

Zaniar Mohammadi

# Open Graph Markup -moduulin kehitys Magento-verkkokauppa-alustalla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

3.11.2016

Tekijä(t) Otsikko  Sivumäärä Aika	Zaniar Mohammadi Open Graph Markup -moduulin kehitys Magento verkko- kauppa-alustalla  56 sivua + 1 liitettä 3.11.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumismuutostohto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander Toimitusjohtaja Henri Halmelahti
<p>Insinöörityön tavoitteena oli perehtyä Magento-verkkokauppa-alustan arkkitehtuuriin ja luoda Magento-moduuli, jolla verkkokaupan omistajat pystyvät lisäämään Open Graph Markup -dataa verkkokauppaansa. Magento-moduulit tehtiin Lamia Oy -yritykselle.</p> <p>Insinöörityössä toteutettiin Open Graph Markup -moduuli, joka mahdollistaa Open Graph Markup -tietojen luomisen verkkokaupan tuotteille, kategorioille ja CMS-sivuille hallintapaneelin kautta. Open Graph Markup -tiedot antavat mahdollisuuden verkkokaupan ylläpitäjien hallita, mitä tietoja näytetään tuotteista, kategorioista tai CMS-sivuista, kun ne jaetaan Facebookissa.</p> <p>Insinöörityön aikana saatiin halutun moduulin logiikka toteutettua lähes kokonaan. Insinöörityön aikana saatiin myös laaja kuva siitä, miten Magento-verkkokauppa-alustan arkkitehtuuri toimii, mitä suunnittelumalleja se käyttää ja miten suunnittelumallit toimivat.</p> <p>Johtopäätöksenä huomattiin, että Magento-verkkokauppa-alusta on vahvassa asemassa markkinoilla. Vaikka alustalla ohjelmointi on hankalaa, se antaa työkaluna erittäin laajat mahdollisuudet ohjelmoijille ja verkkokaupan ylläpitäjille verkkokaupan kehittämiseen ja laajentamiseen. Mikäli haluaa kehittyneempiä ominaisuuksia verkkokauppaan, on ylläpitäjien otettava yhteyttä Magento-ohjelmoijiin.</p> <p>Jatkosuunnitelmana on tarkoitus kehittää Open Graph Markup -moduuli valmiiksi ja sen jälkeen integroida kyseinen moduuli tuleviin projekteihin.</p>	
Avainsanat	Magento, verkkokauppa, suunnittelumalli, Open Graph Markup

Author(s) Title	Zaniar Mohammadi Open Graph Markup Module Development with Magento eCommerce Platform
Number of Pages Date	56 pages + 1 appendix 3 November 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Henri Halmelahti, Chief Executive Officer
<p>The aim of this thesis was to understand Magento ecommerce platform and its architecture. A custom Magento module was created to help achieve a better understanding how to develop with Magento. The module was created for Lamia Oy.</p> <p>Open Graph Markup Magento module was created enabling ecommerce owners to create and modify custom Open Graph Markup data for products, categories and CMS pages via the administrator panel.</p> <p>The custom Magento module was almost completely finished and the study gave a better understanding on how the Magento ecommerce platform works, what different design pattern it takes advantage of and what they provide.</p> <p>In conclusion, Magento is an open source ecommerce platform and although it is quite hard to understand how it works and how to develop custom functionality with it, it gives a lot of room for customization and scalability. Administrators can modify the look of their ecommerce site but if they want custom functionality for their site, they still need to contact a Magento developer.</p>	
Keywords	Magento, design pattern, ecommerce, Open Graph Markup

## Sisälllys

### Lyhenteet

1	Johdanto	1
2	Magento	1
2.1	Magenton historia	1
2.2	Magenton eri versiot	2
3	Verkkokauppa-alustojen vertailu	3
4	Magenton suunnittelumallit	5
4.1	MVC	5
4.2	Front Controller	9
4.3	Factory	9
4.4	Registry	11
4.5	Singleton	12
4.6	Object Pool	13
4.7	Iterator	14
4.8	Lazy Loading	15
4.9	Service Locator	15
4.10	Module	16
4.11	Observer	17
4.12	EAV-malli	18
5	Magenton arkkitehtuuri	20
6	Open Graph Markup -tekniikka	21
7	Open Graph Markup -moduulin toteutus	22
7.1	Moduulin kansiorakenteen luonti ja konfigurointi	26
7.2	Uusien attribuuttien lisäys	27
7.3	Sektioiden luonti hallintapaneelin järjestelmäasetuksiin	29
7.4	Asetusten lisäys sektioihin ja oikeuksien konfigurointi sektioita varten	31
7.5	Model-luokkien toteutus	33
7.6	Block-luokkien toteutus	36

7.7	Javascript-ponnahdusikkunan ja template-tiedoston toteutus	39
7.8	Observer-tapahtumien konfigurointi	44
7.9	Käsittelyluokkien toteutus	45
7.10	Testaaminen	48
8	Johtopäätökset ja yhteenveto	50
	Lähteet	52
	Liitteet	
	Liite 1. Config.xml-tiedoston toteutus	

## Lyhenteet

CMS	Content management system. Sisällönhallintajärjestelmä on nimitys tietojärjestelmälle, joka on verkkosivuston tai palvelun sisällönhallintaan tarkoitettu järjestelmä.
MVC	Model-view-controller. Ohjelmistoarkkitehtuurityyli tai suunnittelumalli, jolla pyritään erittelemään näkymään, ohjelman logiikkaan ja näiden kahden kommunikaatioon liittyvät osat omiksi kokonaisuuksiksiin.
UML	Unified modeling language. Standardoitu graafinen mallinnuskieli, jonka avulla voidaan kuvata erilaisten sovellusten ja järjestelmien rakennetta.
XML	Extensible markup language. Tietty merkintäkielistandardi, jonka avulla voidaan siirtää tietoa eri järjestelmien välillä.
API	Application programming interface. Ohjelmointirajapinnan yhtenä päätarkoituksena on antaa ohjelmoijalle valmiit ja yleisimmät työkalut ohjelman kehitystä varten.

## 1 Johdanto

Magento on PHP-ohjelmointikielellä Zend-sovelluskehityksen avulla kirjoitettu avoimen lähdekoodin verkkokauppa-alusta. Magento on tällä hetkellä maailman johtavin verkkokauppa-alusta, jota käyttävät muun muassa Rosetta Stone ja urheilutarvikkeita myyvä Nike. Magenton vahvuuksia ovat helposti lisättävät moduulit ja teemat, joita verkkokaupan omistajat voivat verkkokaupassaan käyttää. Magento on suunniteltu siten, että käyttäjät voivat käyttää sitä, vaikka eivät sen kummempin osaa ohjelmoida. Mikäli tarvitsee lisäominaisuuksia omien tarpeidensa mukaan, niin on käännyttävä kuitenkin ohjelmoijan puoleen. [1.]

Insinöörityön tavoitteena on perehtyä Magento-verkkokauppa-alustaan, sen tarjoamiin ominaisuuksiin sekä vertailla sitä muihin markkinoilla oleviin verkkokauppa-alustoihin. Raportissa käydään läpi Magenton arkkitehtuuri ja kaikki suunnittelumallit, joita Magento käyttää hyväkseen. Lopuksi raportissa käydään vielä läpi insinöörityön aikana ohjelmoituun Magento-moduulin kehitykseen liittyvät ongelmat ja ratkaisut.

Insinöörityössä tehty moduuli antaa mahdollisuuden asettaa Open Graph -merkinnät verkkokaupan tuotteille, kategorioille ja CMS-sivuille. Open Graph -merkintöjen avulla käyttäjä voi kontrolloida, miten tuotteet näkyvät, mikä osa on tuotteen nimeä, kuvausta tai kuvaa Facebookissa. Ilman niitä Facebook Crawler yrittää parhaansa mukaan arvata parhaat asetukset tuotteelle. Moduuli tehtiin Lamia Oy -yritykselle.

## 2 Magento

### 2.1 Magenton historia

Magenton verkkokauppa-alustan kehitti Yhdysvaltalainen Varien Inc. -niminen yritys, jonka toimitusjohtaja oli Roy Rubin. Nykyään Varien Inc. -yritys tunnetaan Magento Inc. nimellä. Magenton alkuperäisen nimen piti olla Bento, mutta se oli jo tavaramerkitty nimi. Ennen Magenton kehitystä he käyttivät OsCommerce-verkkokauppa-alustaa. OsCommerce on myös ilmainen avoimen lähdekoodin verkkokauppa-alusta kuten Magento. He huomasivat kuitenkin monet rajoitukset, joita OsCommerce-verkkokauppa-alustassa oli ja päättivät luoda oman verkkokauppa-alustansa. OsCommercen rajoituksia olivat muun

muassa vakauden, ominaisuuksien ja joustavuuden puute. Varien Inc. alun perin suunnitteli aloittavansa Magenton kehityksen OsCommercen pohjalta mutta päättyi kuitenkin kehittämään oman kokonaisuutensa Magentosta. [2.]

Magenton kehitys alkoi vuonna 2007, ja ensimmäinen beta-versio julkaistiin samana vuonna elokuussa. Suuren suosion jälkeen Varien julkaisi version 1.0 avoimena lähdekoodina jo maaliskuussa vuonna 2008. Vuonna 2011 alkupuolella Ebay omisti noin 50 % yrityksestä ja myöhemmin samana vuonna Ebay omisti Magentosta 100 %. [2.]

Magento on nykyäänkin suosittu ja maailman johtavin verkkokauppa-alusta. Magento julkaisi 2015 vuoden lopussa 2.0-version, joka toi parannuksia muun muassa tietokannan indeksointiin, välimuistin hallintaan ja sallii samanaikaisia ylläpitokäyttäjiä Magenton hallintapaneelissa. Magento 1.x -version päivitys Magento 2 -versioon on kuitenkin hankala ja suuri prosessi, muun muassa kansiorakenteen muutosten takia. [2.]

## 2.2 Magenton eri versiot

Tällä hetkellä Magentosta on saatavilla kolme eri versiota. Magento Community Edition, on ilmainen kehittäjille tarkoitettu avoimen lähdekoodin alusta. Suuren suosion ja menestyksen myötä suuremmat yritykset alkoivat käyttää Magento verkkokauppa-alustaa ja tästä syystä Magento Enterprise Edition -version kehitys sai alkunsa vuonna 2009. Enterprise Edition on huomattavasti Community Editionia nopeampi esimerkiksi tilanteissa, jossa pitää lisätä monimutkaisempia tuotteita verkkokauppaan. Monimutkaiset tuotteet ovat muokattavat tuotteet ja ryhmätuotteet. Muokattavien tuotteiden avulla voidaan luoda nopeasti eri attribuutteja omaavia yksinkertaisia tuotteita. Attribuutit puolestaan ovat tuotteen eri ominaisuuksia esimerkiksi tuotteen väri ja koko. Ryhmätuotteiden avulla voidaan myydä yksinkertaisia tuotteita joukkona kiinteällä hinnalla. Community Editionilla voi mennä monimutkaisten tuotteiden lisäysprosessiin noin 10 - 20 minuuttia. Enterprise Editionin käyttäjillä on myös saatavilla tekninen tuki. Taulukossa 1 näkyy tarkemmin osa Enterprise- ja Community Editionin eroista. Enterprise Edition on tarkoitettu suurimmille yrityksille, jotka käyttävät Magento-verkkokauppaa. [2.]



Taulukko 1. Magento Community Editionin ja Enterprise Editionin ominaisuuksia [2.].

Ominaisuudet	Community Edition	Enterprise Edition
Asennus	Ilmainen	15000 dollaria vuodessa
Tukipalvelu	Ei ole saatavilla	Saatavilla
Bloggaus	Saatavilla	Saatavilla
CMS	Saatavilla	Saatavilla
Palautusten käsittely	Sisäänrakennettu lisäosa	Oletusominaisuus
Toivelista	Sisäänrakennettu lisäosa	Oletusominaisuus

Magento Enterprise Cloud Edition on sama kuin Enterprise Edition mutta tarjoaa Magenton PaaS-pilvipalveluna niin, ettei käyttäjien tarvitse huolehtia palvelimen tilasta ja huolehtimisesta vaan se hoituu pilvessä ja skaalautuu tarpeen mukaan. PaaS (Platform as a Service) voi olla esimerkiksi tyhjä virtuaalikone pilvipalveluna, johon on asennettu käyttöjärjestelmä ja muutama komponentti tai kirjasto kuten PHP ja MySQL. Magento Enterprise Cloud Edition on tarkoitettu keskisuurille tai suurille yrityksille kuten myös Enterprise Edition. [2.]

Näistä kolmesta eri versiosta suosituin on kuitenkin ilmainen Magento Community Edition monesta eri syystä, mistä yksi suurimmista syistä on sen avoimen lähdekoodin tarjonta. Magento on avoimen lähdekoodin verkkokauppa-alusta. Tämä tarkoittaa sitä, että kehittäjät voivat ladata lähdekoodin käyttöönsä ja kehittää sen päälle lisää ominaisuuksia omien tarpeidensa mukaan. Kehittäjät näkevät myös, miten Magenton eri osat on ohjelmoitu, mikä auttaa heitä ymmärtämään, kuinka Magento toimii ja kehittämään omia moduuleitaan Magenton päälle. [2.]

### 3 Verkkokauppa-alustojen vertailu

Tutkin muita verkkokauppa-alustavaihtoehtoja ja päädyin vertailemaan BigCommerce- ja Shopify-alustoja sillä tutkiessani eri vaihtoehtoja nämä ilmentyivät hyvin usein. Kyseisten verkkokauppa-alustojen suosio on myös nousussa [3.]. BigCommerce ja Shopify ovat suoraan selaimelta käytettäviä verkkokauppa-alustoja eikä niitä tarvitse erikseen asentaa toisin kuin Magento [4.]. Magento vaatii monta eri työkalua ja konfiguraatiota

ennen kuin sen saa käyttökuntoon, eivätkä kaikki peruskäyttäjät tästä syystä osaa asentaa Magentoa toimivaksi. Tästä syystä BigCommerce ja Shopify ovat käyttäjäystävällisempiä kuin Magento. [5.]

Shopifyn aloituspaketti maksaa 14 dollaria kuukaudessa ja mahdollistaa ainoastaan 25 tuotetta verkkokauppaan. BigCommerce maksaa noin 30 dollaria kuukaudessa, joka on melkein 2 kertaa enemmän kuin Shopify [4]. Magento puolestaan mahdollistaa rajattoman määrän tuotteita verkkokauppaan ja on ilmainen, mikäli käyttää Community Editionia. [4.]

Kaikki kolme verkkokauppa-alustaa tarjoavat laajan kokoelman erilaisia teemoja. Ilmaisia Shopify- ja BigCommerce-teemoja on paljon vähemmän Magentoon verrattuna, ja Shopify'n maksulliset teemat alkavat 80 dollarista, kun Magenton maksulliset teemat alkavat yhdestä dollarista [5]. BigCommerce-alustalla on vain kahdeksan ilmaista teemaa, joka on huomattavasti vähemmän kuin Shopifylla ja Magentolla. BigCommercen maksullisetkin teemat ovat huomattavasti kalliimpia kuin Magenton tai Shopify'n teemat [4]. Teemat maksavat satoja euroja. Taulukossa 2 näkyy verkkokauppa-alustojen vertailu keskenään.

Taulukko 2. Verkkokauppa-alustojen vertailu [5.].

Ominaisuudet	Magento	Shopify	BigCommerce
Hinta per kuukausi	Ilmainen	9 \$ - 179 \$	29.95 \$ - 199.95 \$
Rajaton määrä ominaisuuksia tuotteilla	Kyllä	Ei	Kyllä
Käytettävyys	7/10	9/10	10/10
Langaton kaupankäynti	Saatavilla	Saatavilla	Saatavilla

Shopify ja BigCommerce ovat monessa osassa alkeellisempia kuin Magento, mutta se ei kuitenkaan tarkoita sitä, että ne olisivat huonoja. Shopify- ja BigCommerce-verkkokauppoja kannattaa käyttää uusien aloittavien yritysten, joilla ei välttämättä ole resursseja Magenton asennusta ja ylläpitoa varten. Magenton hallintapaneeli on myös huomattavasti vaikeampi sisäistää kuin Shopify tai BigCommerce. Shopifylla on helpompaa

muokata eri teemoja ja näkymiä oman tarpeensa mukaan Magentoan verrattuna. Tästä syystä se on suosittu kuin Magento. Pitkällä tähtäimellä kuitenkin suurien yritysten kannattaa valita Magento-verkkokauppa-alusta sen laajennettavuuden ja erittäin laajan muokattavuuden takia. Magento Community Edition on myös avoimena lähdekoodina, jota monet yritykset arvostavat. [6.]

## 4 Magenton suunnittelumallit

Suunnittelumallit ovat kehittäjille tarkoitettuja ohjelmointikielistä riippumattomia ohjelmointimalleja, joiden tarkoituksena on ratkaista yleiset ohjelmointiongelmien niin, että ne ovat helposti muokattavia ja uudelleenkäytettäviä ratkaisuja. Suunnittelumallit ovat kuitenkin suuntaa antavia malleja ja usein malleista poiketaan hiukan omien tarpeiden mukaan säilyttämällä kuitenkin mallin pääperiaate. [7.]

Suunnittelumallit jakautuvat kolmeen eri ryhmään, jotka ovat luontimallit, rakennemallit ja käyttäytymismallit. Suunnittelumalleja on kuitenkin monta erilaista eikä kaikkia ole jaettu näihin kolmeen eri suunnittelumalliryhmään. Rakennemallit käsittelevät usein eri kokonaisuuksien tai objektien keskeistä relaatiota. Rakennemallien tarkoituksena on helpottaa eri kokonaisuuksien keskinäistä toimintaa näiden relaatioiden avulla. Luontimallin tarkoituksena on tarjota tilanteeseen riippuvaa helppoa tapaa luoda olioita. Käyttäytymismalleja käytetään usein tilanteissa, jossa halutaan toteuttaa joustava tapa eri kokonaisuuksien välisessä kommunikaatiossa. [7.]

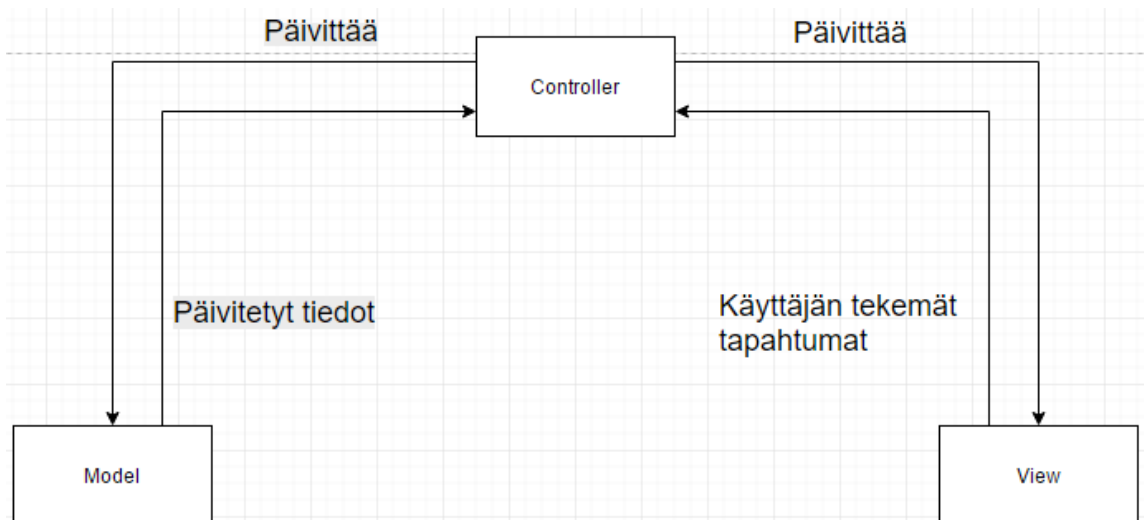
### 4.1 MVC

MVC-malli on yksi käytetyimmistä suunnittelumalleista. MVC-mallia ei ole varsinaisesti jaettu mihinkään kolmeen suunnittelumalliryhmään mutta usein sitä käsitellään rakennemallina. MVC-malli jakautuu kolmeen tärkeään osaan Model, View ja Controller tai suomeksi malli, näkymä ja kontrolleri, joiden tarkoituksena on jakaa koodi kolmeen toisistaan riippumattomaan osaan. Tämä jako helpottaa koodin luettavuutta, muokattavuutta ja uudelleenkäytettävyyttä. [8.]

MVC-mallin Model on tietty osa ohjelmasta, joka sisältää joitain perusfunktioita ja muuttujia, joita halutaan käsitellä. Model-osalla ei kuitenkaan ole mitään tekemistä siihen, miltä sovellus näyttää. [8.]

View-osa huolehtii siitä, miltä sovellus näyttää ja miten Model-osien tiedot näytetään sovelluksen tai verkkosivun käyttäjälle. Kaikki näkymän tapahtumat lähetetään kontrollerille, joka puolestaan käsittelee ja lähettää siitä tiedon mallille tarvittaessa. View ei kuitenkaan tiedä Model-osan olemassaolosta, ja tämä mahdollistaa sen, että malleja voidaan luoda ja liittää näkymään vaivattomasti. Pääohjelmassa luodaan usein mallit ja näkymäinstanssit ensin, jonka jälkeen ne annetaan parametrina kontrollerin luonnin yhteydessä. Näin saadaan näiden kolmen osan välinen kommunikaatio toteutettua. [8.]

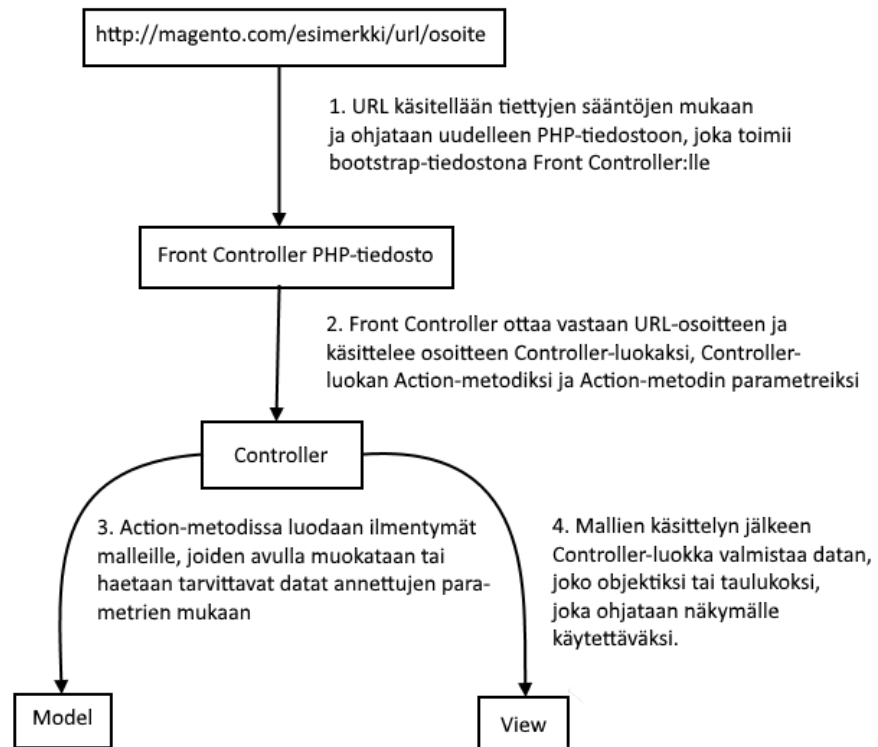
Controller-osa huolehtii View- ja Model-osien välisestä kommunikaatiosta. Controller välittää mallin tiedot näkymälle näytettäväksi. Controller-osa ei vaikuta mitenkään siihen, miten malleja näytetään tai käsittelee ja muuta suoraan mallin tietoja, kuten kuvassa 1 nähdään. [8.]



Kuva 1. Perinteinen MVC-malli [7.].

MVC on kuitenkin vain suunnittelumalli ja siitä löytyy monta erilaista variaatiota, jossa näkymien, mallien ja kontrollereiden välistä kommunikaatiota ei ole eroteltu niin kuin perinteisessä mallissa. Useat PHP MVC -sovelluskehikset käsittelevät URL-osoitetta niin, että tunnistavat osoitteesta Controller-osion ja Action-osion, joka on kontrollerissa suoritettavan funktion nimi. Näillä tiedoilla sovelluskehitys löytää siihen liittyvän Controller-luo-

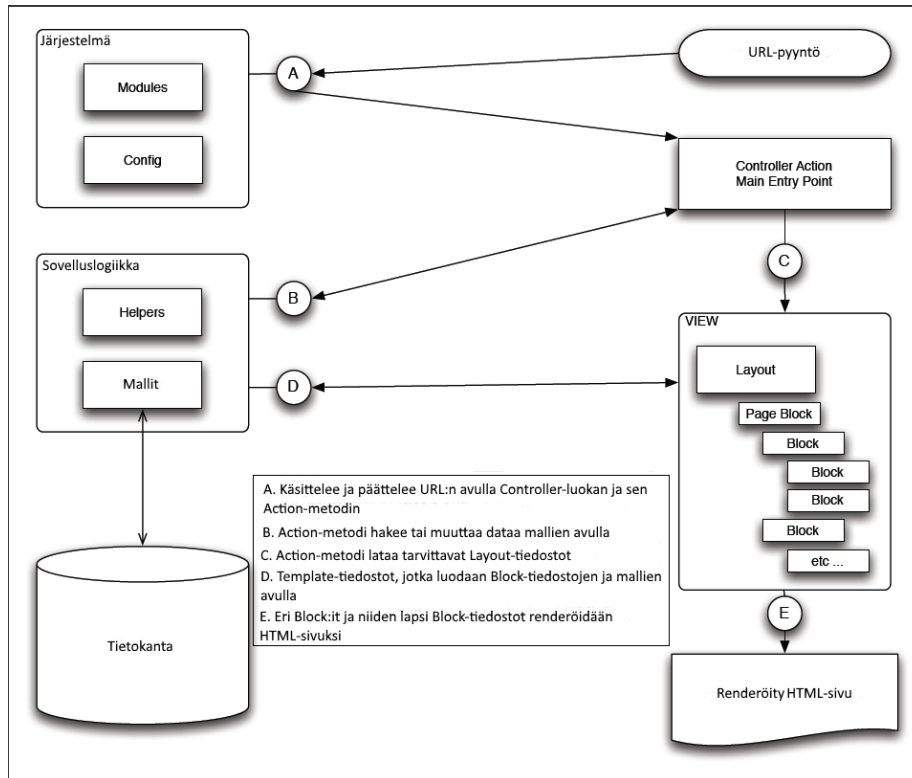
kan ja Action-funktion. Lopuksi Action-funktio käsittelee malleja joillain tavalla, ja kontrolleri ohjaa käyttäjän oikealle näkymälle. Näin saadaan helposti luotua suuria verkkosivuja ja jakamaan yhteenkuuluvat sivustot yhteen. Näin saadaan myös helposti muokattavaa ja laajennettavaa ohjelmakoodia. Kuvassa 2 näkyy paremmin, miten useat PHP:n MVC-sovelluskehukset käsittelevät URL-osoitteita ja ohjaavat käyttäjän oikeaan näkymään. [9.]



Kuva 2. URL-osoitteen käsittely PHP:n MVC-sovelluskehyksissä [9].

Kuvassa 2 Bootstrap-tiedosto tarkoittaa tiedostoa, joka ajetaan ensimmäisenä tiedostona, kun sovellus käynnistetään. Magenton MVC-malli on kuvan 2 mallista tehty vielä abstraktimmalle tasolle, jonka tarkoituksena on parantaa koodin uudelleenkäytettävyyttä entisestään. Magenton MVC-mallissa Magento tunnistaa ensin URL-osoitteesta kontrolleriosan. Tämän jälkeen Magento iteroi ja yrittää yhdistää URL-osoitteen seuraavan osoitteen johonkin kontrollerin funktioon eli tässä tapauksessa Action-funktioon. Tämän takia kontrollerissa funktioiden nimeämiskäytäntönä nimet aina lopetetaan Action-sanaan esimerkiksi `indexAction`. Kun Action-funktio on löytynyt, niin kontrolleri käsittelee malleja ja samalla lataa oikean layoutin. Layout puolestaan käsittelee ja lataa oikeat

Block-tiedostot näkymää varten. Block-tiedostot hakevat malleista tarvittavat tiedot ja valmistelevat nämä tiedot HTML-koodiksi template-tiedostoja varten, ne muodostavat näkymän ja sisältävät ainoastaan PHP -ja HTML-koodia. Template-tiedostojen tarkoituksena on ainoastaan renderöidä näkymä, eivätkä ne sisällä monimutkaista logiikkaa. Kuvassa 3 tarkempi kuvaus Magenton MVC-mallista. [9.]



Kuva 3. Magenton MVC-malli [10].

Magento käyttää konfiguraatiopohjaista MVC-mallia. Perinteisessä MVC-mallissa on hyvin rajoitettu tiedostojen nimeämiskäytäntö sekä niiden sijainti on hyvin tärkeää, jotta järjestelmä tunnistaa ja osaa käyttää tiedostoja oikein. Tämä rajoittaa kehittäjää luomaan vähemmän optimoidumpaa ja modulaarisempaa sovellusta kuin konfiguraatiopohjaisella järjestelmällä. Konfiguraatiopohjaisessa MVC-mallissa luodaan erikseen konfiguraatio-tiedostoja, jotka ovat Magenton tapauksessa XML-muodossa. Näillä konfiguraatiotiedostoilla kehittäjä voi erikseen määrätä, missä eri tiedostot sijaitsevat ja millä luokan nimellä. Konfiguraatiopohjaisen MVC-mallin avulla kehittäjät voivat luoda huomattavasti modulaarisempaa ja helposti ylläpidettävää koodia. [11.]

## 4.2 Front Controller

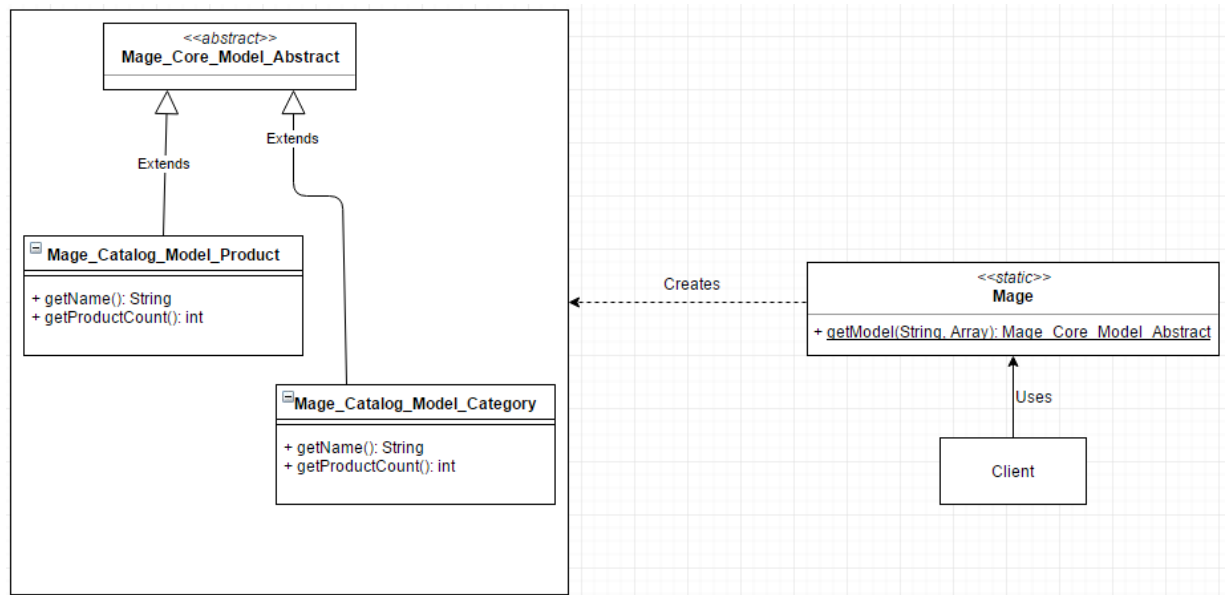
Front Controller -suunnittelumallia käytetään usein verkkosovelluksissa. Front Controller -rakennemallin tarkoituksena on helpottaa luomaan ja yhdistämään yhteenkuuluvia verkkosivuja verkkosovelluksessa jakamalla URL-osoite eri osiin kuten kontrolleriin, tiettyyn kontrollerissa sijaitsevaan funktioon ja eri parametreihin. Suunnittelumalli myös helpottaa koodin luettavuutta ja ylläpidettävyyttä. [12.]

Magenton Front Controller-suunnittelumallilla on yksi `index.php`-niminen sisääntulopiste kaikille sen kutsuille. Magento tarjoaa `Mage_Core_Controller_Front_Action`-nimisen luokan, jota moduulin Controller-luokat perivät. Magentossa URL-osoite jaetaan router-, controller- ja action-nimisiin osiin. Router-osa on määritelty Magento moduulin XML-konfiguraatiodokumentissa, Controller-osa on luokka, jota käytetään sivupyynnön käsittelyä varten ja Action-osa on funktio controller-luokassa, joka käsittelee pyynnön annettujen parametrien avulla. [13.]

Front Controller -mallin prosessi toimii vastaavasti Magentossa esimerkiksi <http://www.magento.com/asiakas/tili/kirjaudu>-osoitteessa. Osoitteen perusosa on `www.magento.com`, `/asiakas/` toimii Router-osana Asiakas-nimiselle moduulille, `/tili/` toimii Controller-luokkana `TiliController.php`-nimisessä tiedostossa ja kutsuu `Mage_Asiakas_TiliController.php`-nimistä luokkaa ja lopuksi `/kirjaudu/` on varsinainen funktio, jota kutsutaan `TiliController.php`-tiedostossa. Action-funktioiden nimet on Magentossa aina päätyttävä sanaan `action`. Esimerkissä kutsuttu funktio olisi `kirjauduAction()`-niminen funktio. Kaikki loput osat mitä URL-osoite sisältää käsiteltäisiin parametreina avain- ja arvopareina. [13.]

## 4.3 Factory

Factory on luontimalli, jonka tarkoituksena on määritellä rajapinta eri olioiden luomiseen kuitenkin niin, että olion luovalla luokalla on mahdollisuus päättää, mitä olion ilmentymää luodaan. Tämä antaa abstraktin tason ohjelmalle, jota on helpompi laajentaa ja testata. Kuvassa 4 nähdään UML-kaavio Factory-suunnittelumallin toteutuksesta. [14.]



Kuva 4. Factory-suunnittelumallin toteutus [14.]

Kuvassa 4 nähdään Mage\_Core\_Model\_Abstract-abstraktiluokka, jonka Mage\_Catalog\_Model\_Product ja Mage\_Catalog\_Model\_Category perivät. Staattisen Mage-luokan getModel()-funktio voi luoda ilmentymiä. Se minkä luokan ilmentymää luodaan, annetaan merkkijonona funktion parametrina. Toisena parametrina voidaan antaa assosiatii- vinen taulukko, joka sisältää argumentteja avain ja arvo pareina. Mage-luokka on staattinen luokka, jota voidaan kutsua missä tahansa luokassa. Mage-luokan avulla luodaan ilmentymiä ja päästään käsiksi muihin hyödyllisiin funktioihin. [14.]

Ennen kuin ilmentymiä voidaan luoda esimerkiksi malleista, on määriteltävä etc-kansi- ossa sijaitsevaan config.xml-tiedostoon mistä kyseiset malli-luokat löytyvät. Kuvassa 5 nähdään malli-luokkien määrittäminen config.xml-tiedostossa. [15.]

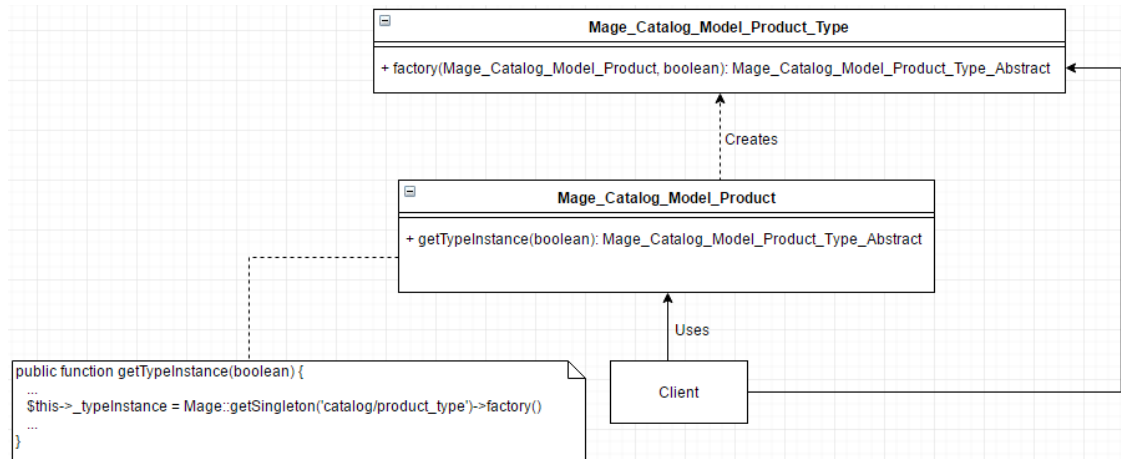
```

<config>
  <global>
    <models>
      <abstractname>
        <class>Nimiavaruus_ModuulinNimi_MallinNimi</class>
      </abstractname>
    </models>
  </global>
</config>
  
```

Kuva 5. Malli-luokkien määrittäminen config.xml-tiedostossa [15.]



Magentossa on myös Prototype-malli. Prototype-malli ei kuitenkaan vastaa Gamman määrittelemää Prototype-mallia, jonka pääperiaatteena on luoda instansseista kloonveja ja näin myös parantaa suorituskykyä. Kuvassa 6. näkyy Magenton Prototype-malli. Kyseisen mallin avulla saadaan Factory-metodi.

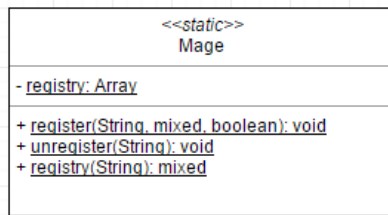


Kuva 6. Magenton Prototype-malli.

Kuvassa 6 näkyy kuinka Magento hyödyntää Factory-mallia. Kyseistä toteutusta kuitenkin kutsutaan Magentossa Prototype-malliksi. Tätä ei saa kuitenkaan sekoittaa Gamman määrittelemään Prototype-malliin. Kuvassa 6 `getTypeInstance()`-funktion boolean-parametrilla määritellään, jos halutaan luoda Singleton instanssi. Funktio `getTypeInstance()` luo `Mage_Catalog_Model_Product_Type` ja kutsuu sen `factory()`-funktiota, jossa itse instanssi luodaan ja palautetaan. Funktion `getTypeInstance()` avulla saadaan muun muassa selville sen, onko tuote muokattava tuote vai yksinkertainen tuote.

#### 4.4 Registry

Registry-suunnittelumalli on hyvin yksinkertainen toteutukseltaan. Registry-malli ei kuulu mihinkään suunnittelumalliryhmään. Registry-suunnittelumalli on hyvin hyödyllinen tilanteissa, jossa ohjelmoija haluaa käyttää yhtä ilmentymää monessa eri paikassa projektissaan mutta kuitenkin luomatta siitä globaalia muuttujaa. Registry-suunnittelumalli mahdollistaa tämän. Suunnittelumalli toimii säiliönä objekteille, johon ohjelmoija voi asettaa eri objekteja avain- ja arvopareina ja myöhemmin hakea tallennetun objektin Registry-luokasta. Kuvassa 7 on Magenton toteutus Registry-mallista [16.]

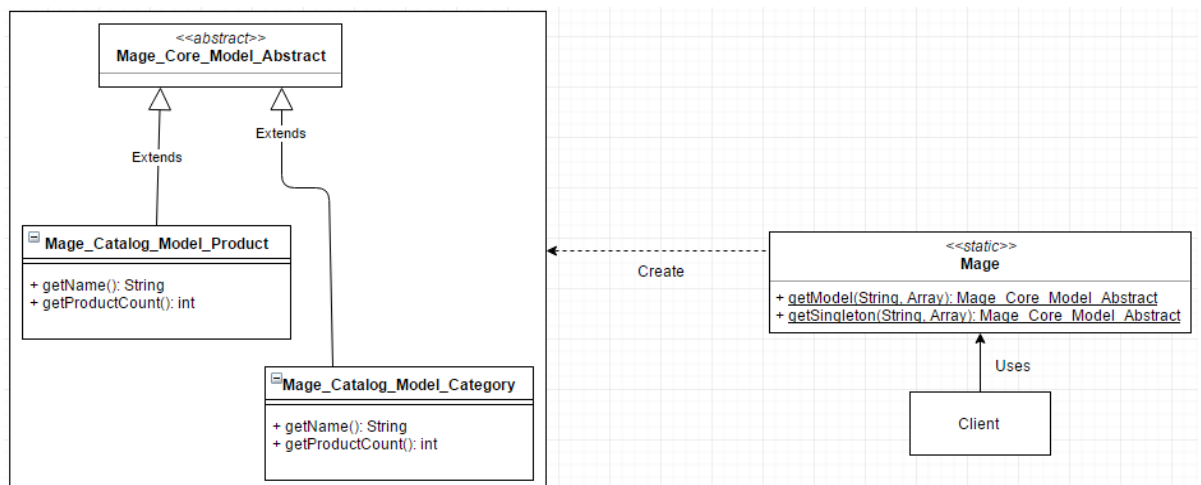


Kuva 7. Magenton toteutus Registry-mallista.

Kuvan 7 registry-muuttuja on assosiatiivinen taulukko, jonka avain- ja arvoparin avaimena on merkkijono ja arvona on tallennettu objekti. Magentossa objekteja voidaan tallentaa Registry-luokkaan kutsumalla funktiota `Mage::register('olionId', olio)`. Sen jälkeen kun Registry-luokkaan on tallennettu olio, se voidaan hakea funktiolla `Mage::registry('olionId')`. Lopuksi Registry-luokasta voidaan poistaa tallennettu olio kutsumalla funktiota `Mage::unregister('olionId')`. Register-funktiolla on myös kolmas boolean-tyyppinen parametri. Kyseisen parametrin avulla voidaan estää samalla avaimella arvon asettamisen registry-muuttujaan. [17.]

#### 4.5 Singleton

Singleton on luontimalli, jonka tarkoituksena on varmistaa, että jostain luokasta tehdään vain ja ainoastaan yksi ilmentymä. Suunnittelumallin avulla ohjelmoijan ei tarvitse itse varmistaa sitä, onko luokasta varmasti luotu vain yksi ilmentymä. Kuvassa 8 näkyy Singleton-suunnittelumallin toteutus. [18.]

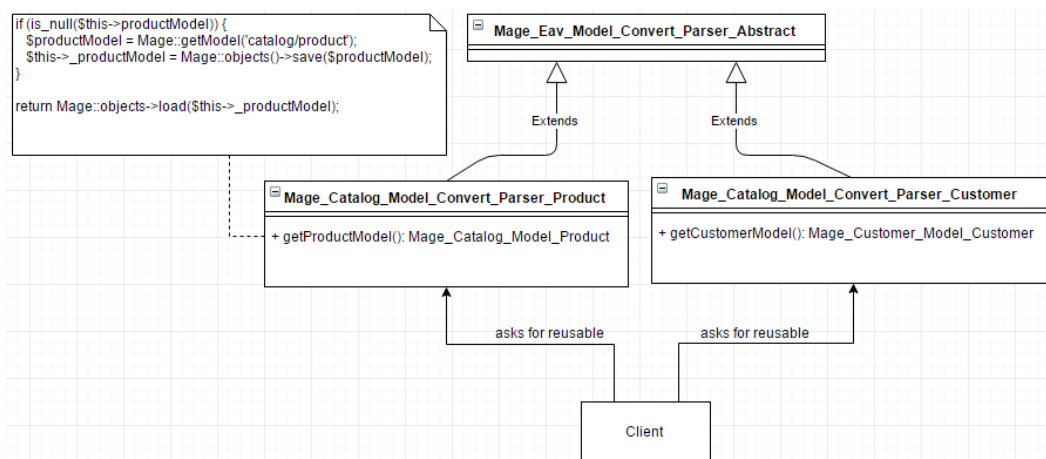


Kuva 8. Magenton Singleton-mallin toteutus.[18.]

Magentossa Singleton-malli toimii hyvin samalla kuin Factory Method -malli. Funktion `getModel()` sijaan kutsutaan `Mage::getSingleton()`-funktiota, joka varmistaa sen, että luokasta luodaan vain yksi ilmentymä. Mikäli ilmentymää ei ole vielä luotu, niin se luodaan ja palautetaan funktion kutsujalle. [19.]

#### 4.6 Object Pool

Object Pool -luontimallin tarkoitus on säästää resursseja ja parantaa suorituskykyä kuten Prototype-mallilla. Object Pool -malli kuitenkin uudelleenkäyttää luotuja objekteja eikä luo niistä kloonveja kuten Prototype-malli. Kuvassa 9 näkyy Object Pool -suunnittelumallin toteutus. [22.]

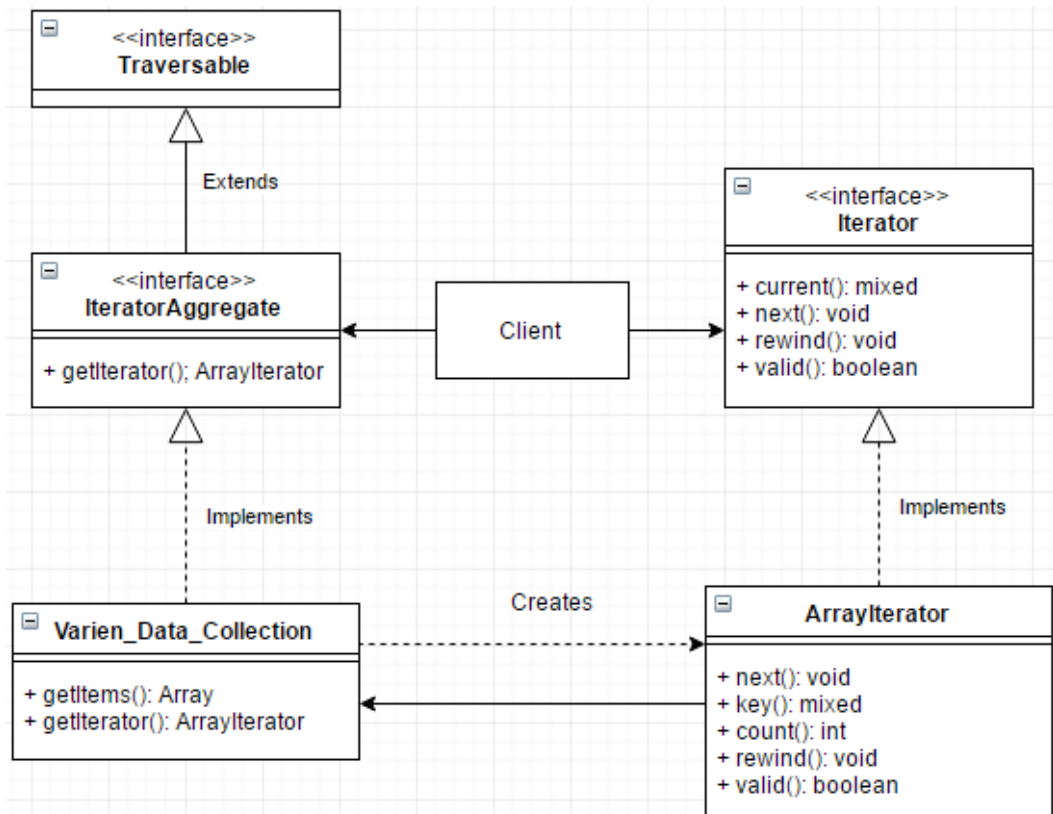


Kuva 9. Object Pool -suunnittelumallin toteutus. [22]

Kuvassa 9 Client-luokka pyytää luokalta käytettävää oliota. funktiot `getProductModel` ja `getCustomerModel` varmistavat, onko pyydetty olio käytettävissä. Mikäli olio on käytettävissä, se palautetaan Client-luokalle `Mage::objects()->load()`-funktiokutsun avulla. Jos oliota ei kuitenkaan löydetä, niin se luodaan ja tallennetaan `Mage::objects->save()`-funktiokutsun avulla. [23.]

## 4.7 Iterator

Iterator on käyttäytymismalli, jonka tarkoituksena on toteuttaa tapa iteroida olioita sisältävä kokoelma kuitenkin niin, ettei iterointia käyttävä ohjelmoijan tarvitse välittää toteutuksesta mitään. Iterator-suunnittelumallin avulla voidaan iteroida kokoelmia riippumatta kokoelman tyypistä. Kuvassa 10 nähdään Iterator-suunnittelumallin toteutus. [24.]



Kuva 10. Iterator-suunnittelumallin toteutus. [24.]

Magentoissa voidaan ladata kokoelma kutsumalla `Mage::getModel('catalog/product')->getCollection()`-funktiota. Funktio palauttaa kokoelman tuoteolioista, joka puolestaan perii kuvan 6 Varien\_Data\_Collection-luokkaa. Varien\_Data\_Collection-luokka toteuttaa PHP:n omaa IteratorAggregate-luokkaa. Tästä syystä kokoelmaa voi iteroida PHP:n `foreach`-silmukalla. [25.]

## 4.8 Lazy Loading

Lazy Loading -suunnittelumallin päätarkoituksena on säästää muistia ja ladata objektit vasta silloin, kun kyseistä objektia käytetään. Lazy Loading -malli ei kuulu varsinaisesti mihinkään suunnittelumalliryhmään. Magentossa tätä ei kuitenkaan toteuteta objekteilla vaan datalle. [26.]

Magentossa datan haku tapahtuu kokoelmien avulla esimerkiksi `$productCollection->addFieldToFilter('sku', 'n2610')`-funktiokutsu, joka palauttaa kokoelman tuotteista, joilla sku-arvo on n2610. Jos datan haku tapahtuisi olioiden luonnin yhteydessä, pitäisi ladata olio, jonka jälkeen lisätä siihen filtteri ja sen jälkeen taas ladata olio ja lisätä siihen filtteri, kunnes kaikki filttrit on tehty oliolle. Tämä vaikuttaa suorituskykyyn ja tästä syystä Magento lataa datan vasta kun kyseistä sitä käytetään. Toisin sanoen Magento tallentaa kaikki tarvittavat filttrit ensin ja sen jälkeen hakee tarvittavan arvon kyseisillä filtereillä. [26.]

## 4.9 Service Locator

Service Locator -suunnittelumallia enkapsuloi prosessin, jolla tietyn palvelun haku tapahtuu vahvalla abstraktiotasolla. Service Locator ei myöskään kuulu mihinkään suunnittelumalliryhmään. Tämä mahdollistaa sen, ettei ohjelmoijan tarvitse tietää tai välittää, mitä palvelua Service Locator -malli palauttaa. [27.]

Magento käyttää Service Locator -suunnittelumallia `getConnection()`-funktiossa. Funktio palauttaa tietokannan ORM:n tietämättä, mikä se todellisuudessa on. Lisäksi tämä mahdollistaa sen, että ohjelmoija voi muuttaa tietokannan palvelua yhdestä tietystä paikasta muuttamatta joka paikasta koodia, jossa kyseistä palvelua käytetään. Magentossa on esimerkiksi `$installer->getConnection()`-funktio, jossa yhteys tietokantaan luodaan. Se, mihin tietokantaan funktio yhdistää, määritellään `config.xml`-tiedostossa. Määritys näkyy kuvassa 11. [27.]

```

1 <resources>
2   <module_read>
3     <connection>
4       <host>localhost</host>
5       <username/>
6       <password/>
7       <dbname>magento</dbname>
8       <model>mysql4</model>
9       <type>pdo_mysql</type>
10      <active>1</active>
11    </connection>
12  </module_read>
13 </resources>

```

Kuva 11. Yhteyden määrittäminen local.xml-tiedostossa. [27.]

#### 4.10 Module

Module-suunnittelumallin tarkoituksena on yhdistää yhteenkuuluvat koodit ryhmiiksi, joita voi kuitenkin lisätä pääohjelmaan helposti. Tämä mahdollistaa sen, että pääohjelmaa varten voidaan rakentaa eri ominaisuuksia, jotka ovat kokonaan erillään pääohjelmasta. Ominaisuuden tai moduulin valmistuttua se voidaan suoraan lisätä pääohjelmaan. Module-suunnittelumalli ei kuulu mihinkään suunnittelumalliryhmään. Moduuleja voidaan ajatella lisäosina, jotka tuovat uusia ominaisuuksia pääohjelman lisäksi. [28.]

Magentossa moduulit sijaitsevat kansiossa `app/code/core`, `app/code/community` ja `app/code/local`. Kansioissa sijaitsevat erinimiset nimiavaruudet, joissa moduulit on kehitetty. Nimiavaruuksien tarkoitus on erotella samanlaiset moduulit toisistaan. Nimiavaruuksien on oltava nimeltään uniikkeja. Esimerkiksi, jos käytössä on kaksi samannimistä moduulia `HelloWorld`, niin Magento ei tiedä kumpaa `HelloWorld`-moduulia käyttää. Nimiavaruuksien avulla kuitenkin erotellaan nämä kaksi, vaikka `CompanyFirstHelloWorld/HelloWorld` ja `CompanySecond/HelloWorld`. Näin Magento osaa erottaa `CompanyFirstHelloWorld`- ja `CompanySecondHelloWorld` sijaitsevat `HelloWorld`-moduulit keskenään. [28.]

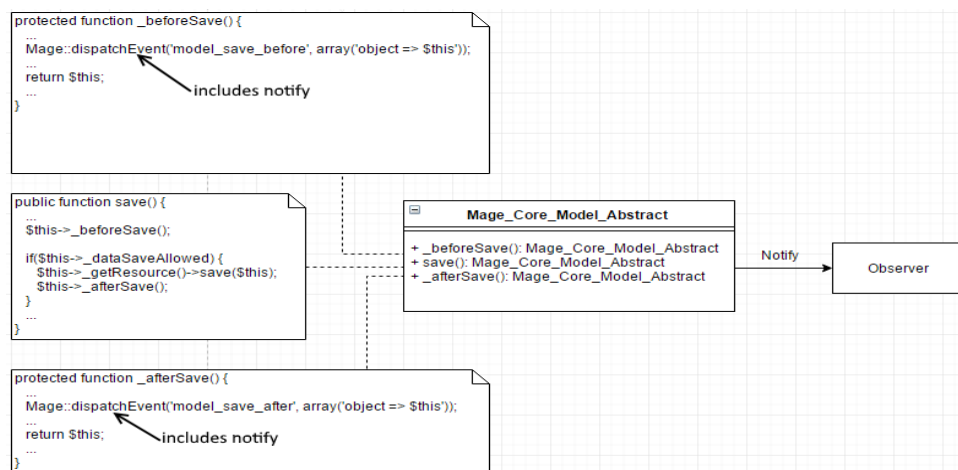
Core-kansio sisältää kaikki Magenton omat moduulit. Näitä tiedostoja ei saa muuttaa, sillä jos moduuleihin tulee päivitys, niin kaikki vanhat muutokset ylikirjoitetaan. Local-kansio sisältää puolestaan kaikki omat moduulit ja community-kansio sisältää kaikki ulkopuolisten kehittämät moduulit, jotka eivät kuitenkaan ole Magenton omia moduuleita.

Kaikki moduulit on myös määriteltävä app/etc/modules-kansiossa XML-tiedostona. Näin Magento havaitsee, että uusi moduuli on asennettu ja käyttää sitä verkkokaupassa. [28.]

#### 4.11 Observer

Observer-käyttämismalli tulee erittäin hyödylliseksi tilanteessa, jossa on yhden suhde moneen riippuvuus objektien välillä. Kun Observer-mallissa yhden objektin tila muuttuu, niin siitä ilmoitetaan muille riippuville objekteille, jotka puolestaan päivittävät omaa tilaansa. Observer-mallin avulla voidaan esimerkiksi helposti luoda erilaisia näkymiä MVC-mallissa kuitenkin välittämättä, miten data näkyy näkymässä ja kun yhden näkymän tilaa muutetaan, niin kaikki saavat siitä ilmoituksen ja päivittävät myös omaa tilaansa. [29.]

Observer-suunnittelumallissa on kaksi pääkokonaisuutta, jotka ovat Observer ja Subject. Observer kuuntelee Subject-kokonaisuutta ja tekee tiettyjä toimintoja, kun jonkinlainen ilmoitus on Observerille annettu. Subject puolestaan tuntee rekisteröidyt Observerit ja ilmoittaa niitä tekemään toimintonsa, kun jokin tietty tapahtuma on tapahtunut. Kuvassa 12 on Observer-mallin toteutus Magentossa UML-kaaviona. [29.]



Kuva 12. UML-kaavio Observer-suunnittelumallin eri osasista [29].

Magentossa Observer-mallia hyödynnetään aina kun halutaan tehdä joitain toimintoja esimerkiksi tuotteen tallennuksen jälkeen tai sen jälkeen, kun ladataan tietty sivu Magenton hallintapaneelistä. Kuten kuvan 12 save()-funktion toteutuksessa huomataan,

aina kun esimerkiksi joku tuote tallennetaan, niin tallennuksen yhteydessä ajetaan `_beforeSave()` funktio, joka puolestaan kutsuu `Mage::dispatchEvent()`-funktioita. `DispatchEvent` tunnistaa ja suorittaa määrättyt Observer-ilmentymät ja sille asetetut funktiot. Funktioiden parametrina lähetetään tallennettava objekti ja näin sitä voidaan muokata halulla tavalla. Observer-ilmentymien suoritusten jälkeen ajetaan vastaavasti `save()`- ja `_afterSave()`-funktioita. [30.]

Magentossa tapahtumat määritellään tapahtumat ja Observer-luokat `config.xml`-nimisessä tiedostossa, joka luodaan moduulin `etc`-kansioon. Kyseisessä tiedostossa määritellään myös muita tärkeitä konfiguraatietietoja, joita Magento käyttää. Kuvassa 13 näkyy `config.xml`-tiedoston konfiguraatio. [31.]

```

1 <config>
2   <global>
3     <events>
4       <my_event>
5         <observers>
6           <my_unique_identifier>
7             <type>model</type>
8             <class>module/observer</class>
9             <method>myEvent</method>
10          </my_unique_identifier>
11        </observers>
12      </my_event>
13    </events>
14  </global>
15 </config>

```

Kuva 13. Observer-luokkien ja tapahtumien määrittelyminen `config.xml`-tiedostossa.

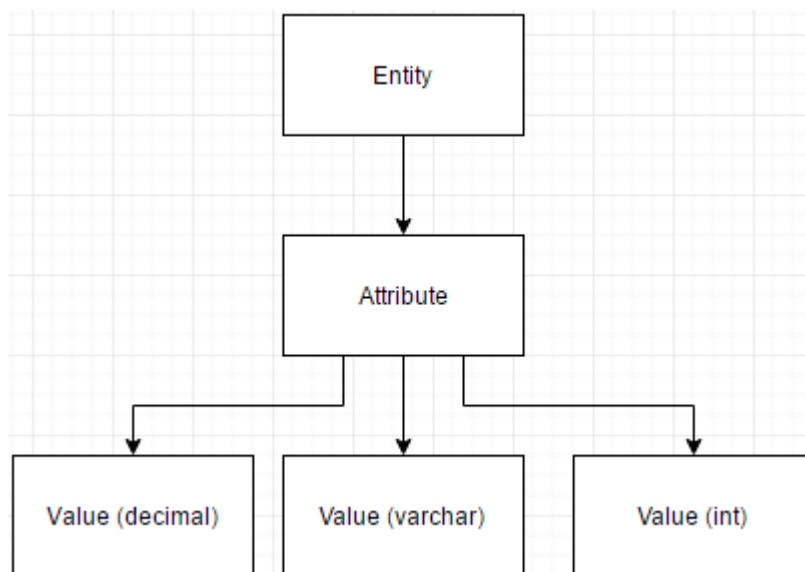
Kuvan 13 `<my_event>`-osa määrää sen, mikä tapahtuma suoritetaan. Eri tapahtumat ovat `load_before`, `load_after`, `save_before`, `save_after`, `delete_before` ja `delete_after`. Kuvan 13 `<my_unique_identifier>` nimetään uniikisti oman moduulin ja tapahtuman mukaan, `<type>` kyseisen Observerin tyyppiin, `<class>`, mikä luokka ajetaan, kun tapahtuma tapahtuu ja lopuksi `<method>` sen mikä luokan funktio ajetaan, kun tapahtuma tapahtuu. Eri tapahtumia voi tapahtua esimerkiksi tallentaessa eri tuotteita tai kategorioita verkkokauppaan. [32.]

#### 4.12 EAV-malli

EAV-mallia (Entity Attribute Value) ei luokitella mihinkään suunnittelumalliryhmään, sillä se on eräänlainen tietokantatoteutus. Magento tallentaa kahdenlaisia malleja tietokan-



taan, jotka ovat yksinkertaiset mallit ja EAV-mallit. Yksinkertaiset mallit tallennetaan tietokantaan niin, että sen kaikki tiedot ovat saatavilla yhdestä tietokantataulusta kuten perinteisissä tietokantatoteutuksissa. EAV-malli kuvaa kokonaisuuksia, joilla on dynaaminen määrä ominaisuuksia. EAV-mallia käytetään usein, kun tarvitaan dynaaminen määrä ominaisuuksia tietokannassa. Magentossa tämä tarkoittaa sitä, että tuotteilla voidaan lisätä attribuutteja eli ominaisuuksia kuten väri, materiaali tai mikä tahansa muu ominaisuus. EAV-mallin avulla saadaan kaikki ominaisuudet lisättyä muuttamatta itse tietokantatoteutusta. Näin saadaan myös tietokantatoteutuksesta helposti laajennettava. EAV-mallissa on kuitenkin huonotkin puolensa yksinkertaisiin tietokantatoteutuksiin verrattuna. EAV-mallin toteutuksessa mitään dataa ei ole helposti saatavilla, vaan tiedot on ladattava monesta eri taulusta. Tämä tarkoittaa sitä, että tietokantakutsuista tulee raskaita. Tästä syystä EAV-malli ei ole kuitenkaan ideaalinen toteutus tilanteissa, jossa suorituskyvyllä on suuri merkitys. Kuvassa 16 näkyy osa Magenton EAV-mallin toteutuksesta. [33, s. 65-66.]

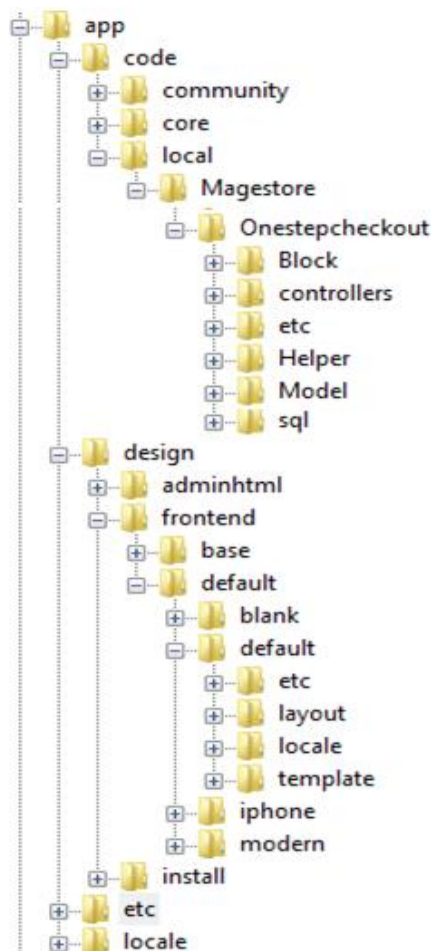


Kuva 14. Osa Magenton EAV-mallin toteutuksesta. [33, s. 65.]

Kuvan 14 EAV-mallin Entity-osa kuvaa tiettyä objektia tai kokonaisuutta. Magenton tapauksessa Entity-taulussa on rivi kaikista verkkokaupassa olevista tuotteista. Attribute-osassa on tallennettu kaikki verkkokaupassa olevat ominaisuudet kuten tuotteen väri ja koko. Lopuksi jokaisesta mahdollisesta tyyppistä on oma taulunsa, johon tallennetaan pelkästään attribuuttien arvot. Näin saadaan tietokannasta helposti laajennettava ja esimerkiksi Magento-verkkokaupan tuotteille ja kategorioille dynaaminen määrä ominaisuuksia. [33 s. 65-68.]

## 5 Magenton arkkitehtuuri

Vaikka Magenton kansiorakenteesta saa hyvän kuvan siitä, kuinka Magento hyödyntää MVC-mallia. Magenton kansiorakenne voidaan jakaa kahteen eri osaan. App/code-kansio sisältää kaikki tietokantaan ja sovelluslogiikkaan liittyvät koodit ja app/design-kansio sisältää kaikki hallintapanelin ja verkkokaupan visuaaliseen näkymään liittyvät tiedostot ja koodit. Tämä jako on tehty helpottaakseen eri tiedostojen järjestelyä. Kuvassa 15 nähdään Magenton kansiorakenne [34.].



Kuva 15. Magenton kansiorakenne. [35.]

Kuvan 15 tärkeitä kansioita ovat kaikki app/code/local-kansion sisältämät kansiot, app/design-kansion sisältämät layout ja template-kansiot ja lopuksi app/etc-kansio. App/etc-kansio sisältää asennettujen moduulien konfiguraatitiedostot, joiden avulla Ma-

gento tunnistaa ja päivittää asennetut moduulit. App/code/local-kansio sisältää Magestore -ja Onestepcheckout kansiot, jotka seuraavat Nimiavaruus/ModuulinNimi nimeämiskäytäntöä. [34.]

Onestepcheckout/etc-kansiossa ovat kaikki moduulin liittyvät konfiguraatiotiedostot kuten config.xml ja system.xml. Helper-kansion tiedostot sisältävät funktioita, jotka eivät kuulu tai sovi mihinkään muuhun luokkaan. Näiden on kuitenkin tarkoitus olla hyvin lyhyitä funktioita, jotka helpottavat esimerkiksi tietyn datan iteroimista tai valmistamista HTML-koodia varten. Lopuksi sql-kansion tiedostot sisältävät tietokantaan kohdistuvaa koodia. Näitä ovat esimerkiksi taulujen tai kategorioiden luonti tietokantaan. Layout-kansio sisältää konfiguraatiotiedostot, mistä kaikki template-tiedostot löytyvät ja rakentuvat ja template-kansio sisältää PHTML-tiedostot, joiden avulla näkymät renderöidään. [34.]

## 6 Open Graph Markup -tekniikka

Facebookin avulla sosiaalisen median käyttäjät voivat jakaa tietoa muilta verkkosivuilta Facebookiin. Tämän mahdollistaa Open Graph Markup -tekniikka, jonka avulla Facebook osaa jakaa jaetun sisällön eri osiin. Näin Facebook pystyy renderöimään nämä omalla sivullaan. Ilman Open Graph Markup -tekniikkaa sisällön jakaminen Facebookiin onnistuu mutta Facebook Crawler ei tiedä, miten tuotteen tiedot pitäisi näyttää ja yrittää parhaansa arvaamalla. Facebook Crawler on tekniikka, jonka avulla Open Graph Markup -data käsitellään. Open Graph Markup on suunniteltu kehittäjille ja yrityksille, jotka haluavat antaa verkkosivunsa käyttäjilleen mahdollisuuden jakaa sisältöä heidän verkkosivultaan Facebookiin. Open Graph Markup -tekniikkaa käyttävät muutkin sosiaaliset mediat kuten Google+ ja Twitter. [36.]

Open Graph Markup -tiedot laitetaan verkkosivun HTML-koodiin metadatanä, kuten kuvassa 16 näkyy. Metadatalle annetaan kaksi attribuuttia, jotka ovat property ja content. Property-attribuutin avulla kerrotaan Facebook Crawlerille, mitä tyyppiä jaettu sisältö on. Tärkeimmät Open Graph -tyypit, joita käytetään ovat URL, type, title, description ja image. Toinen attribuutti content on vain kyseisen Open Graph Markupin sisältö. [37]

```

<meta property="og:url" content="http://venturebeat.com/2015/05/01/article1923
<meta property="og:type" content="article"/>
<meta property="og:title" content="Apple rolls out CarPlay for safely using iPh
<meta property="og:description" content="Apple has launched its CarPlay featur
<meta property="og:img" content="http://venturebeat.com/images/carplay.jpg"/>

```

Kuva 16. Open Graph Markup datan asettaminen HTML-tiedostoissa [37].

Näiden tietojen avulla, kun verkkosivulta jaetaan sisältö Facebookiin, niin Facebook Crawler osaa käsitellä ja renderöidä sisällön onnistuneesti. Kuvassa 17 näkyy esimerkki oikein formatoidusta sisällöstä ja sen eri Open Graph Markup -osista [37.]

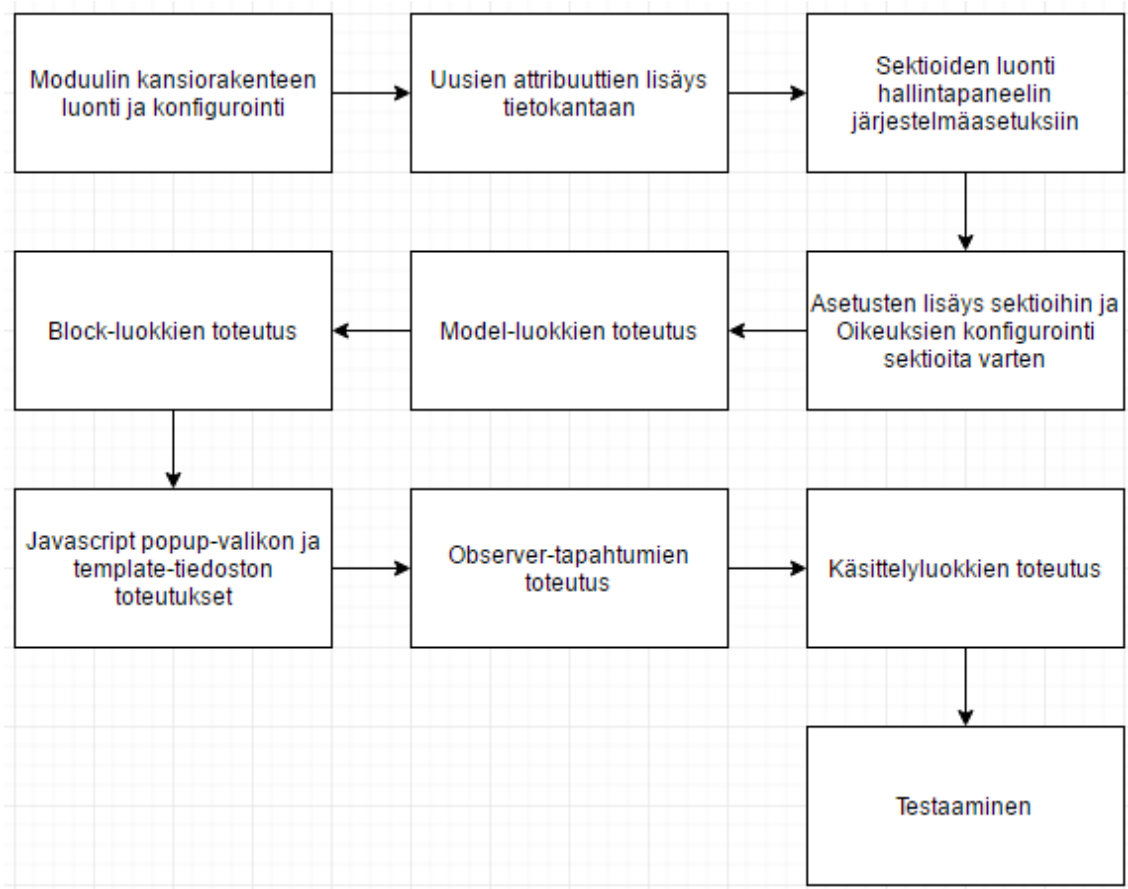


Kuva 17. Esimerkki Facebookiin jaetusta sisällöstä ja sen Open Graph Markup -osista [37].

## 7 Open Graph Markup -moduulin toteutus

Insinööriyössä tehdyn moduulin on tarkoitus lisätä Magentoon ominaisuus, jolla verkkokaupan omistajat voivat lisätä verkkokaupan tuotteille, kategorioille ja CMS-sivuille omat Open Graph Markup -tiedot suoraan hallintapaneelistä koskematta lähdekoodiin. Näin verkkokaupalle saadaan ominaisuus, jonka avulla verkkokaupan asiakkaat voivat jakaa

sisältöä Facebookiin, niin että se näkyy Facebookissa verkkokaupan ylläpitäjien haluamalla tavalla. Kuvassa 18 näkyy Open Graph Moduulin kehityksen eri vaiheet.



Kuva 18. Moduulin kehityksen eri vaiheet.

Ensin on konfiguroitava ja luotava kansiorakenne, joka vastaa tyypillistä Magento-moduulin kansiorakennetta, kuten kuvassa 15 nähtiin. Sen jälkeen lisätään moduulia varten tarvittavat ominaisuudet tietokantaan. Tietokantamuutosten jälkeen luodaan sektiot Magento-verkkokaupan hallintapaneeliin järjestelmäasetukset osioon, josta verkkokaupan omistaja voi muokata moduulin asetuksia. Kuvassa 19 nähdään, mistä hallintapaneelin järjestelmäasetukset löytyvät.

The screenshot shows the Magento Admin Panel dashboard. The top navigation bar includes 'Dashboard', 'Sales', 'Catalog', 'Customers', 'Promotions', 'Newsletter', 'CMS', 'Reports', and 'System'. A message banner at the top reads: 'Latest Message: Reminder: Change Magento's default phone numbers and callouts before site launch'. The main dashboard area contains several widgets: 'Lifetime Sales' (€0.00), 'Average Orders' (€0.00), 'Last 5 Orders' (No records found), 'Last 5 Search Terms' (example product, 0 results, 1 use), and 'Top 5 Search Terms' (example product, 0 results, 1 use). On the right, a sidebar menu lists various system management options, with 'Configuration' highlighted in a red box. Below the sidebar, there are additional widgets for 'Revenue' (€0.00), 'Bestsellers', 'Most Viewed F', and a 'Product Name' field.

Kuva 19. Magenton järjestelmäasetusten sijainti.

Kuvassa 19 on osa Magenton hallintapaneelista. Hallintapaneelissa verkkokaupan omistajat voivat esimerkiksi lisätä tuotteita, kategorioita tai muuta sisältöä verkkokauppaansa. Moduulin sektioiden luonti lisätään järjestelmäasetusten sivupalkkiin, josta verkkokaupan omistajat pääsevät muokkaamaan moduulin asetuksia. Kuvassa 20 nähdään järjestelmäasetusten etusivu.

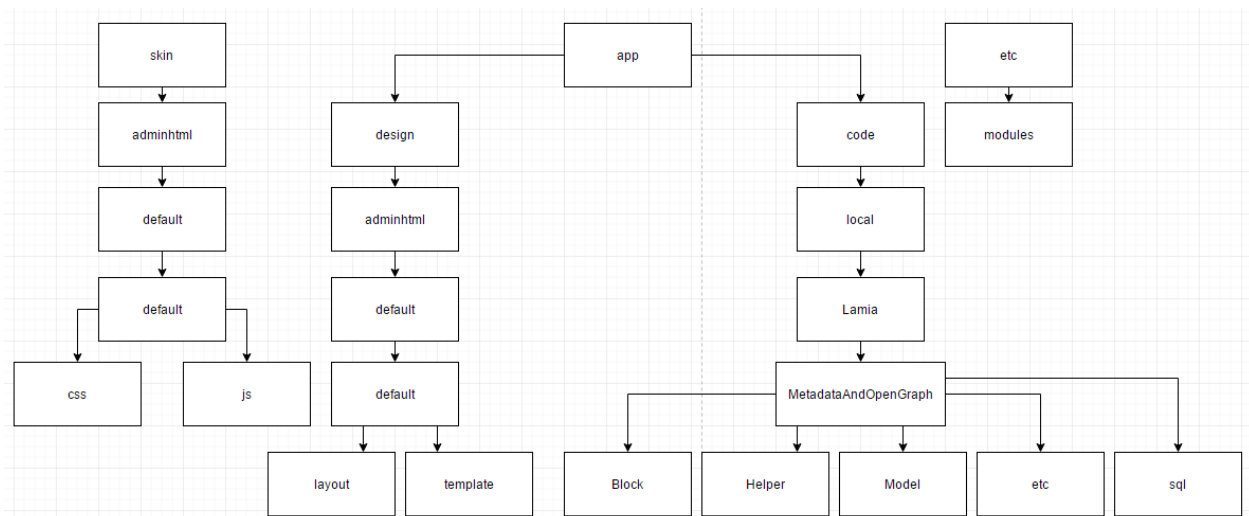
The screenshot shows the Magento Admin Panel interface. At the top, there is a navigation bar with tabs for Dashboard, Sales, Catalog, Customers, Promotions, Newsletter, CMS, Reports, and System. Below this is a 'Latest Message' banner. The left sidebar contains a 'Configuration' menu with a red box highlighting the 'GENERAL' section. The main content area shows the 'General' configuration page with 'Countries Options' settings. The 'Default Country' is set to 'United States'. The 'Allow Countries' list includes Afghanistan, Åland Islands, Albania, Algeria, American Samoa, Andorra, Angola, Anguilla, Antarctica, and Antigua and Barbuda. The 'Postal Code is Optional for the following countries' list includes Guam, Guatemala, Guernsey, Guinea, Guinea-Bissau, Guyana, Haiti, Heard & McDonald Islands, Honduras, and Hong Kong SAR China. The 'European Union Countries' list includes Andorra, Angola, Anguilla, Antarctica, Antigua and Barbuda, Argentina, and Armenia.

Kuva 20. Magenton hallintapaneelin järjestelmäasetusten etusivu ja punaisella merkitty sivupalkki, johon moduulin asetukset lisätään.

Sektiot lisätään kuvan 20 punaisella merkittyyn sivupalkkiin, jonka jälkeen luodaan moduulin malliluokat, jotka käsittelevät ja muokkaavat dataa. Malliluokkien jälkeen luodaan Block-luokat, jotka valmistavat malliluokista haetun datan template-tiedostoille renderöitäväksi. Template-tiedostojen yhteydessä luodaan myös JavaScript-logiikka ponnahdusikkunan toteutukselle, jonka avulla verkkokaupan omistajat voivat asettaa haluttuja verkkokaupan muuttujia Open Graph Markup -moduulin oletusasetuksiin kuten tuotteen tai kategorian nimi. Asetusten jälkeen lisätään vielä Observer-tapahtumat, joiden avulla verkkokaupan omistajat voivat asettaa Open Graph Markup -asetuksia tuotteille, kategorioille ja CMS-sivuille erikseen. Kuvan 18 viimeisenä vaiheena luodaan testit moduulia varten.

## 7.1 Moduulin kansiorakenteen luonti ja konfigurointi

Aluksi luodaan moduulin kansiorakenne ja konfiguroidaan moduuli, jotta Magento tunnistaa moduulin onnistuneesti. Kansiorakenne luodaan usein samanlaisiksi kuin Magenton omat moduulit, sillä se on todettu hyväksi käytännöksi moduuleja kehittäessä. Samalla muut kehittäjät ymmärtävät moduulin rakenteen paremmin ja löytävät esimerkiksi muokattavat tiedostot helposti. Kuvassa 21 nähdään moduulin rakenne ja sen tärkeimmät kansiot.



Kuva 21. Moduulin rakenne ja sen tärkeimmät kansiot.

Magento ei kuitenkaan tunnista moduulia pelkästään sen kansiorakenteen avulla. Moduulille on lisättävä oma konfiguraatiotiedosto MetadataAndOpenGraph/etc-kansion config.xml-tiedostoon, jossa määritellään moduulin versio ja sen tyyppi, kuten kuvassa 22 näkyy. Moduulin config.xml-tiedoston toteutus näkyy kokonaisuudessaan liitteessä 1.

```

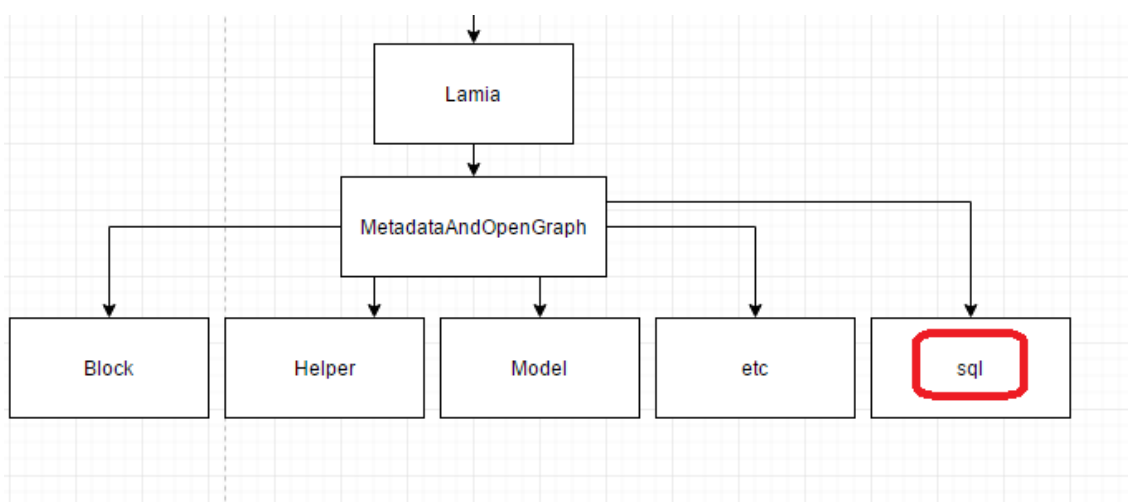
<?xml version="1.0"?>
<config>
  <modules>
    <Lamia_MetadataAndOpenGraph>
      <active>true</active>
      <codePool>local</codePool>
    </Lamia_MetadataAndOpenGraph>
  </modules>
</config>
  
```

Kuva 22. Oman Magento-moduulin määrittely config.xml-tiedostossa.



## 7.2 Uusien attribuuttien lisäys

Magenton tietokannassa ei oletuksena löydy saraketta, johon tallentaa verkkokaupan omistajan asettamat Open Graph Markup -tiedot. Tästä syystä on luotava asennusskriptit kuvassa 23 näkyvään sql-kansioon. Magentossa attribuutteja kutsutaan yleensä ominaisuuksia, jotka lisätään kategorioille, tuotteille tai CMS-sivuille asennusskriptien kautta. Asennusskripteillä voi myös lisätä tuotteita ja kategorioita kätevästi verkkokauppaan.



Kuva 23. Asennusskriptit luodaan moduulin sql-kansioon.

Asennusskriptit suoritetaan, kun verkkokaupan verkkosivulla tai hallintapaneelisivulla käydään. Asennusskriptit kuitenkin ajetaan vain kerran, jonka jälkeen tietokannassa päivitetään moduulin versio. Magento ajaa asennusskriptit versionumerojärjestyksessä. Tästä syystä asennusskriptin oikea nimeäminen on tärkeää. Asennusskriptin versionumero pitää olla sama kuin moduulille määritetty versionumero. Jos moduulin versio on 0.2.0, niin Magento ajaa kaikki kyseistä versiota alemmat skriptit ja lopulta 0.2.0-versiolle määritetyn skriptin. Ensimmäinen skripti nimetään usein install-0.1.0.php-nimellä.

Open Graph Markup moduulissa tarvittiin yksi asennusskripti, jonka avulla lisättiin kategorioille ja tuotteille attribuutit Open Graph Markup -dataa varten. Kuvassa 24 nähdään esimerkki, miten attribuutit määritetään skriptissä.

```

$installer = $this;
$installer->startSetup();

$categoryEntity = $installer->getEntityTypeId('catalog_category');
$productEntity = $installer->getEntityTypeId('catalog_product');

//Category
$categoryAttributesData = array(
    'og_url' => array(
        'backend_type' => 'varchar',
        'input' => 'text',
        'label' => 'Open Graph Markup Url',
        'required' => 0,
        'unique' => 0,
        'global' => Mage_Catalog_Model_Resource_Eav_Attribute::SCOPE_GLOBAL,
        'user_defined' => 0,
        'visible' => false,
        'group' => ''
    ),
    'og_description' => array(

```

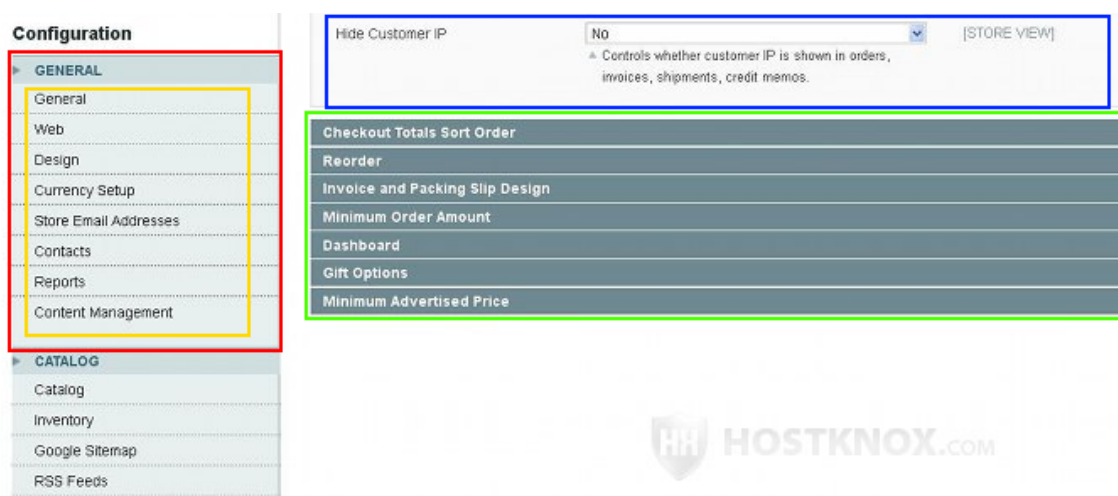
Kuva 24. Attribuuttien määrittely asennusskriptissä.

Kuvassa 24 aluksi alustetaan installer-objekti, jonka avulla saadaan kategorian sekä tuotteen tyyppi-Id:t. Näiden avulla voidaan myöhemmin määrittellä, mihin tauluihin attribuutit lisätään. \$categoryAttributesData-muuttuja sisältää kaikki attribuutit, jotka halutaan kategorioille lisätä. Lisätyt attribuutit olivat URL-osoite, kuvaus, kuva ja otsikko. Tietokannassa nämä nimettiin og\_url, og\_description, og\_img, og\_title. Kuvan 24 attribuuttien sisällä määritellään niihin liittyvät tiedot esimerkiksi, mitä tyyppiä kyseinen attribuutti on ja onko käyttäjillä oikeus muokata sitä attribuuttia. Kyseiset attribuutit lisättiin myös tuotteille ja CMS-sivuille.

Verkkokaupan omistajalla piti olla myös mahdollisuus antaa muuttujia Open Graph Markup -kenttiin, jotka käsitellään tekstiksi ennen asetusten tallennusta. Näin omistajan ei tarvitse käsin muokata Open Graph Markup -asetuksia aina, kun esimerkiksi tuotteen kuvausta tai kuvaa muutetaan. Esimerkiksi {{category.name}} käsitellään muuttujaksi. Sen pitäisi generoitua muokattavan kategorian nimeksi ja {{product.description}} pitäisi käsitellä tuotteen kuvaukseksi. Tämän toteuttamiseksi og\_url, og\_description, og\_img ja og\_title attribuuttien lisäksi asennusskriptissä luotiin samat muuttujat editable-etuliitteellä. Näiden attribuuttien oli tarkoitus sisältää käsittelemätön data, joka käsitellään ja lisätään etuliitteettömiin muuttujiin.

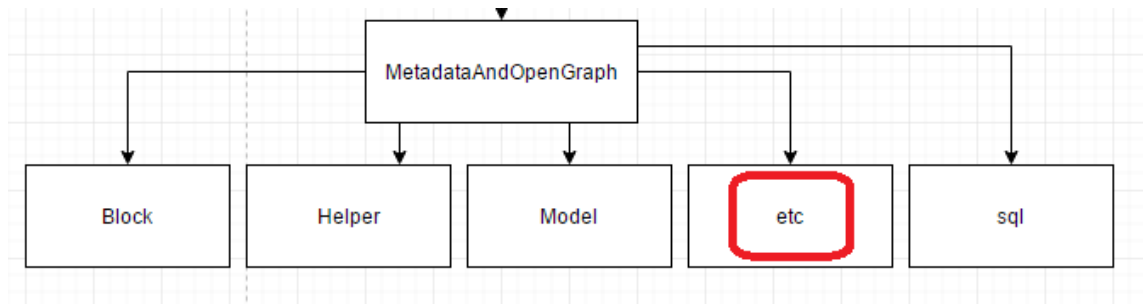
### 7.3 Sektioiden luonti hallintapaneelin järjestelmäasetuksiin

Seuraavaksi haluttiin lisätä Sektiot-hallintapaneeliin järjestelmäasetuksiin Open Graph Markup -tekstikentät, joista ylläpitäjät voivat asettaa Open Graph Markup -oletusasetukset kategorioille, tuotteille ja CMS-sivuille. Näin kaikki uudet tuotteet ja kategoriat saavat oletuksena kyseiset datat, mikäli niille ei ole ennaltaan asetettu niitä. Järjestelmäasetukset on ryhmitelty eri osiin kuten yleisasetukset ja asiakasasetukset, jotka sisältävät niihin kuuluvia alaryhmiä kuten teema-asetukset tai asiakkaan toivelista-asetukset. Näille alaryhmille voidaan vielä lisätä omia asetusryhmiä, joihin eri asetukset voidaan ryhmitellä. Kuvassa 25 nähdään, miten sektiot ja sen eri asetukset jakautuvat järjestelmäasetuksissa. Kuvassa 25 punainen alue vastaa sektiota, keltainen alue vastaa seksion alaryhmiä, vihreä alue vastaa alasektioissa sisältäviä asetusryhmiä ja lopuksi sininen alue vastaa varsinaisia asetuksia, jotka sijaitsevat seksion alaryhmissä. Tämä jako on toteutettu sen takia, että esimerkiksi kehittäjillä olisi mahdollisuus lisätä moduuliin liittyviä asetuksia ja ryhmittää asetukset haluamallaan tavalla. Moduuliin on luotava konfiguraatitiedosto, jonka avulla Magento lisää Open Graph Markup -asetuksia varten oman ryhmän tai seksion järjestelmäasetuksiin.



Kuva 25. Järjestelmäasetusten sektioiden jakautuminen eri osiin.

Uuden seksion lisääminen hallitapaneelin asetuksiin onnistuu system.xml-tiedoston avulla, luomalla se moduulin MetadataAndOpenGraph/etc-kansioon (kuva 26).



Kuva 26. System.xml-tiedosto sijaitsee moduulin etc-kansiossa.

Seksiota luodessa on tiedostossa konfiguroitava sektion nimi ja sen alasektiot. Tässä tapauksessa haluttiin Sektion nimeksi Configure Metadata And Open Graph Markup ja sille alasektiot tuotteita, kategorioita ja CMS-sivuja varten. Kuvassa 27 näkyy sektion ja sen alasektioiden määrittäminen system.xml-tiedostossa.

```

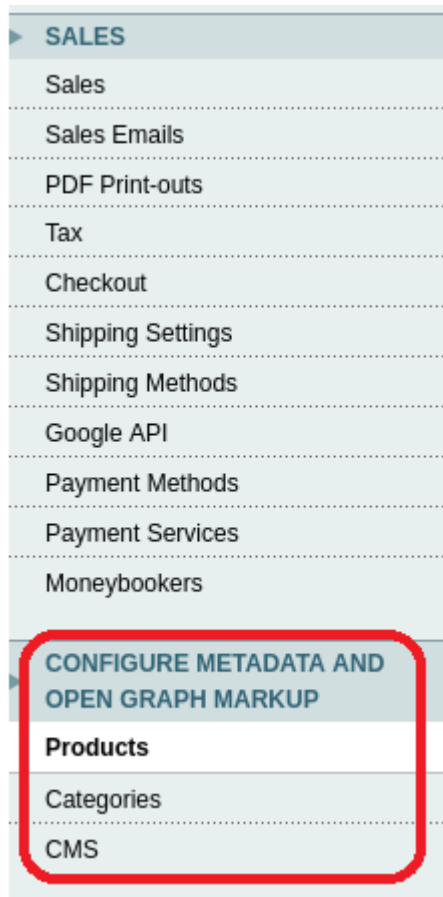
<?xml version="1.0"?>
<config>
  <tabs>
    <metadata_and_open_graph_config>
      <label>Configure Metadata And Open Graph Markup</label>
      <sort_order>99999</sort_order>
    </metadata_and_open_graph_config>
  </tabs>
  <sections>
    <metadata_and_open_graph_products>
      <label>Products</label>
      <tab>metadata_and_open_graph_config</tab>
      <frontend_type>text</frontend_type>
      <sort_order>1</sort_order>
      <show_in_default>1</show_in_default>
      <show_in_website>1</show_in_website>
      <show_in_store>1</show_in_store>
      <groups>

```

Kuva 27. Sektion ja sen alasektioiden määrittäminen system.xml tiedostossa.

Kuvassa 27 nähdään sektion luominen tabs-elementin sisällä. Sort\_order määrittää kyseisen sektion sijainnin hallintapaneelin konfiguraatioasetuksissa. Kuvassa näkyvät myös tärkeimmät osat alasektion määrittämistä varten. Tab-elementillä määritellään, mihin sektioon kyseinen alasektio kuuluu ja frontend\_typer avulla määritellään, mitä tyyppiä kyseinen sektio on. Tässä tapauksessa frontend\_type-elementin määrittämistä ei tarvita mutta sen määrittäminen on osa Magento-käytäntöä. Lopuksi group-elementillä pys-

tytään määrittämään, mitä kaikkia asetuksia pystytään kyseisessä sektiossa asettaa. Kuvan 27 alasektiot luotiin erikseen myös kategorioille ja CMS-sivuille. Kuvassa 28 nähdään lopputulos, miltä kyseiset sektiot näyttäivät hallintapaneelin Järjestelmä->Konfiguraatio-osiossa.



Kuva 28. System.xml:n tuottamat sektiot hallintapaneelin konfiguraatioasetuksissa.

#### 7.4 Asetusten lisäys sektioihin ja oikeuksien konfigurointi sektioita varten

Jotta saadaan alasektioihin eri Open Graph Markup -asetukset näkyviin, ne on määriteltävä Kuvan 29 groups-elementin sisällä. Alasektioille haluttiin kaksi eri asetusryhmää, jotta asetukset olisivat paremmin organisoituja. Ensimmäisessä ryhmässä ylläpitäjät pystyisivät asettamaan tavalliset metadata-tiedot ja toisessa Open Graph Markup -tiedot. Kuvassa 29 nähdään metadata-konfiguraatioryhmän määrittäminen groups-elementin sisällä.

```

<groups>
  <metadata>
    <label>Metadata</label>
    <sort_order>1</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>1</show_in_store>
    <expanded>1</expanded>
    <fields>
      <metadata_title>
        <label>Metadata Title</label>
        <frontend_type>text</frontend_type>
        <frontend_model>
          lamia_metadata_and_open_graph/adminhtml_system_config_product_inputButton
        </frontend_model>
        <sort_order>1</sort_order>
        <show_in_default>1</show_in_default>
        <show_in_website>1</show_in_website>
        <show_in_store>1</show_in_store>
      </metadata_title>
      <metadata_description>
        <label>Metadata Description</label>
        <frontend_type>textarea</frontend_type>
        <frontend_class>textarea</frontend_class>
        <frontend_model>
          lamia_metadata_and_open_graph/adminhtml_system_config_product_textareaButton
        </frontend_model>
        <sort_order>1</sort_order>
        <show_in_default>1</show_in_default>
        <show_in_website>1</show_in_website>
        <show_in_store>1</show_in_store>
      </metadata_description>
    </fields>
  </metadata>
</groups>

```

Kuva 29. Group elementin sisällä määritellyt konfiguraatioryhmät.

Kuvassa 29 näkyy metadata asetusr ryhmän määrittäminen. Metadataelementin alussa määritellään kyseisen asetusr ryhmän nimi, jonka jälkeen sen sisältö fields-elementin sisällä. Kuvassa 29 nähdään tekstikenttien luonti metadataotsikolle ja metadata kuvaukselle. Tiedostossa määritettiin myös samat konfiguraatiot Open Graph Markup -asetusr ryhmää varten.

Kenttien luonnissa tärkeimmät asetukset ovat frontend\_type- ja frontend\_model- elementit. Frontend\_type-elementissä määritellään, mitä tyyppiä kyseinen kenttä on ja frontend\_model elementillä määritellään siihen kuuluva malli, jota se käyttää renderöidessä kenttää kyseiselle sivulle. Frontend\_model-elementissä lamia\_metadata\_and\_open\_graph-osa viittaa moduulin nimiavaruuteen ja moduulin nimeen ja adminhtml\_system\_config\_product\_inputButton viittaa polkuun, josta Magento löytää kyseisen mallin tiedoston. Tässä tapauksessa malli löytyy hakemistosta Block/Adminhtml/System/Config/Product-kansiossa InputButton.php-tiedostona.

Oletuksena konfiguraatioryhmät tuotteille, kategorioille ja CMS-sivuille eivät näy, koska niille ei ole annettu tarpeellisia oikeuksia. Sen sijaan sivulla näkyy HTML 404 -virheilmoitus. Konfiguraatioryhmille pystytään antamaan oikeudet adminhtml.xml-tiedoston avulla,

joka sijaitsee samassa hakemistossa kuin system.xml ja config.xml. Kuvassa 30 näkyy adminhtml.xml-tiedoston toteutus.

```

<?xml version="1.0"?>
<config>
  <acl>
    <resources>
      <admin>
        <children>
          <system>
            <children>
              <config>
                <children>
                  <metadata_and_open_graph_products>
                    <title>Metadata And Open Graph Products</title>
                  </metadata_and_open_graph_products>
                  <metadata_and_open_graph_categories>
                    <title>Metadata And Open Graph Categories</title>
                  </metadata_and_open_graph_categories>
                  <metadata_and_open_graph_cms>
                    <title>Metadata And Open Graph CMS</title>
                  </metadata_and_open_graph_cms>
                </children>
              </config>
            </children>
          </system>
        </children>
      </admin>
    </resources>
  </acl>
</config>

```

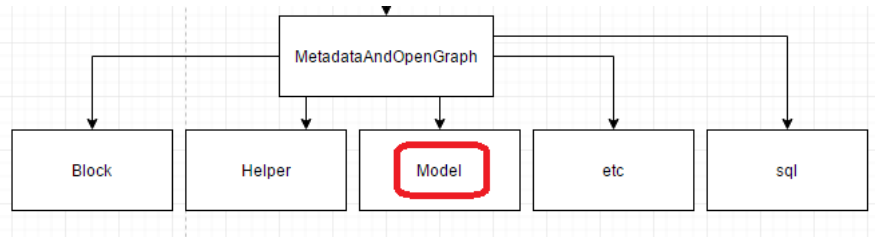
---

Kuva 30. Adminhtml.xml-tiedoston sisältö.

Kuvassa 30 huomataan, että konfiguraatioryhmille annetaan oikeudet admin/system/config-osioon, joka viittaa hallintapaneelin järjestelmäasetuksia. Config-elementissä myös määritellään halutut sektiot, joille oikeudet annetaan. Ilman tätä tiedostoa ja määrittelyä kyseisiä konfiguraatioryhmiä ei pystytä näyttämään hallintapaneelin konfiguraatioasetuksissa.

## 7.5 Model-luokkien toteutus

Seuraavaksi täytyi ohjelmoida malliluokat tuotteita, kategorioita ja CMS-sivuja varten. Kyseisten luokkien pääideana on ladata tarvittavat attribuutit esimerkiksi tuotteiden attribuutit, ladata ja käsitellä kyseiset attribuutit ja palauttaa nämä prosessoidut attribuutit Block-luokkia varten. Model-luokat luodaan moduulin MetadataAndOpenGraph-kansioon (kuva 31).



Kuva 31. Model-luokat luodaan Model-kansioon.

Model-luokkien avulla saadaan näytettyä hallintapaneelin ylläpitäjille kategoria, tuote ja CMS-sivukohtaiset muuttujat, joita verkkokaupan omistaja pystyy asettamaan Open Graph Markup -asetuksiin. Tuotteiden, kategorioiden ja CMS-sivujen lisäksi täytyi luoda erikseen Store-niminen luokka, joka hakee verkkokauppakohtaiset attribuutit kuten verkkokaupan nimen. Kuvassa 32 nähdään luokka tuotetta varten luodulle mallille.

```

class Lamia_MetadataAndOpenGraph_Model_Type_Product
  extends Lamia_MetadataAndOpenGraph_Model_Type_Abstract
{
  /**
   * @return array
   */
  public function getProcessedFields()
  {
    $preparedProductAttributes = $this->getAvailableFields();
    $preparedStoreAttributes = Mage::getModel('lamia_metadata_and_open_graph/type_store')->getProcessedFields();

    return $this->mergeArrays($preparedProductAttributes, $preparedStoreAttributes);
  }

  /**
   * @return Mage_Catalog_Model_Resource_Product_Attribute_Collection
   */
  public function loadFields()
  {
    return Mage::getResourceModel('catalog/product_attribute_collection');
  }

  /**
   * @return array
   */
  public function getAvailableFields()
  {
    $productCollection = $this->loadFields();
    $processedProductCollection = $this->preProcessCollection($productCollection);

    return $this->addPrefixToFields($processedProductCollection, 'product');
  }
}

```

Kuva 32. Tuotteen attribuuttien käsittelyä varten luotu malli.

Kuvassa 32 luokan nimi on perinteistä nimeämiskäytäntöön verrattuna erilainen. Tämä on Magenton nimeämistapa. Tällä nimeämistavalla Magento osaa etsiä ja tunnistaa luo-



kat oikeista paikoista. Kuvassa 32 luokan nimi on Product, ja se sijaitsee Lamia/MetadataAndOpenGraph/Model/Type-hakemistossa. Luokan tarkoituksena on hakea tarvittavat attribuutit, joita tuotteisiin kuuluu ja luoda lista näistä attribuuteista omalla etuliitteellä. Esimerkiksi käsitellyt tuoteattribuutit ovat product.name ja kategorioilla vastavasti category.name.

Kuvan 32 luokka perii abstraktin luokan. Abstraktit luokat on tehty sitä varten, että kyseisen moduulin laajennus olisi mahdollisimman helppoa. Jos joku luokka perii kyseisen abstraktin luokan, niin ohjelmoija tietää heti, mitä funktioita hänen täytyy luokassaan luoda ja mitä valmiita funktioita hänellä on käytössään. Magentossa Abstraktien luokkien ja rajapintojen ohjelmointi on erittäin tärkeää, sillä se helpottaa moduulien laajennusta huomattavasti.

Kuvassa 32 getProcessedFields-funktio hakee tietokannasta tuotteisiin liittyvät attribuutit, prosessoi ja lisää etuliitteet attribuuttiin ja lopuksi hakee verkkokauppaan liittyvät attribuutit sekä yhdistää ja palauttaa nämä listat. Category-, CMS- ja Store-mallit perivät saman abstraktin luokan ja toimivat samalla periaatteella. Kuvassa 33 näkyy osa abstraktia luokkaa ja siinä toteutettuja funktioita.

```

abstract class Lamia_MetadataAndOpenGraph_Model_Type_Abstract
    extends Mage_Core_Model_Abstract
    implements Lamia_MetadataAndOpenGraph_Model_Type_Source
{
    /**
     * @param Varien_Data_Collection $collection
     * @return array
     */
    public function preProcessCollection(Varien_Data_Collection $collection)
    {
        $processedArray = array();

        foreach ($collection as $obj){
            if($obj->getData('attribute_code')){
                array_push($processedArray, $obj->getData('attribute_code'));
            }
        }

        return $processedArray;
    }

    /**
     * @param array $queryResult
     * @return array
     */
    public function preProcessQueryFields(array $queryResult)

```

Kuva 33. Osa abstraktin luokan toteutusta, jonka tuotemalli perii.

Kuvassa 33 huomataan myös se, että abstrakti luokka toteuttaa rajapintaa. Kuvassa näkyvä funktio on hyvin yksinkertainen, joka lisää kaikki parametrina annetun kokoelman arvot, jotka on nimetty `attribute_code` nimellä taulukkoon. Kyseisen taulukon avulla saadaan HTML-koodi generoitua helpommin. Kuvassa 34 näkyy rajapintaluokan toteutus.

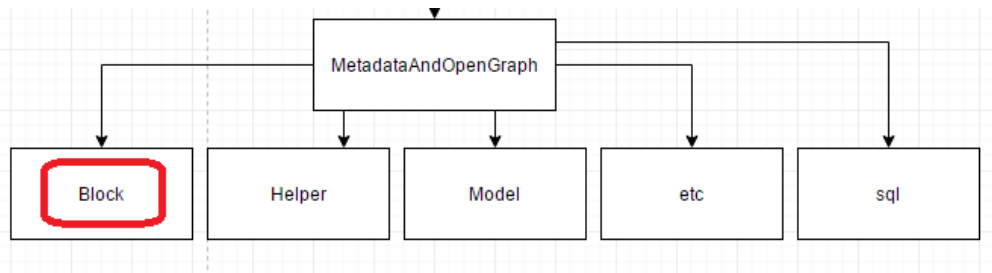
```
interface Lamia_MetadataAndOpenGraph_Model_Type_Source
{
    public function preprocessCollection(Varien_Data_Collection $collection);
    public function preprocessQueryFields(array $queryResult);
    public function getAvailableFields();
    public function getProcessedFields();
    public function loadFields();
}
```

Kuva 34. Rajapintaluokka, jota abstraktiluokka toteuttaa.

Rajapintaluokat ovat toteutuksiltaan hyvin yksinkertaisia ja niiden tarkoituksena on helpottaa moduulien laajennusta kuten abstraktien luokkienkin. Rajapintaluokat eivät kuitenkaan salli toteutuksia funktioille vaan pelkästään niiden esitellyt. Itse toteutukset täytyy kirjoittaa rajapintaa toteuttavissa luokissa, kuten tässä tapauksessa abstraktissa luokassa ja sen aliluokissa. Rajapinnat myös helpottavat ohjelmoijan työtä eikä ohjelmoijan tarvitse muistaa, mitä kaikkia funktioita luokissa pitää kirjoittaa vaan ohjelmointiympäristö yleensä varoittaa näistä puutteista.

## 7.6 Block-luokkien toteutus

Seuraavaksi kehitettiin Block-luokat tuotteita, kategorioita ja CMS-sivuja varten. Näiden avulla pystyttiin vaikuttamaan siihen miten erilaiset tekstikentät ja painikkeet näkyvät hallintapaneelissa muokattaessa metadata ja Open Graph Markup -tietoja. Luokat sijaitsevat moduulin Block-kansiossa kuten (kuva 35).



Kuva 35. Block-luokkiin liittyvät tiedostot luodaan moduulin Block-kansioon.

Tässä tapauksessa tarvittiin kaksi eri luokkaa. InputButton-luokalla generoitiin tavallinen tekstikenttä ja TextAreaButton-luokalla tekstialuekenttä, esimerkiksi tuotteen kuvausta varten. Molemmat luokat ovat sisällöltään hyvin samanlaisia, ainoana erona on tekstikenttien tyyppi. Kyseiset luokat tehtiin myös kategorioita ja CMS-sivuja varten. Kuvassa 36 nähdään InputButton-luokka, joka on luotu tuotteita varten.

```

class Lamia_MetadataAndOpenGraph_Block_Adminhtml_System_Config_Category_InputButton
  extends Mage_Adminhtml_Block_System_Config_Form_Field
{
  /**
   * @param Varien_Data_Form_Element_Abstract $element
   *
   * @return string
   */
  protected function _getElementHtml(Varien_Data_Form_Element_Abstract $element)
  {
    //Model
    $categoryModel = Mage::getModel('lamia_metadata_and_open_graph/type_category');
    $attributesData = $categoryModel->getProcessedFields();

    //Creating textField
    $titleTextField = '<input id="'. $element->getHtmlId().'" name="'. $element->getName()
      .'" value="'. $element->getEscapedValue().'" '. $element->serialize($element->getHtmlAttributes()).' />'. "\n";

    //Attribute button
    $titleButton = $this->getLayout()
      ->createBlock('adminhtml/widget_button')
      ->setData(array(
        'id' => $element->getHtmlId().'_button',
        'label' => $this->helper('adminhtml')->__('Attributes'),
        'onclick' => 'attributesPopup(id)'
      ));

    $popup = $this->getLayout()->createBlock('adminhtml/template')
      ->setData('attributes', $attributesData)
      ->setTemplate('lamiametadata/attributes_popup.phtml');

    return $titleButton->toHtml().$popup->toHtml().$titleTextField;
  }
}

```

Kuva 36. InputButton-luokka tuotteiden Open Graph Markup -asetuksien muuttamista varten.

Block-luokan tarkoituksena on siis generoida html-koodi, joka näytetään hallintapaneelin konfiguraatioasetuksissa Product-konfiguraatioryhmän alapuolella. Luokka perii Magenton oman FormField-luokan, ja sen avulla päästään käsiksi `_getElementHtml`-metodiin ja ylikirjotetaan se. Funktio on hyvin yksinkertainen. Aluksi funktiossa haetaan luodusta malliluokasta halutut attribuutit, luodaan tekstikenttä sekä painike ja lopuksi annetaan painikkeelle oma onclick-tapahtuma ja palautetaan generoitu HTML-koodi. Samalla periaatteella luotiin myös funktiot kategoriasivuja ja CMS-sivuja varten. Kuvassa 37 on renderöity näkymä.

**Current Configuration Scope:**  
 Default Config  
[Manage Stores](#)

**Configuration**

- GENERAL**
  - General
  - Web
  - Design
  - Currency Setup
  - Store Email Addresses
  - Contacts
  - Reports
  - Content Management
- CATALOG**
  - Catalog
  - Configurable Swatches
  - Inventory
  - Google Sitemap
  - RSS Feeds
  - Email to a Friend
- CUSTOMERS**
  - Newsletter
  - Customer Configuration
  - Wishlist
  - Promotions
  - Persistent Shopping Cart
- SALES**
  - Sales
  - Sales Emails
  - PDF Print-outs
  - Tax
  - Checkout

**Products**

**Metadata**

Metadata Url	Attributes	[STORE VIEW]
Metadata Description	Attributes	[STORE VIEW]
Metadata Title	Attributes	[STORE VIEW]

**Open Graph Markup**

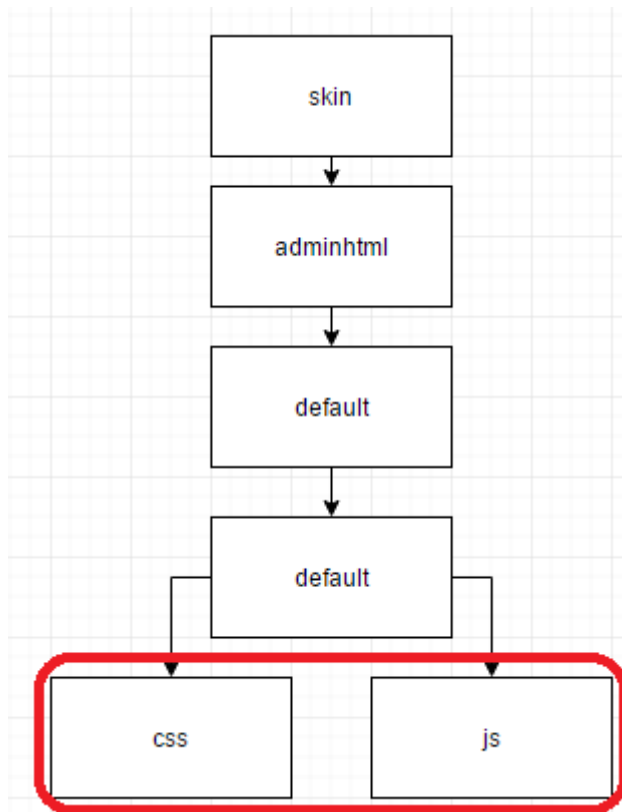
Open Graph Markup Url	Attributes	[STORE VIEW]
Open Graph Markup Description	Attributes	[STORE VIEW]
Open Graph Markup Title	Attributes	[STORE VIEW]

Kuva 37. Product-konfiguraatioryhmään generoituneet konfiguraatioasetukset Block-luokan avulla.

Attributes-painikkeesta tulee ponnahtusikkuna, jonka avulla ylläpitäjä voi lisätä kenttiin muuttujia, jotka haettiin ja prosessoitiin malliluokissa. Kyseisen ominaisuuden toteuttamiseksi tarvittiin Javascriptiä.

### 7.7 Javascript-ponnahtusikkunan ja template-tiedoston toteutus

Javascript-koodilla luotiin ponnahtusikkuna, painikkeen klikkaustapahtuma ja Open Graph Markup -muuttujien lisäykset tekstikenttiin. Javascript-tiedostoja Magentossa ei lisätä app/code/local-hakemistoon vaan skin/adminhtml/default/default/js-hakemistoon (kuva 38). Ponnahtusikkunan tyylit luotiin vastaavasti skin/adminhtml/default/default/css-hakemistoon.



Kuva 38. Moduulien tyylit ja Javascript-tiedostot luodaan skin-kansioon.

Kuvassa 38 kansiot ovat adminhtml-kansion alla. Näin Magento osaa tunnistaa Javascript-tiedoston oikeassa näkymässä eli tässä tapauksessa hallintapaneelissa eikä itse verkkokaupassa. Kuvassa 39 näkyvät Javascript-funktiot, joilla lisätään attribuutit

tekstikenttiin, klikkaustapahtumat Open Graph Markup -muuttujille ja niiden lisäksi tekstikenttiin.

```
function addAttributeToInput(selectedItem){
  //Get inputText
  var textInput = $(selectedItem).closest('.value').childElements()[2];

  //Insert attribute to text field
  textInput.value += document.getElementById(selectedItem.id).innerText;

  //Closing popup after selection
  openedPopup.style.display = 'none';
  openedPopup = null;
}

//Add onclick events to attributes
function addOnClickToAttributes(){
  var contentDivs = document.getElementsByClassName('metadata_and_open_graph_content');
  var anchorTags = [];
  var allAnchorTagsLength;

  var i, j;
  var contentDivsLength = contentDivs.length;

  //Get every anchor element from divs
  for(i = 0; i < contentDivsLength; i++){
    allAnchorTagsLength = contentDivs[i].getElementsByTagName('a').length;

    //Get every anchor tags
    for(j = 0; j < allAnchorTagsLength; j++){
      anchorTags.push(contentDivs[i].getElementsByTagName('a')[j]);
    }
  }

  //Adding onlick
  var anchorTagsLength = anchorTags.length;
  for(i = 0; i < anchorTagsLength; i++){
    anchorTags[i].onclick = function () {
      addAttributeToInput(this);
    }
  }
}
```

Kuva 39. Javascript-funktiot, jolla lisätään attribuuteille klikkaustapahtumat ja lisätään attribuutit tekstikenttiin.

Javascript-tiedosto on konfiguroitava, jotta se latautuu hallintapaneelissa. Konfiguraatio-tiedosto lisätään app/design/adminhtml/default/default/layout/-hakemistoon local.xml-tiedostoon. Kuvassa 40 näkyy kyseiseen hakemistoon luodun tiedoston sisältö.

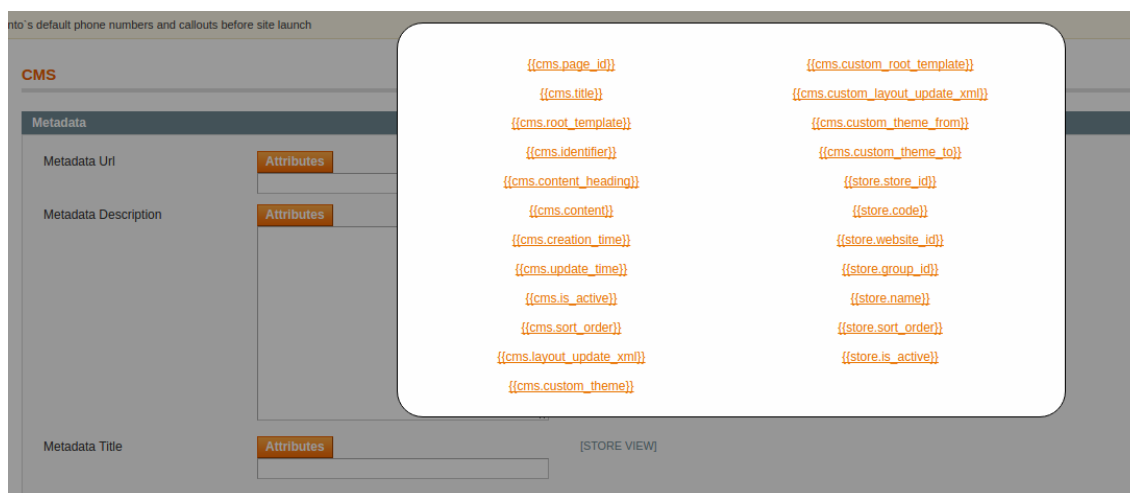
```

<?xml version="1.0"?>
<layout>
  <adminhtml_system_config_edit>
    <reference name="head">
      <action method="addItem">
        <type>skin_css</type>
        <name>css/lamia_mdandog.css</name>
      </action>
      <action method="addItem">
        <type>skin_js</type>
        <name>js/lamia_mdandog.js</name>
      </action>
    </reference>
  </adminhtml_system_config_edit>
</layout>

```

Kuva 40. Konfiguraatio, jonka avulla luotu Javascript- ja CSS-tiedosto lisätään HTML-koodin head-osioon.

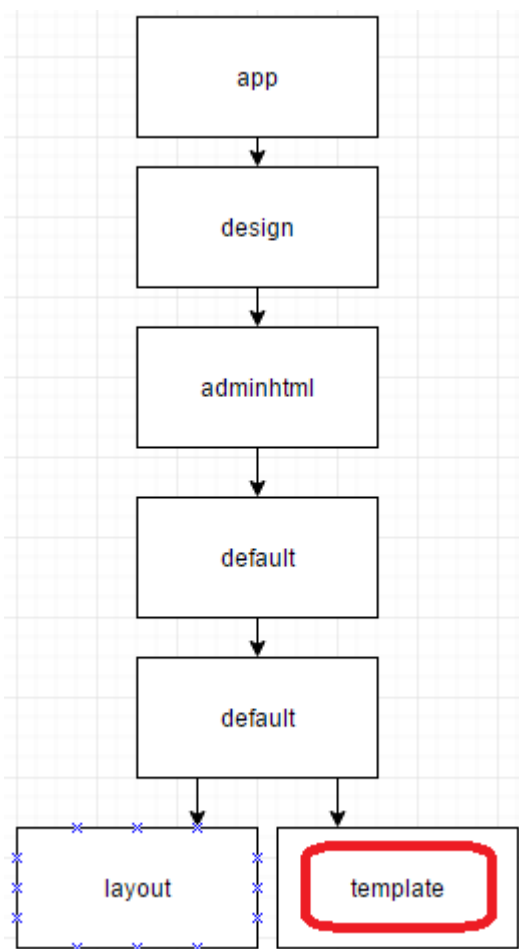
Kuvassa 40 adminhtml\_system\_config\_edit viittaa näkymään, jonka HTML-koodiin skripti lisätään ja reference-elementti referoi Magenton valmiiksi luotuun head-nimiseen Block-luokkaan. Kyseisen luokan toteutus löytyy Magenton core-tiedostoissa. Referenssin sisällä on määritelty JavaScript-tiedoston lisäys. Kuvassa 41 näkyy JavaScriptin- ja CSS-tiedoston tuottama ponnausikkuna.



Kuva 41. Javascript-tiedoston tuottama ponnausikkuna.

Kuvan 41 attribuutit saadaan kuitenkin näkymään template-tiedoston avulla. Template-tiedosto on hyvin yksinertainen PHTML-tiedosto, joka käyttää aiemmin luotujen Block-

luokkien `_getElementHtml`-funktion generoimaa HTML-koodia hyväkseen. Template-tiedostot luodaan `app/design/adminhtml/default/default/template/lamiametadata/-`hakemistoon (kuva 42).



Kuva 42. Template-tiedostot luodaan design-kansioon eikä code-kansioon, jossa kaikki koodi sijaitsee.

Luotu template-tiedosto on hyvin yksinkertainen. Kuvassa 43 näkyy template-tiedoston toteutus. Kyseisen template-tiedoston avulla saadaan attribuutit renderöityä sivulle, kuten kuvassa 41 nähtiin.



```

<div class="metadata_and_open_graph_modal">
  <div class="metadata_and_open_graph_content">
    <ol>
      <?php $fields = $this->getData('attributes');?>
      <?php foreach ($fields as $field): ?>
        <li>
          <a id="<?php echo $field;?>"><?php echo $field;?></a>
        </li>
      <?php endforeach;?>
    </ol>
  </div>
</div>

```

Kuva 43. Template-tiedoston toteutus attribuuttien näyttämiseen ponnahdusikkunassa.

Kuvassa 43 Template-tiedosto iteroi kaikki attribuutit läpi ja luo niistä järjestetyn listan. Lista näytetään, kun verkkokaupan omistaja painaa attributes-painiketta. Kun listalta painetaan haluttua attribuuttia, niin se lisätään painiketta vastaavaan tekstikenttään. Kuvassa 44 näkyvät Open Graph Markup -asetukset, joihin on asetettu attribuutteja.

The screenshot shows the Magento Admin configuration page for 'Products'. On the left is a navigation menu with categories like 'GENERAL', 'CATALOG', 'CUSTOMERS', 'SALES', and 'CONFIGURE METADATA AND OPEN GRAPH MARKUP'. The main content area is titled 'Products' and is divided into two sections: 'Metadata' and 'Open Graph Markup'. Each section contains several fields with 'Attributes' buttons and 'STORE VIEW' labels. The 'Metadata' section includes fields for 'Metadata Url', 'Metadata Description', and 'Metadata Title'. The 'Open Graph Markup' section includes fields for 'Open Graph Markup Url', 'Open Graph Markup Description', and 'Open Graph Markup Title'. The 'Open Graph Markup Description' field contains a rich text editor with pre-defined content like 'Example description {{product.price}}' and 'Group price: {{product.group\_price}}'.

Kuva 44. Open Graph Markup -asetukset, joihin on asetettu tuotteisiin liittyviä attribuutteja.

## 7.8 Observer-tapahtumien konfigurointi

Seuraavaksi haluttiin lisätä ominaisuus, jolla tallennetut Open Graph Markup -asetusten muuttujat käsitellään tekstiksi. Ominaisuuden lisääminen ei ollut kuitenkaan yksinkertaista vaan oli hyödynnettävä Magenton Observer -tapahtumia. Kuvassa 45 on Observer-luokkien ja tapahtumien konfigurointi.

```
<events>
  <adminhtml cms_page_edit_tab_meta_prepare_form>
    <observers>
      <lamia_metadata_and_open_graph>
        <type>singleton</type>
        <class>Lamia_MetadataAndOpenGraph_Model_Observer</class>
        <method>openGraphMarkupAttributes</method>
      </lamia_metadata_and_open_graph>
    </observers>
  </adminhtml cms_page_edit_tab_meta_prepare_form>
  <catalog_product_load_after>
    <observers>
      <lamia_metadata_and_open_graph>
        <type>singleton</type>
        <class>Lamia_MetadataAndOpenGraph_Model_Type_Observer</class>
        <method>saveEditableFields</method>
      </lamia_metadata_and_open_graph>
    </observers>
  </catalog_product_load_after>
  <catalog_product_save_before>
    <observers>
      <lamia_metadata_and_open_graph>
        <type>singleton</type>
        <class>Lamia_MetadataAndOpenGraph_Model_Observer</class>
        <method>saveAdditionalFields</method>
      </lamia_metadata_and_open_graph>
    </observers>
  </catalog_product_save_before>
</events>
```

Kuva 45. Observer-luokkien ja tapahtumien määrittely config.xml-tiedostossa.

Kuvassa 45 nähdään, että tapahtumat on luotu CMS-sivujen muokkausnäköön ja eriliset tapahtumat ennen tuotteiden tallennusta ja tuotteiden tallennusten jälkeen. Observer-elementin sisällä määritellään myös, mistä kyseinen Observer-luokka löytyy ja mikä luokan funktio ajetaan, kun tapahtuma on tapahtunut.

## 7.9 Käsittelyluokkien toteutus

Moduulissa on kaksi eri Observer-luokkaa, jotka on luotu moduulin Model-kansioon, kuten kuvassa 31 nähtiin. Yhdessä luokassa tallennetaan käsittelemättömät muuttujat eli {{store.name}}-muuttujaa ei käsitellä muokattavissa kentissä. Kyseiset muuttujat käsitellään verkkokaupan omistajalta piilossa oleviin editable-etuliitteellä oleviin muuttujiin, jotka luotiin aikaisemmin asennusskripteillä. Kuvassa 46 näkyy saveEditableFields-funktion toteutus, joka hakee ja tallentaa käsittelemättömiin kenttiin annetut tiedot, mikäli niille ei ollut aiemmin mitään arvoa tallennettu.

```
class Lamia_MetadataAndOpenGraph_Model_Type_Observer extends Mage_Core_Model_Abstract
{
    public function saveEditableFields(Varien_Event_Observer $observer)
    {
        $model = $observer->getEvent()->getData('data_object');
        $modelResourceName = $model->getResourceName();
        $storeConfig = $this->getStoreConfig($modelResourceName);
        $fieldsMapping = Mage::helper('lamia_metadata_and_open_graph')->getSystemConfigOpenGraphFieldsMapping();

        //Open Graph
        foreach ($fieldsMapping as $editableField => $ogField){
            if(!$model->getData($editableField)){
                $model->setData($editableField, $storeConfig['open_graph_markup'][$ogField]);
            }
        }

        $this->setMetadataFields($model, $storeConfig);
    }
}
```

Kuva 46. SaveEditableFields-funktion toteutus, jolla muokatut kentät tallennetaan kategoriaille ja tuotteille.

Kun Open Graph Markup -tiedot on saatu tallennettua, niin kenttien sisältämät muuttujat piti vielä käsitellä niin, että esimerkiksi {{product.name}}-muuttujan tilalle tulisi tuotteen nimi. Tämä toiminnallisuus toteutettiin omilla Parse-malliluokilla.

Parse-luokkien päätarkoituksena oli hakea tallennetut Open Graph Markup -asetukset kyseiseltä sivulta, tunnistaa kentissä olevat muuttujat ja käsitellä nämä muuttujat ennen näiden tallennusta. Kuvassa 47 nähdään tuotteelle tarkoitettu attribuuttien käsittelyluokka.

```

class Lamia_MetadataAndOpenGraph_Model_Parse_Product_General
  extends Lamia_MetadataAndOpenGraph_Model_Parse_Abstract
{
  /**
   * @param $model
   */
  public function parseEditableFields($model)
  {
    $parsableAttributes = $this->getParsableAttributes();

    $this->parseAttributes($model, $parsableAttributes);
    Mage::getModel('lamia_metadata_and_open_graph/parse_store_general')->parseEditableFields($model);
  }

  /**
   * @return array
   */
  public function getParsableAttributes()
  {
    return Mage::getModel('lamia_metadata_and_open_graph/type_product')->getAvailableFields();
  }
}

```

Kuva 47. Tuotteen Open Graph Markup -muuttujien käsittelyä varten toteutettu luokka.

Kuvan 47 luokka on hyvin yksinkertainen, koska itse toiminallisuudet on toteutettu kyseisen luokan perivään abstraktiin luokkaan. Käsittelyä varten tehdyt luokat on siis tehty tuotteiden lisäksi kategoria- ja CMS-sivuille. Tuotteilla on myös muutamia omia attribuutteja, joille piti toteuttaa omat käsittelyluokat erikseen. URL-muuttujia varten täytyi tehdä oma käsittelyluokka. Kyseisen luokkien toteutukset toimivat samalla periaatteella kuin kuvan 48 toteutus. Abstraktiin luokkaan on puolestaan toteutettu yksi funktio parseAttributes-funktio, joka käsittelee Open Graph Markup -kentissä olevat muuttujat, välittä-mättä siitä, kuuluuko se tuotteille tai kategorioille. ParseAttributes-funktion toteutus näkyy kuvassa 48.

```

public function parseAttributes(Varien_Object $model, array $parsableAttributes, $separator = ',')
{
    $modelResourceName = $model->getResourceName();
    $openGraphFieldsMapping = Mage::helper('lamia_metadata_and_open_graph')->getParseMapping();
    $customAttributeMapping = Mage::helper('lamia_metadata_and_open_graph')->getCustomAttributesMapping();
    $sparsedString = null;

    //Parsing Og fields
    foreach ($openGraphFieldsMapping as $editableField => $hiddenField){
        foreach ($parsableAttributes as $item){
            $strippedItem = str_replace(array("{", "}"), "", $item);
            list($prefix, $attribute) = explode(".", $strippedItem);

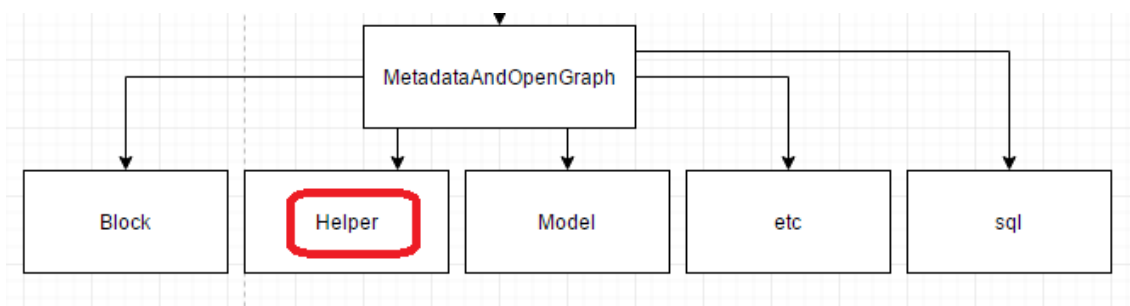
            foreach ($customAttributeMapping as $modelResourceKey => $customAttributeArray){
                foreach ($customAttributeArray as $index => $customAttribute){
                    if(key_exists($modelResourceName, $customAttributeMapping) && $attribute === $customAttribute){
                        $replaceString = Mage::getModel('lamia_metadata_and_open_graph/parse_'.$prefix.'_'.$customAttribute)
                            ->getParsedString($model->getData($attribute), $separator);
                        $sparsedString = str_replace($item, $replaceString, $model->getData($hiddenField));
                    } else if (!is_array($model->getData($attribute))){
                        $replaceString = $this->getReplaceString($attribute, $model);
                        $sparsedString = str_replace($item, $replaceString, $model->getData($hiddenField));
                    }
                }
            }

            if($sparsedString){
                $model->setData($hiddenField, $sparsedString);
            }
            $sparsedString = null;
        }
    }
}

```

Kuva 48. ParseAttributes-funktio, joka on vastuussa muuttujien käsittelystä.

Kuvan 48 funktion toteutus on pitkälti merkkijonojen käsittelyä. Funktion alussa käytetään kuitenkin Helper-luokkia. Helper-luokat sisältävät funktioita, jotka eivät kuulu tai sovi mihinkään luokkaan tai tiedostoon. Helper-luokat luodaan moduulin Helper-kansioon (kuva 49).



Kuva 49. Moduulin Helper-luokat luodaan moduulissa sijaitsevaan Helper-kansioon.

Funktioiden tarkoitus on vain avustaa jonkun tietyn kokonaisuuden saavuttamiseksi. Tässä tapauksessa Helper-luokka sisältää funktioita, jotka yksinkertaistavat käsittelyä varten tehtyjä for-silmukoita. Kuvassa 50 nähdään, kuinka Helper-luokkaan on toteutettu pelkästään funktioita, jotka palauttavat iteroimista helpottavia assosiativisia taulukkoja.

```

class Lamia_MetadataAndOpenGraph_Helper_Data extends Mage_Core_Helper_Abstract
{
    public function getOpenGraphFieldsMapping()
    {
        return array(
            'editable_og_url' => 'og_url',
            'editable_og_title' => 'og_title',
            'editable_og_image' => 'og_image',
            'editable_og_description' => 'og_description'
        );
    }

    public function getSystemConfigOpenGraphFieldsMapping()
    {
        return array(
            'editable_og_url' => 'og_url',
            'editable_og_title' => 'og_title',
            'editable_og_description' => 'og_description'
        );
    }

    public function getDefaultMetadataMapping()
    {
        return array(
            'meta_title' => 'metadata_title',
            'meta_description' => 'metadata_description'
        );
    }
}

```

Kuva 50. Osa Helper-luokan funktioista

Kyseisten luokkien toteutuksien jälkeen piti ne ottaa vielä käyttöön. Observer-luokan tarkoituksena oli kuunnella tapahtumaa, joka tapahtuu aina ennen kategorian, tuotteen tai CMS-sivun tallennusta. Kun kyseinen tapahtuma tapahtui, niin ajettiin funktio `saveAdditionalFields`, joka hoiti itse muuttujien käsittelyn ennen niiden tallentamista tietokantaan.

## 7.10 Testaaminen

Magentossa-moduuleille luodaan testit kuten muissakin sovelluskehyksissä. Testit luodaan kokonaan erillään moduulista. Yksikkötestien avulla testataan kaikki ohjelmassa luodut funktiot, jotta voidaan varmistua siitä, että verkkokaupan toiminnot toimivat halulla tavalla. Magentossa luodaan myös usein funktionaalisia testejä, mutta tässä tapauksessa moduulista päätettiin jättää pois nämä testit ja todettiin, että yksikkötesteillä pärjätään tämän moduulin osalta.

Moduulin yksikkötestit tehtiin PHPUnit-sovelluskehysten avulla. Yksikkötestit ovat luonteeltaan hyvin tyypillisiä testejä, jossa ennen testejä alustetaan tarvittavat datat, jonka

jälkeen testit suoritetaan. Testeissä ainoana ongelmana ilmeni suojattujen eli protected-määreellä luotujen funktioiden testaukset. Testifunktioilla ei ollut oikeutta päästä käsiksi ja ajaa protected-määreellä luotuja funktioita. Tämä ongelman ratkaisuun oli käytettävä PHP:n Reflection API:a. Kuvassa 51. Nähdään erään funktion testi käyttäen Reflection API:a.

```
public function testAddPrefixToFields()
{
    $testArray = array(
        'attributeA',
        'attributeB',
        'attributeC'
    );

    $resultArray = array(
        '{{test.attributeA}}',
        '{{test.attributeB}}',
        '{{test.attributeC}}'
    );

    $testPrefix = 'test';

    $model = Mage::getModel('lamia_metadata_and_open_graph/type_product');

    $reflection = new ReflectionClass(get_class($model));
    $method = $reflection->getMethod('addPrefixToFields');
    $method->setAccessible(true);

    $testResult = $method->invokeArgs($model, array($testArray, $testPrefix));

    $this->assertSame($resultArray, $testResult);
}
```

Kuva 51. Reflection API:n käyttö testissä, jotta suojattu metodi voidaan ajaa.

Kuvassa 51 huomataan, että Reflection API:n avulla päästään projektin luokkiin ja sen funktioihin käsiksi. Luokan-funktiot voidaan myös ajaa, vaikka ne olisi suojattuja protected- tai private-määreellä. Reflection API:lla pääsee myös käsiksi luokan suojattuihin muuttujiin. Käytännössä ohjelmoijalla on täydet oikeudet muokata luokkaa ja sen muuttujia sekä funktioita. Reflection API:a käytetään harvoin mutta se on todella hyödyllinen projektin virheiden havaitsemiseen tai tässä tapauksessa testaukseen. Reflection API:lla pääsee muun muassa myös projektissa luotuihin PHPDoc-kommenteihin. Reflection API on myös toteutettu muissa kielissä kuten Javassa. [40.]

## 8 Johtopäätökset ja yhteenveto

Moduulia kehittäessä opin erittäin paljon siitä, miten Magentolla kehitetään ominaisuuksia verkkokauppaan. Magento on suuri alusta ja sen oppiminen täysin vie vielä muutamaa vuoden aikaa. Insinööriyön aikana kehitin toiminnallisuutta hallintapaneeliin. Itse verkkokaupan näkymän ohjelmointiin liittyi monta asiaa jotka pitää ottaa huomioon.

Vaikeinta Magento-moduulin kehityksessä oli sen arkkitehtuurin ymmärtäminen ja eri tiedostojen nimeämiskäytännöt. Konfiguraatitiedostojen luominen ja ymmärtäminen vei myös oman aikansa. Magento vaikuttaa alussa liian laajalta kokonaisuudelta, mutta mitä enemmän siitä ymmärtää sitä enemmän ymmärtää myös sen, kuinka hyvät työkalut ja menetelmät Magento antaa verkkokauppojen luomiseen ja laajennukseen. Magento käyttää 12 eri suunnittelumallia, jotka käytiin läpi insinööriyössä. Näiden suunnittelumallien ymmärtäminen auttaa ohjelmoijaa ymmärtämään, miten Magento toimii ja helpottaa täten myös Magentolla ohjelmointia.

Moduuliin saatiin toteutettua sen logiikka, mutta seuraavaksi pitäisi vielä saada Open Graph Markup -tiedot näkymään verkkokaupassa. Tämän toteuttamista ei ole vielä toteutettu ja se olisi seuraava askel kyseisen moduulin kehityksessä. Moduulin valmistuksessa on tarkoituksena lisätä kyseinen moduuli haluttuihin verkkokauppoihin eri asiakkaille.

Insinööriyön aikana luotiin Open Graph Markup -moduuli Magentolle, jonka avulla verkkokaupan omistajat voivat lisätä verkkokauppaan omat Open Graph Markup -tiedot. Moduulia kehittäessä opin Magento verkkokauppa-alustan arkkitehtuurin, mitä eri suunnittelumalleja se käyttää ja mitä kyseisten suunnittelumallien päätavoitteena on. Moduulin täytyy enää kehittää käsiteltyjen Open Graph Markup -tietojen käyttöönotto eri tuote-, kategoria- ja CMS-sivuille. Tämän jälkeen on vielä kehitettävä loput testit moduulia varten, jonka jälkeen moduuli voidaan ottaa käyttöön tulevilla asiakasprojekteilla.

Verkkokauppa-alustojen vertailun tarkoituksena oli antaa kuva siitä, mitä muita vaihtoehtoja markkinoilta löytyy. Vertailussa vertailtiin kolme suosituinta verkkokauppa-alustaa, jotka olivat Magento, Shopify ja BigCommerce. Magento oli monessa asiassa parempi kuin muut. Magento ei kuitenkaan ollut käyttäjäystävällisempi ja helpompi ylläpitää kuin muut vertailussa olleet verkkokauppa-alustat. Vaikka Magento Community Edition



on ilmainen ja siihen löytyy laaja valikoima eri teemoja verkkokauppaa varten, niin pienyrityksillä ei välttämättä ole aikaa tai resursseja ylläpitää omaa palvelinta. Magento onkin tarkoitettu lähinnä keskikokoisille tai suurille yrityksille.

Insinöörityön suurimpina haasteina oli Magento-arkkitehtuurin ja työprosessin ymmärtäminen. Suunnittelumallien pääperiaatteiden ymmärtäminen auttoi moduulin kehityksessä kuitenkin paljon. Magento tosiaan hyödyntää kahtatoista eri suunnittelumallia ja näiden ymmärtäminen ei pelkästään auta Magenton kanssa työskentelyssä vaan myös auttaa ymmärtämään, kuinka voi ohjelmoida uudelleenkäytettävää ja helposti ylläpidettävää koodia.

## Lähteet

- 1 Magento is not Zend Framework. Verkkodokumentti. <http://www.webguys.de/magento-1/tuerchen-16-magento-is-not-zend-framework> Luettu: 1.6.2016.
- 2 The history of Magento. Verkkodokumentti. <https://www.onlinemediadi-rect.co.uk/magento/> Luettu: 1.6.2016.
- 3 Ecommerce Platforms: Choosing The Best Ecommerce Platform Partner For Your Business <https://www.abetterlemonadestand.com/ecommerce-platforms/> Luettu: 1.6.2016.
- 4 BigCommerce Review 2016 <http://stylefactoryproductions.com/blog/bigcommerce-review> Luettu: 11.9.2016.
- 5 Wix vs Shopify vs WooCommerce vs BigCommerce vs Magento vs Jimdo <https://www.codeinwp.com/blog/shopify-vs-magento-vs-woocommerce-vs-bigcommerce-vs-wix/> Luettu: 25.9.2016.
- 6 Shopify vs. Magento Community Edition – Ecommerce Platform Comparison. Verkkodokumentti. <http://ecommerce-platforms.com/compare/shopify-vs-magento-community-edition-ecommerce-platform-comparison> Luettu: 2.6.2016.
- 7 A Beginner's guide to Design Patterns <http://code.tutsplus.com/articles/a-beginners-guide-to-design-patterns--net-12752> Luettu: 10.9.2016.
- 8 MVC architecture. Verkkodokumentti. [https://developer.chrome.com/apps/app\\_frameworks](https://developer.chrome.com/apps/app_frameworks) Luettu: 15.6.2016.
- 9 Magento Controller Dispatch and Hello World. Verkkodokumentti. [http://alans-torm.com/magento\\_controller\\_hello\\_world](http://alans-torm.com/magento_controller_hello_world) Luettu: 15.6.2016.
- 10 Magento MVC Architecture. Verkkodokumentti. <http://www.php24.in/2010/11/magento-mvc-architecture.html> Luettu: 15.6.2016.
- 11 Types of MVC: Conventional v/s Configurable <http://www.xpertdeveloper.com/2013/05/types-of-mvc-architecture/> Luettu: 10.9.2016.
- 12 An Introduction to the Front Controller Pattern, Part 1 <https://www.sitepoint.com/front-controller-pattern-1/> Luettu: 4.9.2016.
- 13 Magento Design Patterns – Part 2: Front Controller <http://www.coolryan.com/magento/2012/05/20/magento-design-patterns-part-2-front-controller/> Luettu: 4.9.2016.

- 14 Design Pattern – Factory Pattern [http://www.tutorialspoint.com/design\\_pattern/factory\\_pattern.htm](http://www.tutorialspoint.com/design_pattern/factory_pattern.htm) Luettu: 4.9.2016.
- 15 Magento Design Patterns – Part 3: Factory <http://www.coolryan.com/magento/2012/05/21/magento-design-patterns-part-3-factory/> Luettu: 4.9.2016.
- 16 PHP Pattern, Part I <http://www.sitecrafting.com/blog/php-patterns-part/> Luettu: 4.9.2016.
- 17 Magento Design Patterns – Part 5: Registry <http://www.coolryan.com/magento/2012/06/04/magento-design-patterns-part-5-registry/> Luettu: 4.9.2016.
- 18 Design Pattern – Singleton Pattern [http://www.tutorialspoint.com/design\\_pattern/singleton\\_pattern.htm](http://www.tutorialspoint.com/design_pattern/singleton_pattern.htm) Luettu: 4.9.2016.
- 19 Magento Design Patterns – Part 4: Singleton <http://www.coolryan.com/magento/2012/05/22/magento-design-patterns-part-4-singleton/> Luettu: 4.9.2016.
- 20 Design Patterns – Prototype Pattern [http://www.tutorialspoint.com/design\\_pattern/prototype\\_pattern.htm](http://www.tutorialspoint.com/design_pattern/prototype_pattern.htm) Luettu: 4.9.2016.
- 21 Magento Design Patterns – Part 6: Prototype <http://www.coolryan.com/magento/2014/02/18/magento-design-patterns-part-6-prototype/> Luettu: 4.9.2016.
- 22 Object Pool <http://www.oodeesign.com/object-pool-pattern.html> Luettu: 4.9.2016.
- 23 Magento Design Patterns – Part 7: Object Pool <http://www.coolryan.com/magento/2014/02/19/magento-design-patterns-part-7-object-pool/> Luettu: 4.9.2016.
- 24 Design Pattern – Iterator Pattern [http://www.tutorialspoint.com/design\\_pattern/iterator\\_pattern.htm](http://www.tutorialspoint.com/design_pattern/iterator_pattern.htm) Luettu: 4.9.2016.
- 25 Magento Design Patterns – Part 8: Iterator <http://www.coolryan.com/magento/2014/02/20/magento-design-patterns-part-8-iterator/> Luettu: 4.9.2016.
- 26 Magento Design Patterns – Part 9: Lazy Loading <http://www.coolryan.com/magento/2014/02/26/magento-design-patterns-part-9-lazy-loading/> Luettu: 4.9.2016.
- 27 Magento Design Patterns – Part 10: Service Locator <http://www.coolryan.com/magento/2014/02/28/magento-design-patterns-part-10-service-locator/> Luettu: 4.9.2016.
- 28 Magento Design Patterns – Part 11: Module <http://www.coolryan.com/magento/2014/03/03/magento-design-patterns-part-11-module/> Luettu: 4.9.2016.

- 29 Observer [https://sourcemaking.com/design\\_patterns/observer](https://sourcemaking.com/design_patterns/observer) Luettu: 4.9.2016.
- 30 Custom Events in Magento With the Observer Pattern. [code.tutsplus.com/tutorials/custom-events-in-magento-with-the-observer-pattern--cms-22120](http://code.tutsplus.com/tutorials/custom-events-in-magento-with-the-observer-pattern--cms-22120) Luettu: 2.7.2016.
- 31 Event System in Magento 1 vs. Magento 2. <https://www.atwix.com/magento-2/event-system-m1-vs-m2/> Luettu: 29.6.2016.
- 32 Magento Design Patterns – Part 12: Observer <http://www.coolryan.com/magento/2014/03/04/magento-design-patterns-part-12-observer/> Luettu: 4.9.2016.
- 33 MacGregor, Allan. 2013. Magento PHP Developer's Guide. Birmingham: Packt Publishing Ltd.
- 34 Magento Custom Module Development <http://code.tutsplus.com/tutorials/magento-custom-module-development--cms-20643> Luettu: 10.9.2016.
- 35 Folders Structure and Database System in Magento. Verkkodokumentti. <http://tipsforbeginner.com/folders-structure-and-database-system-in-magento-part-1> Luettu: 11.6.2016.
- 36 What You Need to Know About Open Graph Meta Tags for Total Facebook and Twitter Mastery <https://blog.kissmetrics.com/open-graph-meta-tags/> Luettu: 10.9.2016.
- 37 A Guide to Sharing for Webmasters. Verkkodokumentti. <https://developers.facebook.com/docs/sharing/webmasters> Luettu: 17.6.2016.
- 38 Magento for Developers:Part 1 – Introduction to Magento. Verkkodokumentti. <http://devdocs.magento.com/guides/m1x/magefordev/mage-for-dev-1.html> Luettu: 17.6.2016.
- 39 Magento for Developers: Part 6 – Magento Setup Resources. <http://devdocs.magento.com/guides/m1x/magefordev/mage-for-dev-6.html> Luettu: 29.6.2016.
- 1 What is Reflection in PHP? <http://culttt.com/2014/07/02/reflection-php/> Luettu: 29.6.2016.

**Config.xml tiedoston toteutus**

```

<?xml version="1.0"?>
<config>
  <modules>
    <Lamia_MetadataAndOpenGraph>
      <version>0.1.0</version>
    </Lamia_MetadataAndOpenGraph>
  </modules>
  <global>
    <models>
      <lamia_metadata_and_open_graph>
        <class>Lamia_MetadataAndOpenGraph_Model</class>
        <resourceModel>lamia_metadata_and_open_graph_re-
source</resourceModel>
      </lamia_metadata_and_open_graph>
      <lamia_metadata_and_open_graph_resource>
        <class>Lamia_MetadataAndOpenGraph_Model_Re-
source</class>
      </lamia_metadata_and_open_graph_resource>
    </models>
    <events>
      <adminhtml cms_page_edit_tab_meta_prepare_form>
        <observers>
          <lamia_metadata_and_open_graph>
            <type>singleton</type>
            <class>Lamia_MetadataAndOpenGraph_Model_Ob-
server</class>
            <method>openGraphMarkupAttributes</method>
          </lamia_metadata_and_open_graph>
        </observers>
      </adminhtml cms_page_edit_tab_meta_prepare_form>
      <catalog_product_load_after>
        <observers>
          <lamia_metadata_and_open_graph>
            <type>singleton</type>
            <class>Lamia_MetadataAndOpen-
Graph_Model_Type_Observer</class>
            <method>saveEditableFields</method>
          </lamia_metadata_and_open_graph>
        </observers>
      </catalog_product_load_after>
      <catalog_product_save_before>
        <observers>
          <lamia_metadata_and_open_graph>
            <type>singleton</type>
            <class>Lamia_MetadataAndOpenGraph_Model_Ob-
server</class>
            <method>saveAdditionalFields</method>
          </lamia_metadata_and_open_graph>
        </observers>
      </catalog_product_save_before>
      <catalog_category_load_after>
        <observers>
          <lamia_metadata_and_open_graph>
            <type>singleton</type>

```

```

        <class>Lamia_MetadataAndOpen-
Graph_Model_Type_Observer</class>
        <method>saveEditableFields</method>
        </lamia_metadata_and_open_graph>
    </observers>
</catalog_category_load_after>
<catalog_category_save_before>
    <observers>
        <lamia_metadata_and_open_graph>
            <type>singleton</type>
            <class>Lamia_MetadataAndOpenGraph_Model_Ob-
server</class>
                <method>saveAdditionalFields</method>
                </lamia_metadata_and_open_graph>
            </observers>
        </catalog_category_save_before>
        <cms_page_load_after>
            <observers>
                <lamia_metadata_and_open_graph>
                    <type>singleton</type>
                    <class>Lamia_MetadataAndOpen-
Graph_Model_Type_Observer</class>
                        <method>saveEditableFields</method>
                        </lamia_metadata_and_open_graph>
                    </observers>
                </cms_page_load_after>
                <cms_page_save_before>
                    <observers>
                        <lamia_metadata_and_open_graph>
                            <type>singleton</type>
                            <class>Lamia_MetadataAndOpenGraph_Model_Ob-
server</class>
                                <method>saveAdditionalFields</method>
                                </lamia_metadata_and_open_graph>
                            </observers>
                        </cms_page_save_before>
                    </events>
                <blocks>
                    <lamia_metadata_and_open_graph>
                        <class>Lamia_MetadataAndOpenGraph_Block</class>
                    </lamia_metadata_and_open_graph>
                </blocks>
                <helpers>
                    <lamia_metadata_and_open_graph>
                        <class>Lamia_MetadataAndOpenGraph_Helper</class>
                    </lamia_metadata_and_open_graph>
                </helpers>
                <resources>
                    <lamia_metadata_and_open_graph_setup>
                        <setup>
                            <module>Lamia_MetadataAndOpenGraph</module>
                            <class>Mage_Catalog_Model_Re-
source_Eav_Mysql4_Setup</class>
                        </setup>
                        <connection>
                            <use>core_setup</use>
                        </connection>
                    </lamia_metadata_and_open_graph_setup>
                    <lamia_metadata_and_open_graph_read>

```

```
        <connection>
            <use>core_read</use>
        </connection>
    </lamia_metadata_and_open_graph_read>
    <lamia_metadata_and_open_graph_write>
        <connection>
            <use>core_write</use>
        </connection>
    </lamia_metadata_and_open_graph_write>
</resources>
</global>
<adminhtml>
    <layout>
        <updates>
            <lamia_metadataandopengraph>
                <file>lamia_metadata_and_open_graph.xml</file>
            </lamia_metadataandopengraph>
        </updates>
    </layout>
</adminhtml>
</config>
```