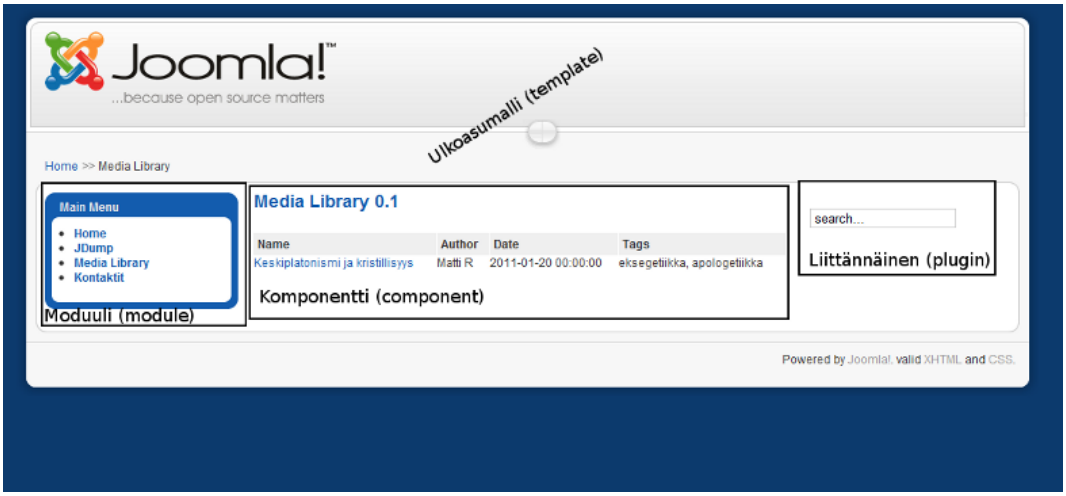
Opinnäytetyötäni tehdessä käytössäni oli Joomlan versioita 1.5.17, koska käyttämäni lähdemateriaali on tämän version aikaista. Suurin osa kuitenkin Joomlan laajennuskehityksen periaatteista pätee myös 1.6:n osalta, koska arkkitehtuuri on pääosin samanlainen.

2.3 Joomla-kehityksen piirteitä

Joomlan avulla voidaan helposti luoda ja päivittää yksikertaisia verkkosivuja. Usein webpalveluissa tarvitaan myös monipuolisempia toimintoja kuten web-kauppa tai esimerkiksi jokin liittymä yrityksen tietokantaan. Joomlan kehityksessä tämä seikka on otettu huomioon mahdollistamalla sisällönhallintajärjestelmän lisäkehitys ja laajennusten ohjelmoiminen.

Joomlan kehitystyötä tehtäessä perusperiaate on, että Joomla on suunniteltu laajennettavaksi, ei muokattavaksi. Laajennusten ohjelmoinnissa ei ole tarpeen koskea Joomlan sovelluskehyksen ytimeen, vaan laajennukset liitetään ytimen ympärille (LeBlanc 2008, 8).

Joomlaa voidaan laajentaa ja muokata viidellä eri tavalla: ulkoasumalleilla (template), kieliversioilla (language), komponenteilla (component), moduuleilla (module) ja liitännäisillä (plug-in).



Kuva 1. Joomla-sivun elementit.

Ulkoasumalleilla voidaan räätälöidä webpalvelun ulkoasua ja asettelua. Näitä malleja voidaan rakentaa HTML:n ja CSS-tyylimäärittelyitä käyttäen.

Komponentilla tarkoitetaan sivun pääosassa näytettävää laajennusta, joka lisää toiminallisuutta sivulle. Komponenteille on tyypillistä kehittyneet ylläpitopuolen ominaisuudet, ns. back-end ominaisuudet, joilla niiden käyttämää tallennetietokantaa ylläpidetään.

Komponenttia ohjelmoitaessa normaalille käyttäjälle näkyvä komponentin luoma näkymä, ns. front-end, ohjelmoidaan erillään ylläpitäjälle näkyvän komponentin hallintanäkymästä, ns. back-end. Komponentin front- ja back-endiä yhdistää usein samat tietokantataulut. Niistä voidaan tarvittaessa ohjelmoida myös toisistaan täysin riippumattomat.

Moduulit ovat sivulla näkyviä pienempiä toiminnallisuuksia ja niitä voi olla useampia. Niillä voidaan esimerkiksi luoda sivuvalikkoja, bannereita tai tuoda jostakin toisesta verkkopalvelusta API-ohjelmoinnilla sisältöä sivulle.

Liitännäiset ovat pieniä laajennuksia, joilla esimerkiksi voidaan kontrolloida jonkin tulosteen ulkoasua kautta koko sivun tai jonkin muun elementin ominaisuuksia sivulla. (LeBlanc 2008, 8–9)

2.4 PHP ja Joomla

Joomla -sisällönhallintajärjestelmä on ohjelmoitu PHP -ohjelmointikielellä. PHP oli alun perin kehitetty serveripuolen tulkattavaksi skriptikieleksi, mutta nykyään sille löytyy jopa kääntäjiä (Wikipedia 2011).

Alkuaan PHP oli lähestymistavaltaan proseduraalinen ohjelmointikieli, mutta versiosta 3 alkaen se sisällytti toimintaansa myös oliosuuntautuneen lähestymistavan (Zandstra 2008,11–13). Versioista 5 alkaen PHP:n ohjelmointiparadigma on ollut vahvasti oliosuuntautunut, vaikkei kieli täysin puhdas oliokieli olekaan. Oliopohjaiseen lähestymiseen ei pakoteta, mutta sitä suositellaan ja oliosuuntautuminen näkyy myös vahvasti PHP:n omissa luokkakirjastoissa (Zandstra 2008,14). PHP:stä on kehittynyt vakavasti otettava olio-ohjelmointikieli ja tämän vuoksi sillä voidaan rakentaa monimutkaisia ja tehokkaita websovelluksia.

2.5 PHP ja olio-ohjelmointi

Joomla hyödyntää ydinkoodissaan ja sovelluskehyksessään PHP:n olio-ohjelmointiominaisuuksia ja siksi ajattelenkin, että olio-ohjelmoinnin periaatteiden ymmärtäminen on suositeltavaa alettaessa ohjelmoida laajennuksia Joomlalle.

Luokan ja olion käsitteet ovat tärkeä hahmottaa. Luokka on abstrakti tietotyyppi, joka kertoo minkälaisia piirteitä sen ilmentymällä, oliolla on. Nämä piirteet ovat attribuutteja, kuten muuttujat ja vakiot, sekä metodeita eli funktioita. Luokka on siis kuin muotti, josta olio luodaan. (Koskimies 2000, 37).

Olio on luokasta luotava ajon aikainen ilmentymä, jolla on luokan määrittelemät attribuutit ja metodit. Lisäksi olion tietotyypiksi määräytyy luokka, josta se on ilmentymä.

Listauksen 1 mukaisesti PHP:ssä luokka määritellään komennolla ”Class”, jonka jälkeen aaltosulkujen sisään sijoitetaan sen attribuutit ja metodit.

```
Class Henkilö {
      public $nimi = null;
      public $ikä = null;

      public function kerroNimi() {
            echo $this->nimi;
      }

      public function kerroNimi() {
            echo $this->ikä;
      }
}

$Henkilö1 = new Henkilö();
$Henkilö1->nimi = "Matti";
```

Listaus 1. Luokka ja olio PHP:llä ilmaistuna.

Esimerkissä käytettävä $this -muuttuja tarkoittaa, että viittaamme luotavaan olioon itseensä. Operaattori ”->” tarkoittaa, että viittamme olion attribuuttiin tai metodiin.

Luokasta luodaan ajon-aikainen olio komennolla ”New”, ja usein tähän olioon viitataan jollain muuttujalla, kuten listauksen ”$Henkilö”. Listauksessa 1 luomme siis ilmentymän luokasta Henkilö ja asetamme olion $nimi -attribuutin arvoksi merkkijonon ”Matti”.

Olio-ohjelmointikielissä perintä on mekanismi, jolla yksi tai useampi luokka voidaan johtaa perusluokasta. Johdettu lapsiluokka perii isäluokaltaan sen tyypin, attribuutit ja metodit (Zandstra 2008, 30). Listauksessa 2 näemme kuinka PHP:ssä perintä toteutetaan komennolla ”Extends”.

Luokalla ”MiesHenkilö” on kaikki samat attribuutit ja metodit kuin Henkilö -luokallakin, vaikka niitä ei ekspliittisesti ilmaistakaan koodissa. Johdetulla luokalla on myös sille ominaisia piirteitä, joita isäluokalla ei ole. Tässä tapauksessa piirteet ovat $sotilasarvo -attribuutti ja kerroSotilasarvo -metodi.

```
Class MiesHenkilö extends Henkilö {
	public $sotilasarvo = null;

	public function kerroSotilasarvo() {
		echo $this->sotilasarvo;
	}
}
```

Listaus 2. Isäluokasta johdettu luokka

Joomla -laajennusten ohjelmoinnissa perintämekanismi on tärkeä ja tehokas, koska järjestelmän sovelluskehyksestä voidaan periyttää tarvittavia luokkia, niin ettei koodia tarvitse aina kirjoittaa uusiksi.

2.6 Ohjelmistoarkkitehtuuri

Joomlan kaltaisen järjestelmän suunnittelussa, ohjelmoinnissa ja ylläpidossa tarvitaan järjestelmäarkkitehtuuria.

Lyhyesti ilmaistuna arkkitehtuuri määrittelee kuinka järjestelmää on kehitettävä ja kuinka sitä on ylläpidettävä. Arkkitehtuurin päämääränä on estää tarpeetonta luovuutta ja asettaa rajat ohjelmoijille ja suunnittelijoille, jotka työskentelevät järjestelmän kanssa. Osuva kuvaus ohjelmiston arkkitehtuurista onkin, että se on ikään kuin järjestelmän perustuslaki. Sitä ei pitäisi muuttaa ja siitä ei pitäisi poiketa kuin painavista syistä. (Koskimies, Mikkonen 2005, 18–19).

Hyvän koodin ja järjestelmän piirre on muutenkin ortogonaalisuus eli koodia voidaan käsitellä ja muokata yksittäisinä kokonaisuuksina, niin ettei aiheuteta dominoefektin tavalla muutoksia muualla järjestelmässä. (Zandstra 2008, 103). Tämän vuoksi Joomla onkin suunniteltu tukemaan laajennusten ohjelmointia, niin ettei järjestelmän ydintä tarvitse muuttaa.

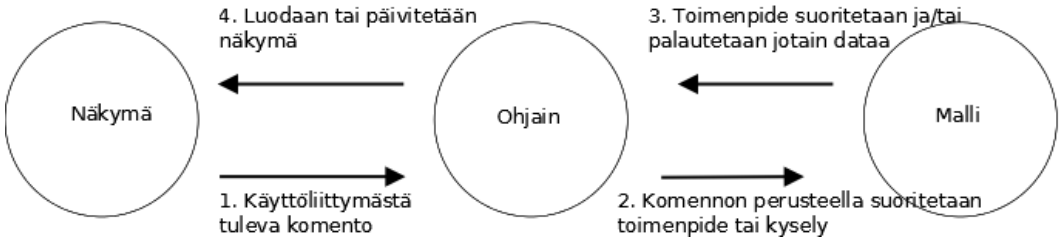
2.7 Joomlan ohjelmistoarkkitehtuuri

Model-View-Controller (MVC) eli malli-näkymä-ohjain on ohjelmistoarkkitehtuurimalli (software architectural pattern) (Wikipedia 2011) tai arkkitehtuurityyli (Koskimies 2005, 125), jossa sovelluslogiikka on erillään käyttöliittymästä. Malli ei ole järin uusi, vaan sen kehitettiin jo 1970-luvun lopulla (Reenskaug 2011).

MVC-malli on varsin yleisesti käytössä websovelluksissa, ymmärrettävästä syystä: tavallinen XHTML -koodi ja CSS -tyylimäärittelyt halutaan pitää mahdollisimman paljon erillään sovelluslogiikan toteuttavasta koodista.

Komponenttien, moduulien ja muiden laajennusten kehityksessä suositellaankin käytettävän MVC-mallia, koska näin laajennuksista tulee joustavia ja niiden kehityksessä voidaan hyödyntää monipuolisesti Joomlan sovelluskehyksen luokkia ja niiden metodeita. Joomlan sovelluskehys ohjelmoitu niin, että oletuksena se etsii MVC-mallin mukaisesti hakemistorakenteesta tiedostoja ja luokkia. Tämän vuoksi vähintäänkin MVC-mallin perusteiden hahmottaminen on tärkeää tehokkaan

työskentelyn mahdollistamiseksi ja epävarmuuden vähentämiseksi. (Joomla Documentation 2011).



Kuvio 1. MVC- mallin toiminta.

Malli on rakenne (esim. olio), joka vastaa sovelluksen käyttäytymistä ja tietojen käsittelystä eli sovelluslogiikasta, vastaten ohjaimelta tuleviin tilakyselyihin tietojen esittämistä varten tai käskyihin käsitellä ja tallentaa tietoja (Koskimies 2005, 142).

Malli ei välttämättä aina palauta dataa. Joskus ohjain voi vain välittää näkymältä tulevan komennon muokata mallin hallinnoimaa dataa tai suorittaa jotain tarkistuskyselyitä ilman palautettavaa dataa, joka näkyisi käyttäjälle. (Wikipedia 2011.).

Ohjain ottaa vastaan käyttäjältä tulevat syötteet ja välittää ne mallille, joka suorittaa niiden pohjalta toimenpiteitä ja ohjaa mallilta tulevan tiedon näkymälle, jolla tieto esitetään käyttäjälle (Koskimies 2005, 143).

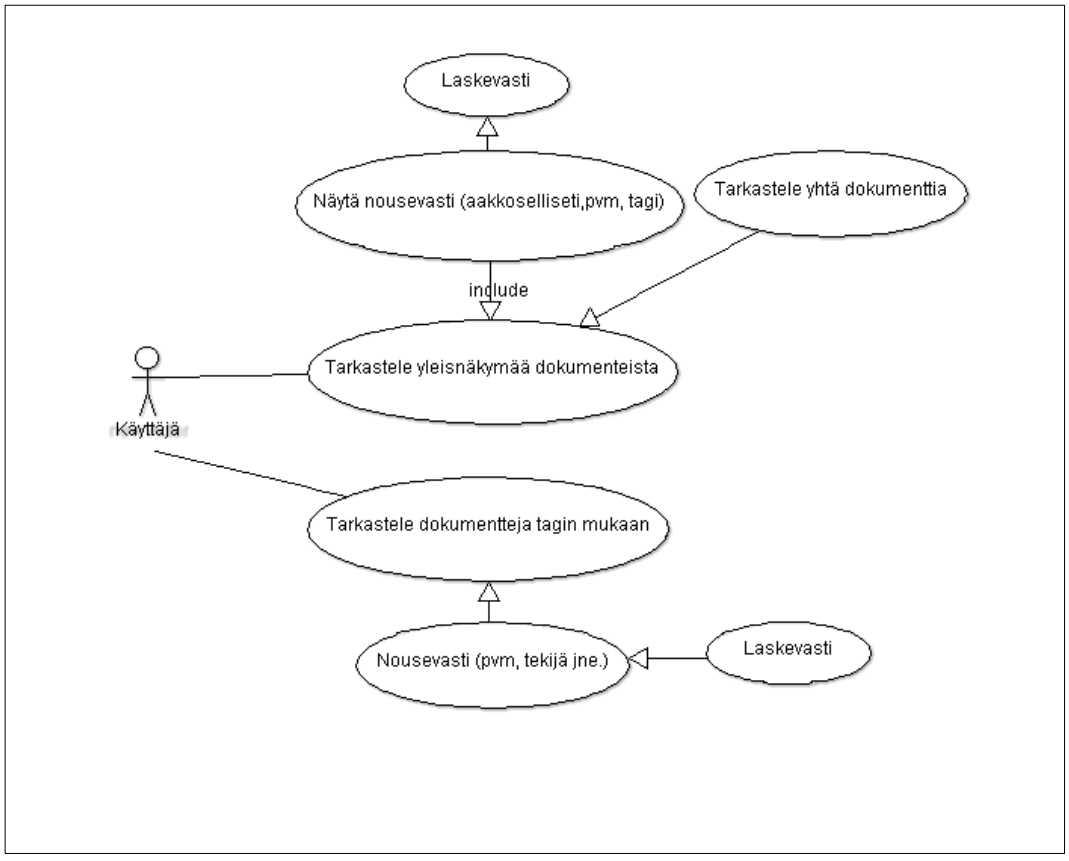
Näkymä mallintaa mallilta tulevat tiedot käyttäjälle ymmärrettävään muotoon käyttöliittymäelementtinä, kuten kuvio 1 esittää. Mallista voidaan luoda erilaisia näkymiä eri käyttötarkoituksia varten. Mallin luoma käyttöliittymä antaa ohjaimelle toimenpidekäskyjä (Koskimies 2005, 142).

Tämä on MVC-rakenteen pääpiirteinen toiminta. Erilaisissa sovellutuksissa rakenne voi toimia jäykemmin tai joustavammin, esimerkiksi Joomlan komponenteissa näkymä saa hieman keskeisemmän roolin kuin tässä peruskuvauksessa.

3 KOMPONENTIN SUUNNITTELU

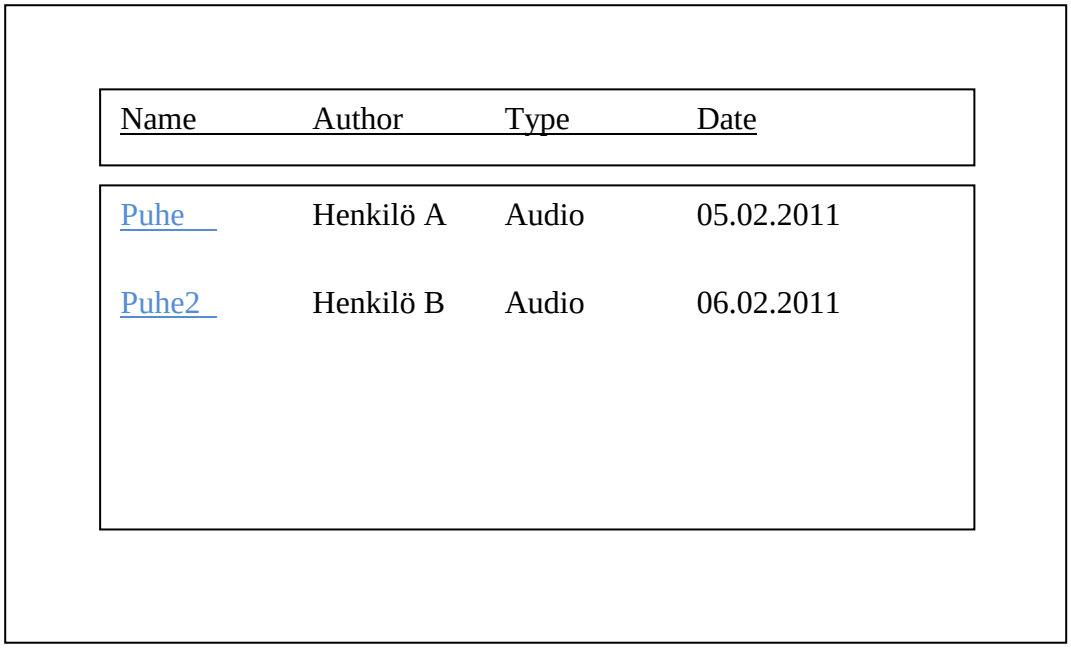
Ennen komponentin kehitystä ja ohjelmointityön etenemisen selostamista, on selkeyden kannalta syytä kuvata, minkälaisen komponentin kehityksestä on kyse.

Kehitettävä komponentti, on erilaisten dokumenttien tallentamiseen ja jakelemiseen tarkoitettu mediakirjasto. Sivuston tavalliselle selaajalle tarkoitettu front-end -näkymä koostuu listasta tietokantaan ja palvelimelle tallennetuista dokumenteista. Kuviossa 2 nähdään käyttötapauksia, joihin komponentin front-end tulisi pystyä vastaamaan.



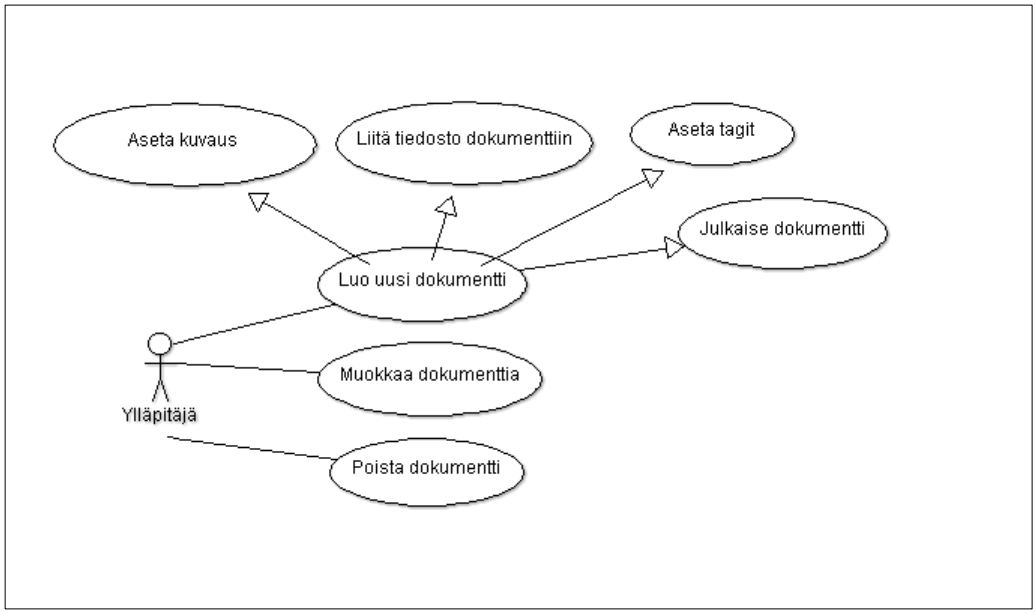
Kuvio 2. Komponentin front-end -käyttötapaukset

Kuvio 3:n mukaisesti listasta voidaan valita yksittäinen dokumentti (nimen linkkiä klikkaamalla) ja siirtyä tarkastelemaan liittyviä otsikkotietoja ja kuvauksista sekä päästä käsiksi itse dokumenttiin tai mediatiedostoon. Mediatiedoston tyyppiä ei ole rajoitettu, se voi olla tiedostomuodoltaan mitä muotoa hyvänsä.



Kuvio 3. Komponentin front-end käyttöliittymä

Niin ikään ylläpitäjän back-end näkymä kostuu listasta tallennettuja dokumenteista ja niiden tiedoista. Samaan tapaan kuin front-end näkymästä voidaan klikkaamalla valita yksittäinen tallennettu tietue ja tarkastella sitä, myös back-endissä olisi oltava samanlainen toiminto.



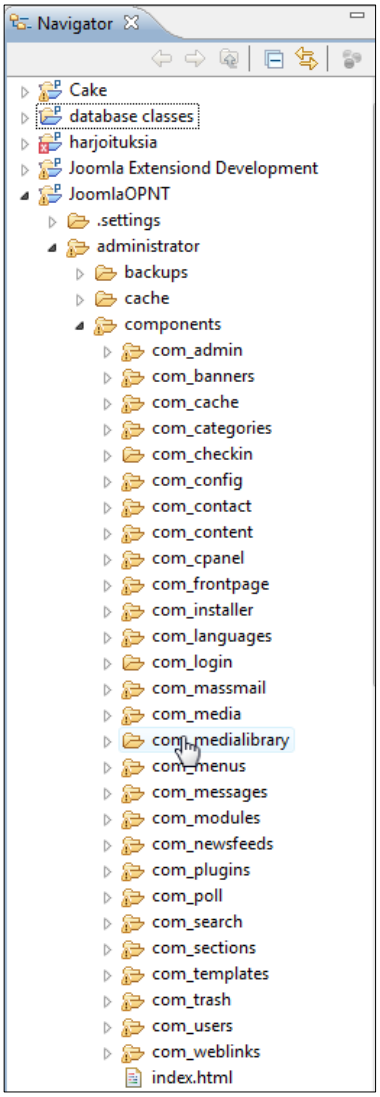
Kuvio 4. Back-end -käyttötapaukset

Erona front-end näkymään on kuitenkin, että ylläpitäjä voi lisätä tietokantaan uusia dokumentteja ja niiden tietoja sekä ladata palvelimille itse dokumenttitiedoston. Ylläpitäjällä on vastaavasti myös mahdollisuus poistaa tietueita tietokannasta ja poistaa palvelimelle ladatut tiedostot. Lisäksi ylläpitäjälle on oltava mahdollista muokata jo olemassa olevia tietueita.

4 TOTEUTUS JA OHJELMOINTI

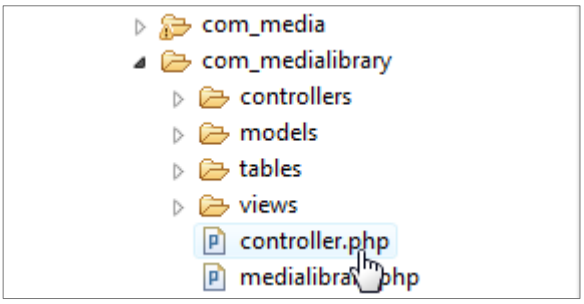
4.1 Hakemistorakenne

Joomla -komponentin ohjelmoimisen aloitin luomalla komponentin hakemistorakenteen. Kuvasta 2 voidaan nähdä, että Joomlan juurihakemistossa on hakemisto ”administrator” ja tämän alikansio ”components”, jonka alta löytyvät kaikkien komponenttien back-end -tiedostot. Yksittäisten komponenttien hakemistot alkavat etuliitteellä ”com_”.



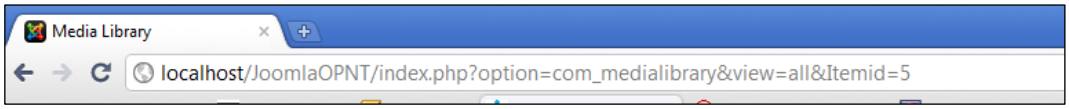
Kuva 2. Komponenttien hakemistorakenne.

Ohjelmoimani komponentin hakemisto on ”com_medialibrary”. Luomastani kansiosta (katso kuva 3) löytyvät merkillepantavasti MVC-rakenteen mukaisesti nimetyt kansiot ”models” ja ”views” sekä controller.php, joka toimii ohjainrakenteen lähdekooditiedostona. Lisäksi komponentin juurihakemistossa on controllers -kansio, jonne voitaisiin sijoittaa vaihtoehtoisia ohjainrakenteita rakenteen selkeyden säilyttämiseksi, mutta kansio on jätetty tyhjäksi tällä erää.



Kuva 3. Kehitettävän komponentin hakemisto.

Komponentin juurihakemistossa on myös medialibrary.php -tiedosto, joka toimii eräänlaisena porttina koko komponentin suoritukseen ja ohjelmointiin. Kun Joomlan sovelluskehys saa kuvan 4 mukaisesti osoiterivillä kulkevan muuttujan ”option” arvona komponentin hakemiston nimen, lukee se ensimmäiseksi komponentin kansiosta tiedoston komponentin nimeä vastaavan tiedoston ”medialibrary.php”.



Kuva 4. Osoiterivillä välitetyt muuttujat.

4.2 Alkupiste – medialibrary.php

Joomla -sovelluskehykseen ohjelmoitaessa suositellaan tarkistettavan onko vakio ”_JEXEC” asetettu, muuten koodin suorittaminen lopetetaan listauksen 3 mukaisesti. Tällä varmistetaan, ettei koodia voida suorittaa muuten kuin, että ensiksi suoritetaan Joomlan juurihakemistossa sijaitseva index.php, jossa tämä _JEXEC-vakio asetetaan.

```
<?php
/**
 * @package     MediaLibrary
 * @subpackage Components
 * @author Matti Ritala (2011)
 */
defined ( '_JEXEC' ) or die ('Restricted access');
// Haetaan komponentin perusohjain
require_once (JPATH_COMPONENT_ADMINISTRATOR.DS.'controller.php');
/*Tarkastetaan JRequest -luokan getWord-metodilla, jos kyselytaulukosta
 * löytyy controller-muuttujalle arvo ja luodaan polku ohjaimen tiedos-
toon ja
 * haetaan require_once -functiolla tuon ohjaimen luokkakoodi */
if($controller = JRequest::getWord('controller')) {
	$path = JPATH_COMPONENT.DS.'controllers'.DS.$controller.'php';
	if(file_exists($path)){
		require_once($path);
	}
	else
	{
		$controller='';
	}
}
```

Listaus 3. Medialibrary.php:n alkuosa.

Seuraavaksi kutsun komennolla ”require_once”, Joomlan hakemistorakennevakio ”JPATH_COMPONENT_ADMINISTRATOR”, avulla controller.php -tiedoston, joka sisältää myöhemmin koodissa kutsuttavan luokan ”MediaLibraryController”.

Tämän jälkeen tulee ohjausrakenne, jolla tarkistetaan mahdollinen controller -muuttujan arvo. JRequest -luokan avulla tarkistan GET, POST, FILES ja COOKIE -taulujen sisältämiä muuttujia ja arvoja. Ohjausrakenne on käyttämätön tällä hetkellä, koska en ohjelmoinut muita ohjaimia.

```
JTable::addIncludePath(JPATH_COMPONENT_ADMINISTRATOR.DS.'tables');
/* Luodaan ohjain. Jos JRequestilla ei saatu mitää erityistä ohjainta,
niin käytetään komponentin juuresta löytyvää controller.php:ta ja Me-
diaLibraryController:ia.
 */
$classname = 'MediaLibraryController'.ucfirst($controller);
$controller = new $classname;
/* Suoritaan task -kysely */
$controller->execute(JRequest::getWord('task'));
$controller->redirect();
```

Listaus 4. Medialibrary.php:n loppuosa.

Myöhempää käyttöä varten kutsun koodissa JTable -luokan metodia ”AddIncludePath”. Tällä Joomlan sovelluskehyksen tietokantakyselyistä vastaava JTable -luokka tietää mistä etsiä nimenomaan tämän komponentin tietotokantaulujen käsittelyyn tarvittavia tietokantaluokkia. Websovelluksissa käytetään yleisesti jonkinlaista abstraktiota, kun käsitellään tietokantatauluja, Joomlassa tämä tapahtuu JTable -luokan avulla.

Hyvän ohjelmointitavan mukaan kutsun controller.php -tiedostosta löytyvää komponentin pääohjaimen ilmentymää, vaikka luokan voisi sijoittaa tähänkin tiedostoon. Tiedostot on kuitenkin hyvä sijoitella erikseen selkeästi ja MVC-arkkitehtuuria kunnioittavasti.

MediaLibraryController -luokasta tehdään ilmentymä ja tällä suoritetaan metodi ”execute”, joka suorittaa taas ilmentymän metodin, joka vastaa nimeltään task-muuttujan arvoa. Redirect -metodi on siltä varalta, että jos yritetään kutsua tuntematonta tehtävämetodia, niin tämä metodi ohjaa sovelluksen takaisin ylläpitäjäkäyttöliittymän päänäkymään.

Tästä siis kaikki alkaa. En selosta ohjelmoimani komponentin jokaista lähdekooditiedostoa tällä tarkkuudella, ja tyydyn hieman suuripiirteisempään kuvaukseen seuraavissa osioissa.

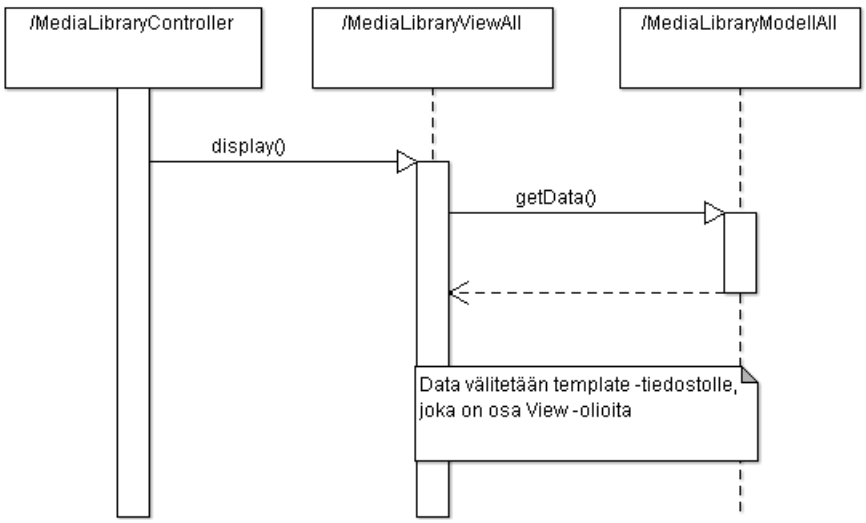
4.3 Ketjutus

Tässä vaiheessa on syytä tarkastella Joomlan MVC-rakenteen toimintaa ja kuinka komponentin eri osat välittävät tietoja toisilleen.

Kuten edellisessä osuudessa näimme, kaikki alkaa ns. entry-point -tiedostosta, tässä tapauksessa on ”medialibrary.php”. Tiedostossa luodaan ilmentymä MVC-arkkitehtuurin mukaisesta MediaLibraryController -ohjainluokasta. Ilmentymällä on useita metodeja, joilla komponentin toiminnallisuus toteutetaan. Yksi metodeista on esimerkiksi ”display”, jolla kutsutaan komponentin erilaisia näkymiä, joilla data tulostetaan selaimeen. Tietojen käsittely ja tallentaminen tietokantaan tapahtuu ohjaimen muiden metodien ja niihin liittyvien mallien avulla.

Oletuksena Joomlan JController -luokasta periytetyt ohjaimet suorittavat aina display -metodinsa, katso kuvio 5. Tästä seuraa seuraavanlainen ketju:

1. Joomlan sovelluskehys etsii komponentin views -hakemistosta näkymän nimeä vastaavan hakemiston, esimerkiksi ”all”.
2. Hakemistosta löytyvästä view.html.php -tiedoston View -luokasta luodaan ilmentymä.
3. Ilmentymän display -metodi kutsuu oletuksena models -kansiosta näkymän nimen mukaista mallia ja sen jotain metodia, esim. getData.
4. Tällä Models -luokan metodilla haetaan ja käsitellään tietokannasta näkymälle olennaista dataa.
5. Tämä data välitetään näkymän ns. template -tiedostolle, jossa tietokannasta haettu tieto esitetään tavallisesti HTML-koodina.



Kuvio 5. Näkymän luonti, sekvenssikaavio

4.4 Ohjain - Controller

Entry-point -tiedosto siis luovuttaa komponentin suoritusvastuun erillisessä tiedostossa sijaitsevalle Controller -luokan ilmentymälle. Joomlassa Controller -luokat periytetään JController -luokasta (katso listaus 5), joka on Joomlan ohjelmistokehykseen valmiiksi ohjelmoitu isäluokka.

Isäluokalla on komponentin suorittamiseen tarvittavia metodeja, kuten mainitsemani display -metodi ja siksi komponenttia ohjelmoitaessa onkin suositeltavaa, että komponentin luokat periytetään tästä Joomlan sovelluskehyksen luokasta. Muutenhan pyörän joutuisi niin sanotusti keksimään joka kerta uudelleen.

```
jimport('joomla.application.component.controller');
/**
 * Media Library Component Controller
 *
 * @package    Media Library
 * @subpackage Components
```

```
 */
class MediaLibraryController extends JController
{
	/**
	* Method to display a list of all media items in the library
	*
	*
	*/
	  function display()
	  {
		$task = JRequest::getVar('task');
		switch ($task) {
			case 'edit':
			case 'add':
				JtoolBarHelper:: apply();
				JtoolBarHelper:: save();
				JtoolBarHelper:: cancel();
				JtoolBarHelper:: deleteList();

			break;
			default:
				JtoolBarHelper:: Title ( JText::_( 'Media Li-
brary 0.1' ), 'generic.png');
				JtoolBarHelper:: editList();
				JtoolBarHelper:: deleteList('Are you sure?');
				JtoolBarHelper:: addNew();
			break;
		}

		$view = JRequest::getVar('view');
		if(!$view) {
			JRequest::setVar('view', 'all');
		}
		parent::display();
	  }
```

Listaus 5. Controller.php:n alkuosa

Listauksessa 5 näemme ohjaimen display -metodin. Display -metodia kutsutaan aina oletuksena ja siksi sijoitinkin komponentin ylläpitokäyttöliittymän luomiskäskyjä tähän metodiin.

Kutsun Joomlan sovelluskehyksestä tulevan globaalin $task -muuttujan arvosta riippuen tiettyjä back-end käyttöliittymän elementtien luontityökaluja JHelper -luokan metodilla. Oletuksena, kun $task -muuttujalla ole mitään erityistä arvoa, kutsutaan esimerkiksi komponentin otsikon luontiin tarkoitettua metodia ”Title”. Muilla metodeilla luodaan hallintapainikkeita. Nämä ovat standardinomaisia, koska tällöin Joomlan JavaScript -kehys vastaa painikkeiden toiminnallisuudesta.

Määrittelen, että jos $view -muuttujalla ei ole arvoa, niin sen arvoksi asetetaan ”all”. Tällä varmistan, että aina kun komponenttia kutsutaan, niin ladataan oletusnäkymänä all-näkymä. Lopuksi kutsun ohjaimen isäluokan metodia ”display”, johon on ohjelmoitu Joomlan ohjelmistokehyksen mukaiset ohjausrakenteet näkymien etsimiseksi hakemistorakenteesta.

Muita metodeja ohjainluokalla on mm. “add”, “save”, “edit”, “upload, “delete-file”, “remove”. Näillä metodeilla hoidetaan tietojen välittäminen näkymiltä malleille, joilla tietojen käsitteleminen ja tallentaminen itse asiassa tapahtuu.

Kyse on niin sanotusta ”laihat ohjaimet, pulskat mallit” (skinny controllers, fat models) -periaatteesta. Ohjainluokan metodien ei tulisi sisältää kovin paljon malleille tarkoitettua tietojen käsittelyä ja tallentamista, vaan suurin osa ns. älykkäästä toiminnasta tulisi sijoittaa malleihin. Ohjaimen olisi hyvä vain välittää tietoja näkymältä mallille ja toisin päin. Näin kunnioitetaan MVC-arkkitehtuuria ja koodista tulee uudelleenkäytettävää, sillä mallien metodeita voidaan käyttää ohjaimen muissa metodeissa tai usein jossain toisessa luokassa. (Developer.com 2010).

Usein koodissa vilisee metodeita (esim. JToolBarHelper::apply()), joita kutsutaan ilman, että niiden luokasta tehtäisiin ilmentymää. Tällöin on kyseessä staattinen metodikutsu, jota käytetään juuri tällaisissa abstrakteissa apuluokissa ja ns. tehdasluokissa. Palvelimen suorituskyky olisi kovilla nopeasti, jos jokaista metodia varten tulisi luoda ilmentymä kyseistä luokasta. Siksi onkin kätevämpää vain viitata näiden tehdas- ja apuluokkiin staattisesti. Usein staattisiin metodeihin sijoitetaankin yksinkertaisia komentoja ja palautteita, joita tarvitaan joustavasti kaikkialla järjestelmässä ja joihin ei tarvita erillistä olioita. (Zandstra 2008, 47–48).

4.5 Malli – Model

Yleensä komponentin toiminta alkaa näkymän lataamisella, siksi ennen mallin lataamista joudutaan luomaan ilmentymä näkymäluokasta. Edellä totesin, että kun ohjaimen display -metodi suoritetaan, ohjain etsii view -muuttujan sisällön nimistä hakemistoa view -hakemiston alta. Määrittelin ohjaimen display -metodissa, että oletuksena view -muuttujan arvoksi asetetaan ”all”, eli all -näkymä. Luokka,

josta ilmentymä luodaan, sijoitetaan view -hakemiston view.html.php -tiedostoon, josta Joomlan sovelluskehys osaa hakea sen ilman eksplisiittistä sisällytyskäskyä.

Tiedoston koodissa sovelluskehyksen JView -luokasta johdetaan lapsiluokka (katso listaus 6), joka on komponenttini tapauksessa nimeltään ”MediaLibraryViewAll”. Nimi kuvaa mitä näkymä tulostaa selaimeen. Tässä tapauksessa selaimeen tulostetaan kaikki mediakirjastoon tallennetut tietueet. Lisäksi nimeämiskäytäntöä ”komponentin nimi – View - näkymän nimi” seuraamalla voidaan hyödyntää Joomlan sovelluskehyksen ketjutusta, joka luo ilmentymiä luokista automaattisesti ja kutsuu niiden metodeita.

```
class MediaLibraryViewAll extends JView
{
      /**
      * display method of All view, pushes data to the template.
      * @return void
      **/
       function display($tpl=null)
       {
              $rows = $this->get('data');
              $this->assignRef('rows', $rows);
              parent:: display();
       }
}
```

Listaus 6. View.html.php

Ilmaisu ”$this” on PHP:ssä viittaus luokan ilmentymään eli olioon itseensä ja tässä yhteydessä käytän JView -luokalta perittyä metodia ”get” hakemaan näkymään liittyvän MediaLibraryModelAll -mallin metodia ”getData”. Mallin palauttama data asetetaan ilmentymän viitemuuttujiin, joita voidaan hyödyntää, kun luodaan näkymää lähettäväksi selaimelle.

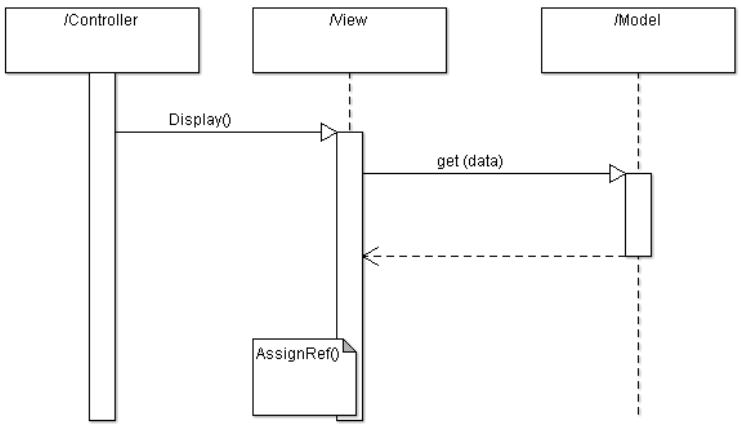
Listauksessa 7 näemme ModelLibraryModelAll -malliluokan koodin, joka on varsin selkokielinen. GetData -metodissa muodostetaan SQL-kielinen kysely, joka lähetetään JModel -luokalta perityllä _getList -metodilla Joomlan sovelluskehyksen tiedossa olevalle tietokannalle ja vastauksena saatu taulukko kyselyobjekteista

asetetaan luokan data-attribuutin arvoksi, jonka siis view.html.php -tiedostossa assignRef -metodi välittää eteenpäin.

```
class MediaLibraryModelAll extends JModel
{
	var $data = null;
	function getData()
	{
		if(empty($this->data)) {
			$db = JFactory::getDBO();
			$prefix = $db->getPrefix();
			$query = "SELECT * FROM " . $prefix . "mediali-
braryitems";
			$this->data = $this->_getList($query);
		}
	return $this->data;
	}
}
```

Listaus 7. All.php ja ModelAll -luokka.

View.html.php -tiedostossa kutsumani get -metodin loogisen suorituksen olen siis erottanut tähän malliluokkaan ja sen ilmentymään, koska MVC-periaatteen mukaan haluan toteuttaa tietojen käsittelyn ja hakemisen tietokannasta malliluokissa. Kuvion 6 sekvenssikaavioista voimme nähdä edellä selostetun toiminnan.



Kuvio 6. Komponentin MVC-rakenteen toiminta sekvenssikaaviona

All -näkymän malli on varsin yksikertainen koodiltaan, mutta sijoitin muihin malliluokkiin erilaisia toiminnallisuuksia, kuten esimerkiksi tiedoston siirron palvelimelle, tietueiden järjestelyn eri sarakkeiden perusteella ja merkkijonojen jäsentely taulukoksi.

Sijoittamalla toiminnallisuuden malleihin MVC-rakenne pysyy selkeänä ja koodiin on helpompi palata myöhemmin.

4.6 Näkymä – View

Kun data on haettu tietokannasta, täytyy se vielä muokata seilamessa tulostettavaan muotoon. Tämä tapahtuu niin, että view -olio kutsuu omaa isäluokaltaan perimäänsä metodiansa ”AssignRef” ja tämän parametriksi asetetaan viittaus model -oliolta saatuun dataan, katso listaus 6 muistin virkistämiseksi.

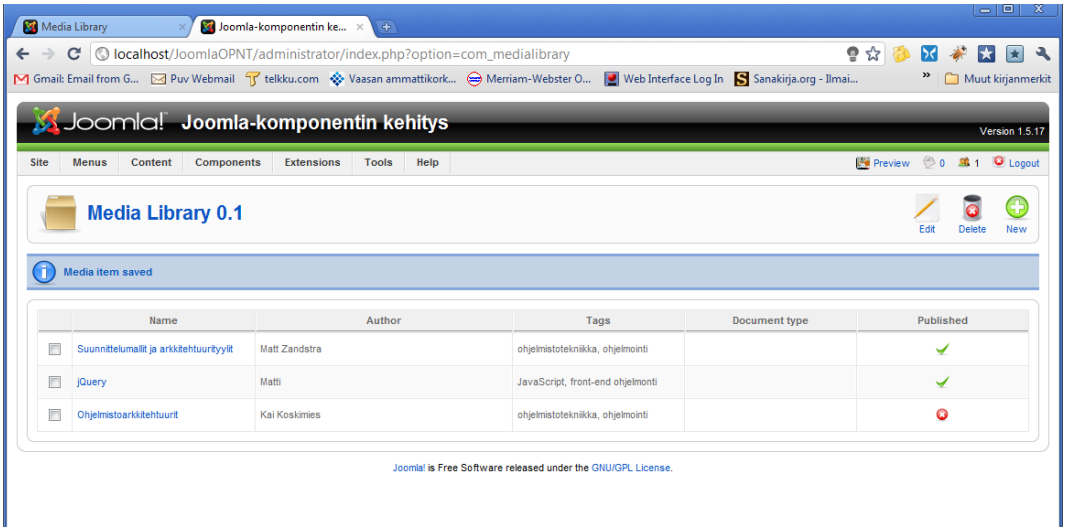
Tämän jälkeen suoritetaan JView -isäluokalta peritty display -metodi, joka osaa hakea hakemistorakenteesta niin sanotun sapluunan, templaten. Template -tiedosto muistuttaa eniten tavallista XHTML -tiedostoa, johon erilaisilla käskyillä ja ohjausrakenteilla tietokannasta haettu data muotoillaan. Listauksessa 8 näemme All -näkymän oletussapluunan.

```
<form action="index.php" method="post" name="adminForm" id="adminForm">
<table class="adminlist">
<thead>
		<tr>
			<th width="1%">
			</th>
			<th class="title" width="10%">Name</th>
			<th width="15%">Author</th>
			<th width="10%">Tags</th>
			<th width="10%">Document type</th>
			<th width="10%" nowrap="nowrap">Published</th>
		</tr>
	</thead>
	<?php
	/* Populoidaan html-taulukko MediaLibraryViewAll -luokan tulevalla
	 * rows-muuttuja datalla */
	jimport('joomla.filter.output');
```

```
	$css = 0;
	for ($i=0, $n=count($this->rows); $i < $n; $i++)
	{
		$row = $this->rows[$i];
		$checked = JHTML::_('grid.id', $i, $row->id);
		$published = JHTML::_('grid.published', $row, $i);
		$link = JFilterOutput::ampReplace('index.php?option=' .
$option . '&task=edit&cid[]=' . $row->id);
	?>
	<tr class="<?php echo "row$css";?>">
		<td align="center">
			<?php echo $checked; ?>
		</td>
		<td>
		<a href ="<?php echo $link; ?>"><?php echo $row->name;
?></a>
		</td>
		<td>
			<?php echo $row->author; ?>
		</td>
		<td>
			<?php echo $row->tags; ?>
		</td>
		<td>
			<?php echo $row->documenttype; ?>
		</td>
		<td align="center">
			<?php echo $published; ?>
		</td>
	</tr>
		<?php
		$css = 1 - $css;
	}?>
</table>
<!-- option-muuttujaa varten piilotettu kenttä, että osataan ladata oi-
kea komponentti -->
<input type="hidden" name="option" value="<?php echo $option?>" />
<!--  JavaScript varten piilotetut kentät
JS-tulostaa task-kenttään task-muuttuja arvon, että MediaLibraryControl-
ler
osaa ladata oikean näkymän formin lähettämisen jälkeen
 -->
<input type="hidden" name="task" value="" />
<input type="hidden" name="boxchecked" value="0" />
</form>
```

Listaus 8. Template -tiedosto default.php

Koodissa luon sivulle Joomlan ylläpitäjä käyttöliittymän lomakkeen sarakkeineen, johon ohjausrakenteiden avulla tulostetaan tietokannasta tulevien tietueiden mukaisesti rivejä. Viimeistään tässä vaiheessa voidaan ymmärtää miksi webohjelmoinnissa suositaan yleensä MVC-mallia. Kuvassa 5 näemme millainen näkymä selaimeen tulostuu.



Kuva 5. All -näkymä selaimessa

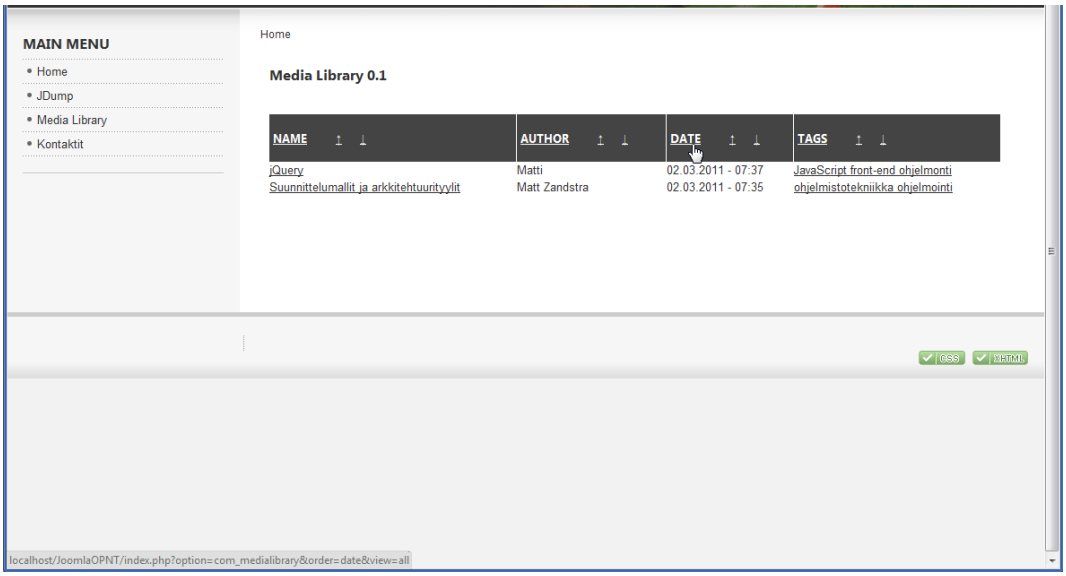
Selaimeen tulostuu siis taulukko mediakirjaston dokumenteista. Näkymän oikeassa yläkulmassa näemme Joomlan käyttöliittymäpainikkeet ”Edit”, ”Delete” ja ”New”. Rastimalla taulukon valintalaatikosta dokumentin voimme suorittaa sille eri toimenpiteitä kun klikkaamme painikkeita. Nämä painikkeet välittävät JavaScriptin avulla lomakkeen submit -käskyjä suorittaa eri metodeita komponentin ohjaimessa. Esimerkiksi Delete -painikkeen avulla kutsutaan ohjaimen remove -metodia, joka välittään kutsun poistaa tietuen tietokannasta tiedostojen käsittelystä vastaavalle mallille.

4.7 Front-end

Komponentin front-end -käyttöliittymän ohjelmointi ei oleellisesti eroa edellä kuvastusta ylläpitäjäkäyttöliittymän kehityksestä. Kumpiinkin pätee samat MVC-arkkitehtuurinmallin periaatteet.

Erona on, että front-end -tiedostot sijoitetaan eri hakemistoon Joomlan hakemistorakenteessa ja näin front-end -käyttöliittymän koodi pidetään erillään back-end -käyttöliittymästä. Tämän komponentin tapauksessa polku on ”Joomla/components/medialibrary”.

Usein käyttäjän näkymässä tarvitaan eri toimintoja kuin ylläpitäjän käyttöliittymässä. Nämä toiminnot voidaan toteuttaa lisäämällä uusia metodeja ohjainluokkaan ja luomalla uusia malliluokkia tiedon käsittelemiseksi.



Kuva 6. Front-end näkymä

Komponenttini front-end malliluokissa sijoitin malliluokkiin metodeja, jolla tietokannasta voidaan hakea tietyn sarakkeen mukaan järjestettyä dataa. Esimerksi mediadokumentit voidaan järjestää päivänmäärän mukaan, nousevasti tai laskevasti kuvan 6 mukaisesti ja kuten komponentin suunnitteluvaiheessa määriteltiin.

Listauksessa 9 on toteuttamani malliluokka ja sen metodi, jolla tämä järjestely toteutetaan.

```
class MediaLibraryModelAll extends JModel
{
	var $data = null;

	function getData(){
		if(empty($this->data)){
			$db = JFactory::getDBO();
			$prefix = $db->getPrefix();
			$query = 'SELECT * FROM ' . $prefix .
'medialibraryitems WHERE published = 1 ORDER BY date';
			$this->data = $this->_getList($query);

		}
		return $this->data;
	}
	function orderBy($order, $dir)
	{
		if(empty($this->data)) {
			$db = JFactory::getDBO();
			$prefix = $db->getPrefix();
			if(!empty($dir)){
				$query = 'SELECT * FROM ' . $prefix .
'medialibraryitems WHERE published = 1 ORDER BY '. $order .' '.
$dir .'';
				$this->data = $this->_getList($query);
			} else {
				$query = 'SELECT * FROM ' . $prefix .
'medialibraryitems WHERE published = 1 ORDER BY '. $order .'';
				$this->data = $this->_getList($query);
			}
		}
		return $this->data;
	}
}
```

Listaus 9. Front-end -näkymän malliluokka

Luokassa on oletusmetodina ”getData”, jolla tietokannasta haetut tietueet järjestetään päivänmäärä sarakkeen mukaan. Toisena metodina on ”OrderBy”, jolla voidaan järjestää metodeja eri järjestykseen sarakkeen mukaan, joko laskevasti tai nousevasti. Parametrit tälle metodille välitetään GET-taulukosta.

Komponentin front-end -käyttöliittymä hyödyntää samoja tietokantatauluja kuin ylläpitäjän näkymä, siksi front-end liittymä pysyy ajan tasalla ylläpitäjän tekemien muutosten kanssa. Kuten näemme listauksesta 9, front-end näkymään haetaan kuitenkin vain ne rivit tietokannasta, joiden published -sarakkeen arvona on 1. Yllä-

pitäjän käyttöliittymässä voidaan asettaa dokumentti published -tilansa osalta joko ”unpublished” tai ”published” tilaan ja tieto tästä tallennetaan ko. sarakkeeseen.

5 YHTEENVETO

5.1 Tutkimustuloksia ja johtopäätöksiä

Joomla -komponenttien ohjelmointi on varsin suorivaviivaista, kun on päästy olio-ohjelmointiparadigman ja MVC-mallin sisäistämiseen liittyvien kysymysten yli. Kun tunnetaan näiden kahden konseptin toiminta, suunnittelu helpottuu ja voidaan keskittyä itse asiaan.

Tehdyn työn ja retrospektion perusteella lähestyisin komponentin kehittämisestä seuraavalla tavalla:

1. Hahmottele komponentin käyttötapaukset ja sovelluslogiikka
2. Suunnittele mitkä malliluokat käsittelevät käyttötapauksiin liittyviä toiminnallisuuksia ja sovelluslogiikkaa
3. Suunnittele kuinka ja millä metodeilla ohjain keskustelee mallien kanssa
4. Mieti miten käyttää Joomlan sovelluskehyksen apu- ja tehdasluokkia toiminnallisuuksien ohjelmoinnissa ja näkymän luomisessa

Ensimmäinen kohta on mielestäni tärkein. On tiedettävä mitä toiminallisuutta komponentilta odotetaan, että voidaan hahmottaa kuinka toiminnallisuus voitaisiin toteuttaa Joomlan ohjelmistokehyksen puitteissa.

Toinen vaihe on mielestäni myös tärkeä, koska ohjelmakoodi on tällä tavalla alusta alkaen hyvin järjesteltyä ja ortogonaalista. Luokkien hahmottelemisessa ei tarvitse olla kovin luova, koska voidaan vain seurata MVC-mallia.

Kolmannessa vaiheessa on hyvä miettiä mitä metodeja kutsutaan ja mitä parametreja ohjain välittää mallille. Samalla on hyvä pohtia, minkälaisia parametreja metodeille välitetään.

Neljännessä vaiheessa, kun on selvillä minkälaista dataa näkymien, ohjaimen ja mallien välillä liikkuu ja kuinka sitä käsitellään, voidaan selvittää kuinka työtä voidaan helpottaa Joomlan sovelluskehyksen työkaluluokilla.

Työtä tehdessäni huomasin usein kuinka juutuin korjaamaan huonosta suunnittelusta johtuvia seikkoja ja toisaalta samaan aikaan yritin lisätä toiminallisuutta käyttämällä Joomla apuluokkia. Tällaisten yksityiskohtien toteuttaminen olisi hyvä jättää viimeiseksi.

5.2 Tavoitteiden saavuttaminen ja parannuskohteita

Mielestäni onnistuin komponentin ohjelmoinnissa varsin hyvin. Sain toteutettua kaikki komponentille asetetut toiminnalliset vaatimukset. Komponentin toiminta vastaa määriteltyjä käyttötapauksia.

Jälkeenpäin kuitenkin näen, että olisin voinut tehdä asioita paremmin ja johdonmukaisemmin. Komponentin ohjelmointi oli kuin matka tuntemattomaan, sillä en tiennyt alussa paljoakaan miten Joomlan sovelluskehys toimii ja kuinka se soveltaa MVC-mallia. En alun perin esimerkiksi käyttänyt malliluokkia hyväkseni ja sijoitin paljon ohjelmakoodia komponentin ohjainluokkaan, joka teki koodista muutosherkkää ja vaikeaselkoista.

Parannettavaa komponentissa olisi siis koodin rakenteen tasolla ja myös joidenkin tietokantakäsittelyjen muodostamisessa. Lisäksi tehtävää olisi vielä virheenkäsittelyiden ohjelmoinnissa. Tuotantokäyttöä varten tulisi vielä miettiä tarkemmin tietoturvanäkökohtia ja tiedostonsiirtoa.

5.3 Loppumietteitä

Työskentely opinnäytetyöni kanssa on ollut antoisaa ja opettavaista. On ollut innostavaa nähdä miten olio-ohjelmointi ja sovellusarkkitehtuuri pelaavat yhteen sovelluksen pinnan alla.

Olio-ohjelmoinnin ja sovellusarkkitehtuurin teorian lukeminen ohjelmoinnin lomassa on antanut lisää itseluottamusta kokeilemaan muitakin kieliä kuin PHP:tä. Periaatteet pysyvät samoina kielestä kieleen vaikka syntaksi vaihtelee.

Mielenkiintoista on ollut huomata, että kun sisäistää jonkin järjestelmän toimintaa pintaa syvemmältä, niin toisen järjestelmän oppiminen helpottuu. Useat avoimen

lähdekoodin verkkokauppasovellukset, sisällönhallintajärjestelmät ja websovelluskehykset, kuten esimerkiksi Zend Framework, CakePHP, CodeIgniter, hyödyntävät MVC-mallia. Uskon, että tulevaisuudessa tulen perehtymään edellä mainittuihin, koska valmiin sovelluskehyksen avulla voi rakentaa nopeasti mielenkiintoisia websovelluksia.

LÄHTEET

1. Kirjallisuus

LeBlanc, Joseph L. (2008). Learning Joomla! 1.5 – Extension Development. Birmingham: Packt Publishing Ltd.

Zandstra, Matt (2008). PHP Objects, Patterns and Practice. Berkeley: Apress.

Koskimies, Kai. Mikkonen, Tommi (2005). Ohjelmistoarkkitehtuurit. Talentum.

Koskimies, Kai (2000). Oliokirja. Satku – Kauppakaari.

2. Elektroniset julkaisut

Wikipedia (2011). Joomla [viitattu: 13.1.2011]. Saatavilla internetissä: <URL: http://en.wikipedia.org/wiki/Joomla>

Joomla (2011). What is Joomla? [viitattu: 13.1.2011]. Saatavilla internetissä: <URL: http://www.joomla.org/about-joomla.html>

Joomla Documentation (2011). Developing a Model-View-Controller Component, [viitattu: 13.2011]. Saatavilla internetissä: <ULR: http://docs.joomla.org/Developing_a_Model-View-Controller_Component_-_Part_1>

Wikipedia (2011). Architectural Pattern (Computer Science), [viitattu 11.1.2011]. Saatavilla internetissä: <URL: http://en.wikipedia.org/wiki/Architectural_pattern_%28computer_science%29>

Wikipedia (2011). Model-View-Controller, [viitattu 11.1.2011]. Saatavilla internetissä: <URL: http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

Wikipedia (2011). PHP, [viitattu 15.1.2011]. Saatavilla internetissä: <URL: http://en.wikipedia.org/wiki/Php>

Trygve M. H. Reenskaug (2011), MVC [viitattu 10.3.2011]. Saatavilla internetis-sä: <URL: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html>

Developer.com, Jason Gilmore (2010). Fat Models and Skinny Controllers Bring Out the Best in Your MVC Framework, [viitattu 21.2.2010]. Saatavilla internetis-sä: <URL: http://www.developer.com/design/article.php/3856246/Fat-Models-and-Skinny--Controllers-Bring-Out-the-Best-in-Your-MVC-Framework.htm>