TEKNILLINEN KORKEAKOULU

Elektroniikan, tietoliikenteen ja automaation tiedekunta

Tietoliikenne- ja tietoverkkotekniikan laitos

Juha Havu

Dynaamisen oliomallin suunnittelu ja toteutus

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 14.4.2008

Työn valvoja: Professori Markku Syrjänen

Työn ohjaaja: Kehityspäällikkö, DI Dennis Hagström

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Juha Havu	
Työn nimi:	Dynaamisen oliomallin suunnittelu ja toteutus	
Päivämäärä:	14.4.2008	**Sivumäärä:** 62 + 2
Osasto:	Elektroniikan, tietoliikenteen ja automaation tiedekunta	
Professuuri:	T-93 Tietämystekniikka	
Työn valvoja:	Professori Markku Syrjänen	
Työn ohjaaja:	Kehityspäällikkö, DI Dennis Hagström	

Tämä diplomityö keskittyy dynaamiseen oliomalliin ja sen hyödyntämiseen ohjelmistoalustaa kehitettäessä. Diplomityö tutkii dynaamista oliomallia kirjallisuudessa ja tieteellisissä julkaisuissa esitettyjen suunnittelun ratkaisumallien avulla ja pyrkii soveltamaan näitä ratkaisumalleja ohjelmistoalustan toteutukseen.

Diplomityö on jaettu kolmeen osaan: tutkimusprojektiin, toteutukseen ja yhteenvetoon. Tutkimusprojektiosiossa tutustutaan kirjallisuuteen ja olemassa oleviin tekniikoihin, joiden avulla saavutetaan dynaamisen oliomallin kaltainen konfiguroitava järjestelmä. Toteutusosiossa käytetään hyväksi tutkimusprojektin tietoja ja toteutetaan oma dynaamiseen oliomalliin perustuva konfiguroitava ohjelmistoalusta. Yhteenvedossa arvioidaan diplomityön onnistumista ja verrataan toteutusta kirjallisuudessa esitettyihin ratkaisuihin sekä esitetään eriävyydet ja mahdolliset parannukset näihin ratkaisuihin.

Avainsanat:	dynaaminen oliomalli, suunnittelun ratkaisumallit, ASP.NET, C#, Microsoft SharePoint	**Julkaisukieli:** suomi

HELSINKI UNIVERSITY OF TECHNOLOGY ABSTRACT OF MASTER'S THESIS

Author:	Juha Havu
Title of the thesis:	Design and development of a dynamic object model
Date:	14.4.2008 **Number of pages:** 62 + 2
Faculty:	Faculty of Electronics, Communications and Automation
Professorship:	T-93 Knowledge Engineering
Supervisor:	Professor Markku Syrjänen
Instructor:	Development Manager, MSc Dennis Hagström

This thesis focuses on the dynamic object model and its usage for developing a software platform. The thesis examines design patterns introduced in literature and scientific publications and tries to employ them in the development of a software platform.

The thesis is divided into three parts: research project, implementation and summary. The research project focuses on finding out the techniques described in literature and scientific publications which make it possible to implement a dynamic object model -based configurable system. In the implementation part the knowledge inquired in the research project is put into practice by implementing a dynamic object model -based software platform. The summary part focuses on evaluating the thesis and comparing the implementation of the dynamic object model to the solutions described in literature.

Keywords: dynamic object model, design patterns, ASP.NET, C#, Microsoft SharePoint	**Publishing language:** Finnish

Alkusanat

Tämä diplomityö on kirjoitettu Proha Oyj:ssä. Työn tekeminen alkoi kesäkuussa 2007 ja saatiin päätökseen huhtikuussa 2008.

Erityiskiitokset työn valvojalle professori Markku Syrjäselle ohjeista ja kommenteista sekä joustavasta aikataulusta. Erityiskiitokset myös työn ohjaajalle Dennis Hagströmille neuvoista ja kommenteista sekä rakentavasta kritiikistä.

Espoossa 14.4.2008

Juha Havu

Sisällysluettelo

Kuvaluettelo

Taulukkoluettelo

Lyhenteet ja termit

.NET
.NET –arkkitehtuuri on Microsoftin ohjelmistokomponentti, joka on asennettavissa Microsoft Windows käyttöjärjestelmille.

AOM
Active Object Model / Adaptive Object Model, dynaamisesta oliomallista yleisesti käytetty lyhenne ja nimitys.

ASP.NET
Active Server Pages, dynaamisten www-sivujen, verkkosovellusten ja verkkopalvelujen luontiin tarkoitettu Microsoftin ohjelmistoympäristö.

C#
C# on Microsoftin kehittämä olio-orientoitunut ohjelmointikieli.

ECOOP
European Conference on Object-Oriented Programming, eurooppalainen vastine OOPSLA:lle.

Metatieto
Tietoa tiedosta, eli kuvailevaa ja määrittävää tietoa jostakin tietovarannosta tai sisältöyksiköstä.

OOPSLA
Conferences on Object-Oriented Programming, Systems, Languages, and Applications, olio-orientoituneeseen ohjelmistokehitykseen erikoistunut konferenssi.

Sovellussuuntautunut/sovellusaluekohtainen kieli
Ohjelmointikieli, jossa on sovellusaluekohtaisia rakenteita, tai joka on tarkoitettu tietynlaisten ongelmien ratkaisemiseen.

Suunnittelun ratkaisumalli
Suunnittelun ratkaisumalli (engl. design pattern) on hyväksi koettu tapa ratkaista jokin oliosuunnittelun ongelma.

Web Part
Web Part on koottu joukko ASP.NET komponentteja, joiden avulla luodaan loppukäyttäjälle mahdollisuus muuttaa WWW -sivujen sisältöä

XML
Extensible Markup Language on merkintäkieli, jossa tiedon merkitys on kuvattu tiedon sekaan.

XSLT
Extensible Stylesheet Language Transformations, XML:ään pohjautuva kieli XML –dokumenttien muuttamiseksi toisiksi XML –dokumenteiksi.

1 Johdanto

Vuosi 2006 oli suurien muutosten aikaa Proha Oyj:ssä. Aikaisemmin Prohan ohjelmistoalustoina olivat toimineet PowerBuilder –sovelluskehitysohjelmalla ja Javalla toteutetut tuotteet. Vuoden 2006 aikana Proha myi tytäryhtiönsä Artemiksen ja samalla hävisi suuri määrä vanhoja tuotteita Prohan hallinnasta. Tämän johdosta oli aika lähteä kehittämään uusia projektinhallintaa tukevia tuotteita. Vuonna 2006 Proha myös saavutti Microsoftin Gold Partneruuden, joten oli luontevaa lähteä kehittämään uutta ohjelmistoarkkitehtuuria juuri Microsoftin tarjoaman tuotekehitysalustan päälle.

Uusien tuotteiden ohjelmistoalustaa suunniteltaessa tutkittiin menneiden vuosien painolastia ja löydettiin seuraavanlaisia ongelmia aikaisemmista tuotteista ja niiden kehityksestä:

- Liiketoiminnan säännöt muuttuvat alati ja tuotteiden pitää mukautua niihin.
- Tuotteiden pitää olla konfiguroitavia ja joustavia mukautuakseen asiakkaan vaatimiin projektihallinnan prosesseihin.
- Uudet tuoteversiot pitää saada markkinoille mahdollisimman nopealla aikataululla.

Näistä lähtökohdista lähdettiin tutkimaan dynaamisen oliomallin tuomaa vapautta sovelluksen liiketoimintamallien ja sääntöjen kuvaamiseen. Toisin sanoen tarkoituksena oli kehittää ohjelmistoalusta, jossa abstraktioiden kautta liiketoiminnan mallit ja säännöt nostettaisiin osaksi tuotettavien ohjelmien tietoa sen sijaan, että tämä tieto olisi haudattuna ohjelmiston sisältämään koodiin.

Tämän diplomityön keskeisenä päämääränä on kuvata kirjallisuudessa ja tieteellisessä yhteisössä esitettyjä ratkaisuja dynaamisen oliomallin toteuttamiseen ja rakentaa Proha Oyj:lle samaiseen ideaan perustuva tuotealusta.

1.1 Diplomityön rakenne

Diplomityö on jaettu kolmeen osaan: tutkimusprojektiin, toteutukseen ja yhteenvetoon. Tutkimusprojektiosiossa tutustutaan kirjallisuuteen ja olemassa oleviin tekniikoihin, joiden avulla saavutetaan dynaamisen oliomallin kaltainen konfiguroitava järjestelmä.

Toteutusosiossa käytetään hyväksi tutkimusprojektin tietoja ja toteutetaan oma dynaamiseen oliomalliin perustuva konfiguroitava ohjelmistoalusta. Yhteenvedossa arvioidaan diplomityön onnistumista ja verrataan toteutusta kirjallisuudessa esitettyihin ratkaisuihin sekä esitetään eriävyydet ja mahdolliset parannukset näihin ratkaisuihin.

1.2 Diplomityön rajaukset

Dynaamisella oliomallilla tarkoitetaan tämän diplomityön puitteissa tiettyjen suunnittelumallien avulla saavutettavaa ohjelmistoa, jossa liiketoiminnan olioiden ja niiden sisältämien sääntöjen määritteleminen on mahdollista metatiedon avulla (Dynamic Object Model [1]). Diplomityön tarkoituksena ei ole ottaa kantaa olioiden tilan säilyttämiseen, vaan tämä toteutetaan ennalta määrätyllä Nolics nimisellä tuotteella tietokantaan.

2 Taustaa

Tässä kappaleessa esitellään yleistä taustaa diplomityöhön liittyen.

2.1 Proha Oyj

Proha Oyj on vuonna 1983 perustettu yritys, joka koostuu kahdesta liiketoimintaryhmästä; Dovre Consulting and Services ja Safran Systems. Prohan liiketoiminnan kaksi päälinjaa ovat projektijohtamisen palvelut sekä projektijohtamisen menetelmiin perustuvat ohjelmistoratkaisut. Vuonna 2006 Proha myi tytäryhtiönsä Artemiksen ja samalla suuri osa projektihallinnan tuotepaletista siirtyi pois Prohan hallinnasta. Täten syksyllä 2006 Proha aloitti uusien tuotteiden kehityksen, jonka seurauksena tämäkin diplomityö on kirjoitettu.

2.2 Projektinhallinta

Projektinhallinta ja projektijohtaminen on sovelluskehityksen kannalta haastava toimintakenttä, sillä usein yritysten projektihallinnan tarpeet ja käsitteistöt eroavat huomattavasti toisistaan. Projektihallinnan sovellusten pitää usein mukautua asiakkaiden toiveisiin ja tukea yrityksissä jo pitkälle juurtuneita projektihallinnan menetelmiä ja käytäntöjä, joiden muuttaminen uuden projektihallintaohjelmiston johdosta voi olla haasteellista tai peräti mahdotonta. Tämän lisäksi projektihallintaohjelmistojen toteuttamiseen lisäävät haasteita eri projektihallinnan standardit, jotka eriävät toisistaan niin käsitteistön kuin toimintatapojenkin osalta.

Osa A: Tutkimusprojekti

3 Diplomityön tarkoitus ja tavoitteet

Tässä kappaleessa käydään läpi joitain diplomityön tärkeimpiä tavoitteita ja selvitetään diplomityön tarkoitusperää. Näiden tavoitteiden onnistumista arvioidaan diplomityön luvussa yhdeksän.

3.1 Dynaaminen oliomalli

Diplomityön oleellisena tarkoituksena on tutustua dynaamiseen oliomalliin ja selvittää tämän mallin mahdollinen soveltuvuus projektihallintajärjestelmien määrittelemiseen. Tarkoituksena on myös selvittää mahdollisia tiedossa olevia hyötyjä ja haittoja, jotka liittyvät dynaamisen oliomallin käyttöön.

3.2 SafranOne –ohjelmistoalusta

Diplomityön toisena oleellisena tarkoituksena on kehittää dynaamiseen oliomalliin perustuva SafranOne –ohjelmistoalusta, jonka päälle pystytään rakentamaan projektihallinnan sovelluksia. Ohjelmistoalustan tulee tukea dynaamista oliomallia ja sallia mahdollisen oliomäärittelyn asettaminen ulkoisesti metatiedon avulla. Metatiedon syöttämistä varten toteutetaan sovellus, jonka avulla liiketoimintaolioiden määrittelyä ja sääntöjä voidaan muokata ajonaikana.

3.3 Teknologiset tavoitteet

Diplomityötä sivuavina tavoitteina ovat Microsoft –teknologioihin perehtyminen ja tiedon saattaminen yrityksen tuotekehityksen tietoon. Oleellisimpina teknologioina ovat .NET –sovellusympäristö sekä Microsoftin SharePoint –portaalisovellus, jonka osaksi SafranOne ohjelmistoalusta on tarkoitus integroida.

4 Kohti dynaamista oliomallia

Tässä luvussa käsitellään nykyaikaista ohjelmistokehitystä ja asioita, jotka ovat vaikuttaneet dynaamisen oliomallin syntyyn. Tarkoituksena on selventää staattisen oliomallin heikkouksia ja kankeutta nykyaikaisessa ohjelmakehityksessä ja sen muuttuvassa toimintaympäristössä.

4.1 Staattinen oliomalli

Useimmat olio-orientoituneet järjestelmät koostuvat staattisesta oliomallista. Toisin sanoen oliomalli on kiinnitetty sovellusta suunniteltaessa ja pysyy täten muuttumattomana ajonaikana [1]. Olio-orientoitunut suunnittelu synnyttää luokkia kuvaamaan eri tyyppisiä liiketoiminnan olioita ja yhdistää joukon ominaisuuksia ja metodeja niihin [7]. Liiketoiminnan olioiden kuvaaminen staattisen oliomallin avulla usein myös johtaa suureen määrään luokkia, sillä joillain toimialoilla liiketoiminta on erittäin monimutkainen. Tällaisia toimialoja ovat esimerkiksi vakuutus- ja pankkitoimiala. [4]

Nykyään ohjelmistoja kehitetään yleensä oliokeskeisillä kielillä, kuten Java ja C#, joissa luokat kuvataan koodina. Pätkä koodia kuvaa täten luokan ja on eräänlainen luokan abstraktio, joka pysyy muuttumattomana käännöksestä eteenpäin. Luokkien kuvatessa liiketoimintaa, aiheuttaa muutos itse liiketoiminnassa muutoksen sovelluksen toiminnassa. Muutokset sovelluksen toiminnassa vaativat aina koodin muutoksen ja tätä kautta myös sovelluksen kääntämisen. Muutos aiheuttaa siis sovelluksen uuden version syntymisen. Itse muutoksen lisäksi sovelluksen version julkaisuun liittyy usein myös muita tehtäviä, kuten testausta, jotka pitää suorittaa ennen kuin se voidaan toimittaa asiakkaalle. [1], [13]

Jotta jokainen sovelluksen toiminnan muutos ei aiheuttaisi tarvetta sovelluksen kääntämiselle on jo kauan käytetty erilaisia tekniikoita sovelluksen ajonaikaisen toiminnan muuttamiseksi. Näitä ovat olleet esimerkiksi konfigurointitiedostot, käynnistysparametrit ja rekisterit. Näiden tekniikoiden avulla tosin pystytään varautumaan vain ennalta määrättyihin muutoksiin sovelluksen toiminnassa. Näitä muutoksia varten on ohjelmaan koodattava tarkistuspisteitä, joissa mahdollinen konfiguraatio aiheuttaa normaalista poikkeavan sovelluksen toiminnan. Ennalta määrätyiden muutosten avulla ei kuitenkaan voida normaalein konfigurointikeinoin varautua kokonaisen oliomallin mahdolliseen

muuttumiseen ajonaikana, mikä saattaa hyvinkin olla vaatimuksena nykyaikaiselle ohjelmistolle. [3]

4.2 Nykyaikaisen ohjelmiston vaatimuksia

Ohjelmistojen vaatimukset niin teknisellä kuin toiminnallisella tasolla ovat kasvaneet suuresti sitten ohjelmistotuotannon alkuaikojen. Siinä missä ohjelmistojen perusvaatimukset, kuten tehokkuus ja turvallisuus ovat edelleen tärkeitä, ovat ajonaikainen konfiguroitavuus, mukautumiskyky ja laajennettavuus nousseet yhä tärkeämpään osaan ohjelmistojen vaatimusten pitkällä listalla. Tarpeiden saavuttamiseksi ohjelmistosuunnittelijoiden on tullut tarjota yhä joustavampia arkkitehtuureja, jotka mahdollistavat ohjelmistojen mukautumisen alati muuttuviin vaatimuksiin. [6], [8]

Nopeat muutokset asiakkaan liiketoiminnan käytännöissä ja säännöissä ovat luoneet tarpeen sovelluskehitykselle sallia nopeita muutoksia sovellusten toiminnassa [5]. Suuressa yrityksessä liiketoiminnan säännöt ja käytännöt myös vaihtelevat osastolta toiselle ja voivat jopa vaihdella projektin elinajan aikana. Tämän johdosta järjestelmien tulee myös joissain tapauksissa mukautua uusiin käyttäjävaatimuksiin ajonaikana. Asiakkaiden vaatimuksina ovat myös järjestelmien skaalautuminen pienistä käyttöönotoista suuriin, jolloin ohjelmistojen tulee olla muokattavissa toiminnallisuudeltaan vastaamaan näitä tilanteita [10]. Yllämainittujen asioiden johdosta ohjelmistojen sovittaminen muuttuviin liiketoiminnan sääntöihin on välttämätöntä ja väistämätöntä [12]. Tulevien sukupolvien tietojärjestelmien pitääkin olla riittävän mukautumiskykyisiä ja ottaa huomioon liiketoimintojen rakenteiden muutos salliakseen liiketoimintojen mukautuminen toimintaympäristöjen muuttuessa. [5]

Jotta ohjelmistot pystyisivät mukautumaan muutoksiin, on osa järjestelmän ominaisuuksista, kuten liiketoiminnan säännöt siirrettävä osaksi järjestelmän sisältämää tietoa. Tietojärjestelmän suunnasta katsottaessa liiketoiminnan sääntö on lause, joka määrittelee tai rajoittaa jotain liiketoiminnan osa-aluetta. Säännön tarkoituksena on kontrolloida tai vaikuttaa liiketoimintaan ja selittää sen rakennetta. Säännöt voidaan tallentaa metatietona järjestelmän ulkopuolelle esimerkiksi tietokantaan tai xml-tiedostoihin koodin sijasta. Metatiedon lisääntyessä voidaan päästä tilanteeseen, jossa järjestelmälle

oleellinen oliomalli on osa järjestelmän sisältämää tietoa ja koodin sisältämä oliomalli on vain tämän oliomallin tulkitsija. [7], [12]

Liiketoiminnan sääntöjen siirtäminen pois koodista osaksi järjestelmien sisältämää tietoa nostaa myös esiin tarpeen kehittää välineitä ylläpitäjille ja päätöksentekijöille, joiden avulla he voivat muuttaa liiketoiminnan sääntöjä ja mallia ajonaikana. Käyttäjät voivat täten halutessaan muuttaa liiketoiminnan sääntöjä koodaamatta. Tämä voi puolestaan säästää aikaa uusien ideoiden saamiseksi markkinoille kuukausista viikkoihin tai peräti päiviin. Järjestelmän muokkaus voidaan myös asettaa niiden ihmisten vastuulle, jotka omaavat tiedon itse liiketoiminnasta ja pystyvät suorittamaan sen tehokkaasti. [10], [7]

4.3 Ohjelmiston dynaamisuuden kehittyminen

Asiakkaiden asettamat tarpeet sovelluksien toiminnan muuttumiselle heidän muuttuvien liiketoiminnan käytäntöjen ja sääntöjen tahdissa on aiheuttanut sovellusten kehittymisen toiminnallisesti staattisista ohjelmista yhä konfiguroitavammiksi ja dynaamisiksi. Tämä muutosprosessi on vienyt oman aikansa ja se voidaan nähdä evoluutioprosessina yksittäisen tehtävän suorittavasta ohjelmasta kohti dynaamista oliomallia. [3]

Alun perin ohjelmat on usein tehty suorittamaan yksittäistä tehtävää. Myöhemmin ohjelman toiminnan ohjaamiseksi voidaan lisätä optioita ja parametreja, jotka vaikuttavat ohjelman johonkin toiminnallisuuteen. Parametrien ja optioiden käydessä riittämättömiksi konfigurointitiedon lisääntyessä voidaan nämä tallentaa erillisiin konfigurointitiedostoihin. Tiedostojen muodostuessa yhä monimutkaisimmiksi niitä voidaan kytkeä ohjelman sisältämiin olioihin käyttäen ominaisuuksia ja dynaamisia muuttujia. Yksittäiset arvot eivät kuitenkaan usein riitä kaikkiin tilanteisiin. Tämä johtaa jäsentimien ja lausekkeiden analysoijien mukaantuloon. Nämä tuovat esiin houkutuksen myös lisätä kontrollirakenteet, sijoitukset ja läpikäyntimahdollisuudet lausekeanalysoijien sanastoon. Tämä voi johtaa täysimittaisiin skriptausmahdollisuuksiin. Skriptausmahdollisuuksien lisääntyessä liiketoiminnan tai toimialueen oliot muodostavat eräänlaisen ohjelman, jota voidaan dynaamisesti rakentaa ja manipuloida käyttäjien toimesta. Tämän evoluutioprosessin aikana tieto-olioiden kuvauksia aletaan tallentaa metatietona. [3]

Järjestelmän arkkitehtuurin kypsyessä ja kehittyessä nousevat kytkökset olioiden välillä yhä enenemissä määrin tärkeäksi osaksi metatietoa ja yhä vähemmän asioita pysyväiskoodataan järjestelmän koodiin. Oliot tällaisessa dynaamisessa oliomallissa ovat ajonaikaisen konfiguroinnin ja muutoksen alaisia, kuten mikä tahansa muu järjestelmän sisältämä tieto. Muutokset dynaamisessa oliomallissa ajonaikana aiheuttavat muutoksen järjestelmän toiminnassa. [3]

5 Dynaaminen oliomalli

Tässä luvussa käsitellään kirjallisuudessa ja tiedeyhteisössä esitettyjä ratkaisuja dynaamisen oliomallin toteuttamiseksi. Suurin osa tiedosta on peräisin kansainvälisistä konferensseista, kuten OOPSLA (Object-Oriented Programming, Systems, Languages & Applications) ja sen eurooppalaisessa vastineessa ECOOP (European Conference on Object-Oriented Programming), joissa dynaamista oliomallia on käsitelty useaan otteeseen 90-luvun lopulta alkaen. Dynaamisen oliomallin muotoutumiseen ovat suuresti vaikuttaneet muun muassa Illinoisin yliopiston (University of Illinois at Urbana-Champaign) professorit Ralph Johnson ja Joseph Yoder, jotka ovat esittäneet useita julkaisuja asiaan liittyen.

5.1 Yleistä

Dynaaminen oliomalli on järjestelmä, joka esittää käyttäjän määrittelemät luokat, muuttujat, kytkennät ja toiminnan metatietona. Järjestelmä koostuu enemmänkin ilmentymistä kuin itse luokista. Käyttäjä muuttaa metatietoa ilmentääkseen muutoksia toimialassa. Nämä muutokset muuttavat järjestelmän toimintaa. Toisin sanoen dynaaminen oliomalli tallentaa oliomallinsa ulkoiseen tietovarastoon ja tulkitsee sitä. Tämän seurauksena oliomalli on mukautuva ja kuvausten muutokset aiheuttavat välittömän muutoksen järjestelmässä. [13]

Dynaamisen oliomallin rakenne eroaa yleisimmistä olio-orientoituneista malleista. Yleisesti olio-orientoituneissa malleissa luokat kuvaavat erilaisia liiketoiminnan olioita muuttujineen ja kytkentöineen. Dynaaminen oliomalli ei kuvaa näitä liiketoiminnan olioita luokkina. Sen sijaan ne kuvataan metatietona, joka tulkitaan ajonaikana. Tämän johdosta muutokset liiketoiminnassa muuttavat kuvauksia, jotka aiheuttavat sovelluksen välittömän toiminnan muutoksen. Toisin sanoen se mitä tavallisesti kuvattaisiin luokkina, kuvataan metatietona, joka sitten tulkitaan dynaamisessa oliomallissa [13]. Yleisesti tällaisissa järjestelmissä oliomalli on helposti muutettavissa tarkoituksenmukaisen käyttöliittymän kautta. [1]

Tässä diplomityössä yllä kuvatun mallin mukaisesta toteutuksesta käytetään nimitystä dynaaminen oliomalli (engl. Dynamic Object Model). Tämä nimitys ei tosin ole kovinkaan vakiintunut, vaan kyseisellä mallilla on monia muitakin nimityksiä, jotka ilmenevät

kirjallisuudessa ja tieteellisissä julkaisuissa. Taulukkoon 5.1 on listattu eräitä nimityksiä, joita on käytetty kuvaamaan dynaamista oliomallia.

Taulukko 5.1 Dynaamisen oliomallin nimityksiä

Nimitys	**Vapaa suomennos**
Dynamic Object Model [1]	Dynaaminen oliomalli
Object System	Oliojärjestelmä
Runtime Domain Model	Ajonaikainen kohdealueen malli
Adaptive Object Model [11]	Mukautuva oliomalli
Reflective Architecture	Kuvastava arkkitehtuuri
Meta-architecture	Meta-arkkitehtuuri
Type Instance Pattern	Tyyppiolio-ratkaisumalli
User Defined Product architecture [2]	Käyttäjän määrittelemä tuote arkkitehtuuri

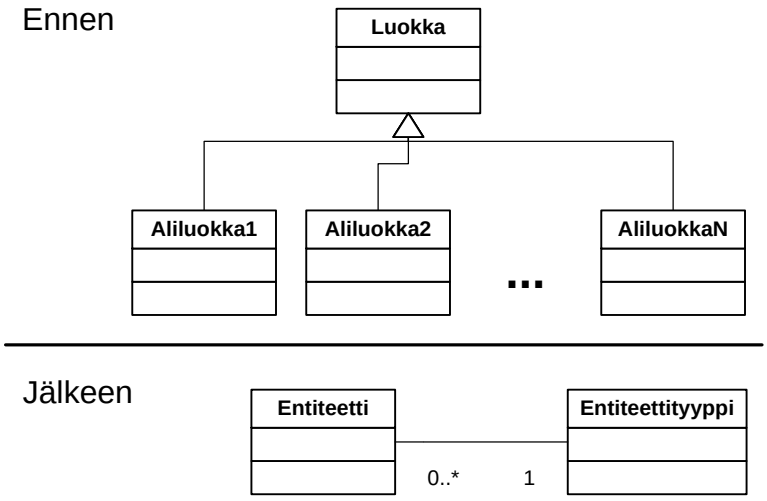
Osa nimityksistä viittaa käytettyihin ratkaisumalleihin kuten tyyppiolio-ratkaisumalliin (Type Instance Pattern) tai mahdollisiin dynaamisen oliomallin käyttökohteisiin kuten myytävien tuotteiden kuvaamiseen järjestelmässä (User Defined Product architecture). Tällä hetkellä käytetyimmäksi englanninkieliseksi nimeksi näyttää nousseen Adaptive Object Model, josta useat käyttävät lyhennettä AOM.

5.2 Dynaamisen oliomallin ratkaisumalleja

Yleisesti ottaen dynaamisen oliomallin toteuttaminen osaksi sovellusta onnistuu useita tunnettuja suunnittelun ratkaisumalleja (engl. design patterns) yhdistäen ja hyväksi käyttäen. Tärkein ratkaisumalli on tyyppiolio-ratkaisumalli (engl. Type Object Pattern), joka eriyttää itse olion sen tyypistä. Tyyppioliomalli mahdollistaa uusien liiketoimintaolioiden määrittelemisen järjestelmään. Olioilla on attribuutteja, jotka toteutetaan ominaisuus-ratkaisumallilla (engl. Property Pattern) ja tyyppiolio-ratkaisumallia käytetään uudelleen erottamaan attribuutit niiden omista tyypeistään. Strategia-ratkaisumallia (engl. Strategy Pattern) käytetään usein määrittelemään tietyn oliotyypin käyttäytymistä. Seuraavissa alakohdissa esitellään tarkemmin dynaamisen oliomallin suunnittelun ratkaisumalleja ja niiden vaikutusta sovelluksen dynaamisuuteen. [1], [13]

5.2.1 Tyyppiolio-ratkaisumalli

Useimmat oliokeskeiset kielet koostavat ohjelman joukkona luokkia, missä luokka kuvaa olion käyttäytymistä. Oliokeskeiset järjestelmät yleisesti käyttävät erillisiä luokkia eri tyyppisten olioiden kuvaamiseen, joten uudenlaisen oliotyypin julkaiseminen edellyttää ohjelmointia. Usein kuitenkin törmätään tilanteeseen, jossa luokasta pitäisi luoda lukematon määrä aliluokkia. Muutokset aliluokkien välillä ovat usein pieniä ja ne voidaan parametroida asettamalla arvoja tai olioita kuvaamaan algoritmeja. Tyyppiolio-ratkaisumalli (engl. Type Object Pattern) tekee tuntemattomista aliluokista yksinkertaisia geneerisen luokan ilmentymiä, jolloin uusia luokkia voidaan esitellä dynaamisesti ajonaikana yleisen luokan ilmentymin. Kuva 5.1 esittää yksinkertaista esimerkkiä, jossa annettu luokkahierarkia esitetään luokkana *Entiteettityyppi* (engl. Entity Type) ja sen ilmentymät luokkana *Entiteetti* (engl. Entity). Tällainen yksinkertaistus voidaan tehdä, jos luokkien välinen toiminta on yksinkertaista tai se voidaan eriyttää erillisiksi olioiksi ja täten pääasiallinen ero aliluokkien välillä on niiden sisältämät attribuutit. [7]

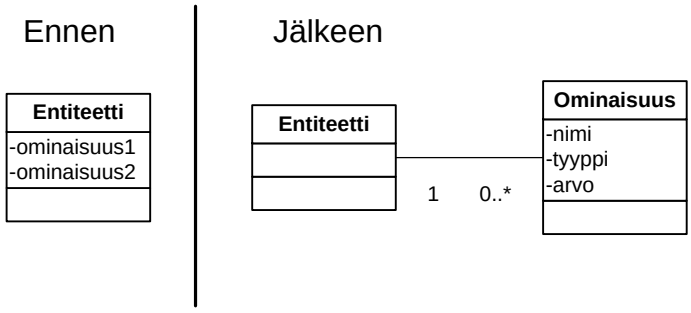


Kuva 5.1 Tyyppiolio-ratkaisumalli [7]

5.2.2 Ominaisuus-ratkaisumalli

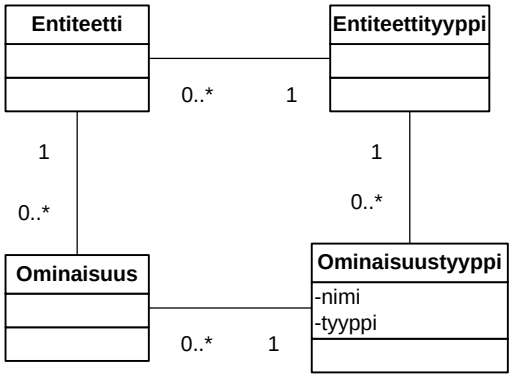
Normaalisti olion attribuutit toteutetaan sen instanssimuuttujina, jotka on määritelty luokalle. Jos oliot ovat saman luokan ilmentymiä, kuten tyyppiolio-ratkaisumallissa, niin tulee attribuutit toteuttaa eri tavalla kuin instanssimuuttujina, jotta oliot voisivat eriytyä

toisistaan. Tähän tuo ratkaisun ominaisuus-ratkaisumalli (engl. Property Pattern), jossa sen sijaan, että jokainen attribuutti on luokan instanssimuuttuja, voidaan attribuutit toteuttaa instanssimuuttujana, joka on kokoelma attribuutteja. Tämä kokoelma voi olla esimerkiksi vektori (engl. vector), hakemisto (engl. dictionary) tai hakutaulukko (engl. lookup table). Kuvan 4.2 esimerkissä *Entiteetti* –luokka sisältää kokoelman *Ominaisuus* –luokan ilmentymiä, jotka sisältävät attribuutin nimen, tyypin ja arvon.[7]



Kuva 5.2 Ominaisuus-ratkaisumalli [7]

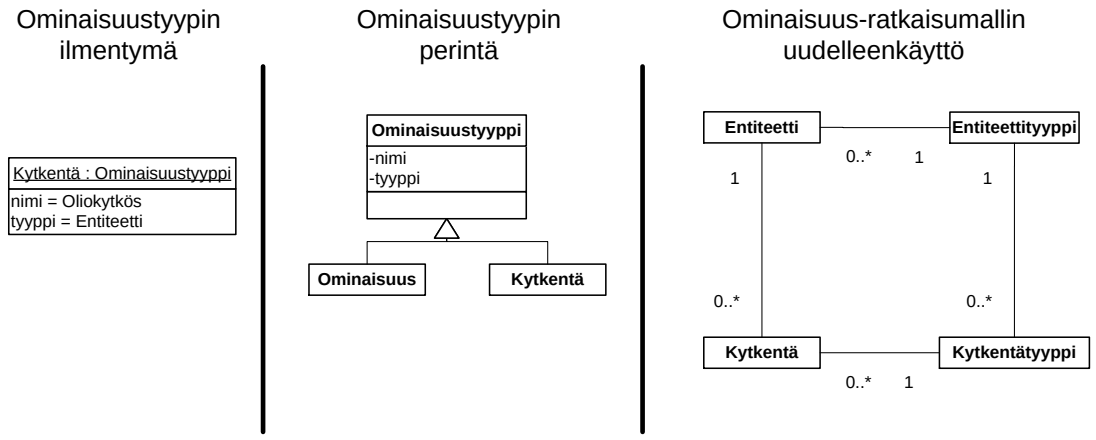
Kun yhdistetään tässä alakohdassa esitelty ominaisuus-ratkaisumalli ja edellisessä alakohdassa esitelty tyyppiolio-ratkaisumalli ja sovelletaan samalla tyyppiolio-ratkaisumallia erottamaan ominaisuus ominaisuuden tyypistä saadaan kuvan 5.3 esittämä arkkitehtuuri, jota eräät kutsuvat tyyppineliöksi (engl. Type Square). Tämän avulla pystytään helposti esittämään yksinkertaisia luokkia ja niiden ominaisuuksia ilman, että joudutaan luomaan ohjelmoimalla uusia luokkia. Tällä mallilla ei tosin vielä pystytä esittämään olioiden toimintaa ja sääntöjä. Näiden kuvaamiseen esitetään ratkaisu tulevissa alakohdissa. [1], [7]



Kuva 5.3 Tyyppineliö (engl. Type Square) [7]

5.2.3 Olioiden väliset kytkennät

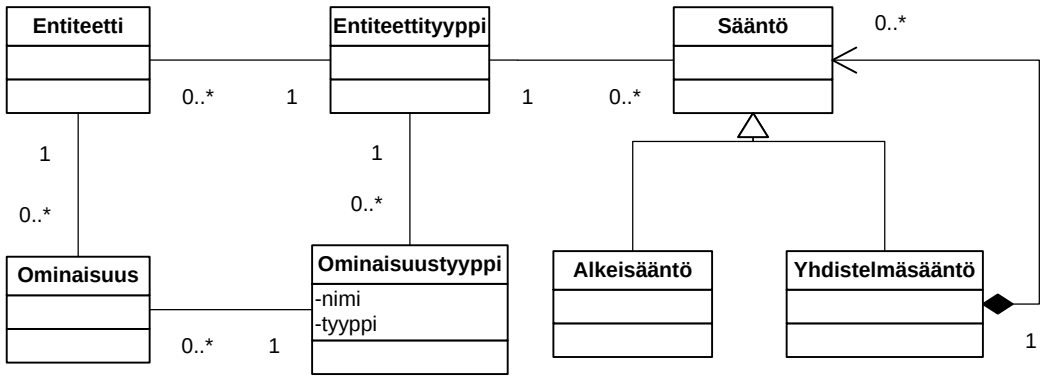
Olioiden väliset kytkennät voidaan toteuttaa joko jo yllä kuvatulla ominaisuus-ratkaisumallilla siten, että olioiden välinen kytkös on vain yhden tyyppinen ominaisuus oliolla muiden ominaisuuksien joukossa (ks. Kuva 5.4, ominaisuustyypin ilmentymä). Jos ominaisuudet ja kytkennät halutaan eriyttää toisistaan, voidaan ominaisuuden tyyppiolio periyttää, jolloin kytkentä ja ominaisuus ovat kaksi erillistä tyyppiolion aliluokkaa (ks. Kuva 5.4, ominaisuustyypin perintä). Toinen mahdollisuus on soveltaa ominaisuus-ratkaisumallia toisen kerran olioiden välisille kytkennöille, jolloin ne saavat oman rakenteensa arkkitehtuuriin (ks. Kuva 5.4, ominaisuus-ratkaisumallin uudelleenkäyttö) [1], [7]. Usein dynaamisen oliomallin järjestelmät kuvaavat olioiden väliset kytkennät omana rakenteenaan, sillä tämä mahdollistaa uuden tyyppisten kytkentöjen ja niihin liittyvien sääntöjen kuvaamisen metatietona erillään attribuuteista. [13], [9]



Kuva 5.4 Olioiden väliset kytkennät dynaamisessa oliomallissa

5.2.4 Strategia-ratkaisumalli ja sääntöoliot

Jotta dynaamiseen oliomalliin voitaisiin määritellä olioille sääntöjä ja toiminnallisuutta voidaan malliin soveltaa strategia-ratkaisumallia, jossa jokaiseen tyyppiolioon liitetään sääntöolio, joka on strategia-ratkaisumallin mukainen rajapinta algoritmille. Tästä sääntöjen yläluokasta voidaan tämän jälkeen periyttää tarvittava määrä aliluokkia, joiden avulla sääntöjä voidaan esittää sekä koostaa suuremmaksi joukoksi sääntöjä käyttäen yhdistelmä-ratkaisumallia (engl. Composite Pattern). Kuvassa 5.5 on esitetty luokkakaavio dynaamisesta oliomallista, johon on liitetty strategia-ratkaisumallin mukaiset sääntöoliot. [1], [7]



Kuva 5.5 Dynaaminen oliomalli yhdistettynä sääntöolioihin [7]

5.2.5 Dynaamisen oliomallin ratkaisumallien yhteenveto

Yllä esitettyjen dynaamisen oliomallin ratkaisumallien avulla pystytään toteuttamaan metatiedolla kuvattavissa oleva järjestelmä, joka kuvaa liiketoiminnan rakenteen ja rajoitteet sekä pystyy mukautumaan liiketoiminnan muutoksiin metatiedon muutoksen avulla. Ratkaisumallit itse asiassa abstrahoivat normaalin olio-mallin esittämällä luokat tyyppeinä ja niiden sisältämät ominaisuudet ominaisuustyyppeinä sekä olioiden sisältämät liiketoimintasäännöt sääntöolioina. Ratkaisumallit yksinkertaistavat liiketoimintaolioiden rakenteen ja mahdollistavat tätä kautta niiden kuvaamisen helposti metatiedon avulla.

5.3 Dynaamisen oliomallin esittämisen ratkaisumalleja

Yleisellä tasolla dynaaminen oliomalli esitetään usein kirjallisuudessa koostuvan kahdesta ohjelmallisesta kerroksesta, tietokerroksesta ja toiminnallisesta kerroksesta. Nämä kaksi kerrosta tarkastelevat dynaamisten olioiden kuvausta ja rakennetta. Näiden kerrosten lisäksi kuitenkin melkein aina tarvitaan dynaamisen oliomallin esittämisestä vastaava ohjelmallinen esityskerros, joka koostuu esittämiseen erikoistuneista komponenteista. Komponentteja dynaamisesti yhdistelemällä pystytään luomaan dynaamisesta oliomallista monimutkaisia näyttöjä. Esityskerroksen avulla pystytään koteloimaan ja abstrahoimaan esittämiseen liittyvät asiat. Taulukossa 5.2 on esitetty dynaamisen oliomallin ohjelmalliset kerrokset. [13]

Taulukko 5.2 Dynaamisen oliomallin ohjelmalliset kerrokset [13]

Kerros	**Tehtävä**
Tietokerros (engl. Knowledge level)	Määrittelee toiminnallisuuden ja rakenteen (entiteettityyppi, ominaisuustyyppi).
Toiminnallinen kerros (engl. Operational level)	Pitää sisällään sovellusalueen arvot eli luokkien ilmentymät ja ominaisuudet. (entiteetti, ominaisuus)
Esityskerros (engl. Visualization level)	Koostuu esittämiseen erikoistuneista komponenteista.

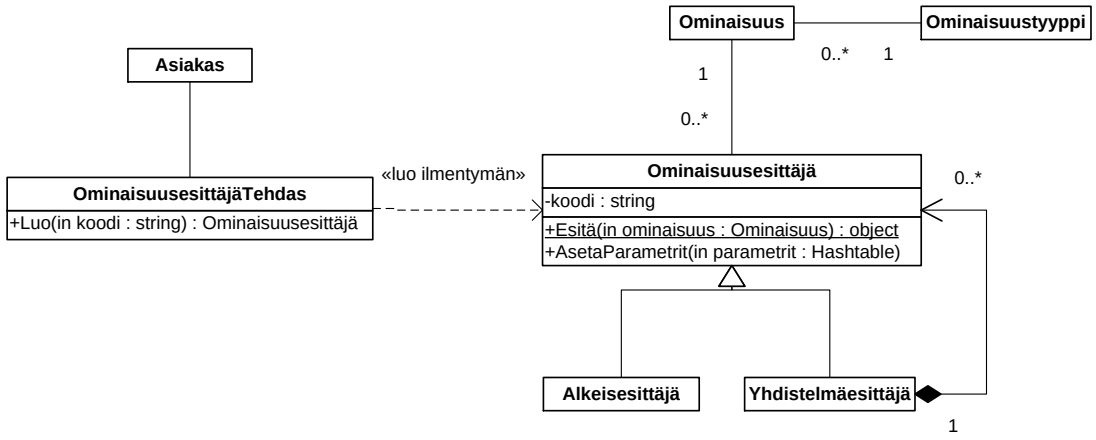
Seuraavissa alakohdissa käsitellään ratkaisumalleja, joita voidaan käyttää dynaamisen oliomallin esityskerroksessa.

5.3.1 Ominaisuusesittäjä-ratkaisumalli

Ominaisuusesittäjä-ratkaisumallin (engl. Property Renderer) perusideana on ratkaista se miten tietyt ominaisuudet esitetään ja kapseloida tämä toiminnallisuus. Tämä mahdollistetaan luomalla olio, jonka tehtävänä on esittää tietyn tyyppinen dynaamisen oliomallin ominaisuus tietyssä asiayhteydessä. Olio sisältää toiminnallisuuden, jonka avulla se pystyy tuottamaan käyttöliittymän tietylle *Ominaisuus* –oliolle. Ominaisuusesittäjä-ratkaisumalli auttaa eriyttämään itse dynaamisen oliomallin ja sen esityksen eristämällä näyttämiseen liittyvä koodi liiketoiminnanmallista itsestään. [13]

Kaikki ominaisuusesittäjät ovat pieniä tietyn ominaisuustyypin esittämiseen erikoistuneita luokkia. Ne käsittelevät tietyn ominaisuustyypin ilmentymän esittämistä ottamatta kantaa esityksen lopulliseen sijaintiin käyttöliittymässä. Ominaisuusesittäjät toteuttavat yhtenäisen rajapinnan, jonka avulla varsinaiseen käyttöliittymän sommitteluun erikoistuneet oliot pystyvät käsittelemään niitä. [13]

Kuvassa 5.6 on esitetty ominaisuusesittäjä-ratkaisumallin luokkakaavio. *Ominaisuusesittäjä* –luokka on kaikkien ominaisuuksia esittävien luokkien yläluokka ja määrittää myös niiden rajapinnan. *Ominaisuusesittäjä* –luokka sisältää metodit, joiden avulla sille voidaan välittää esitettävän *Ominaisuus* –luokan ilmentymä sekä mahdollisia esittämiseen liittyviä parametreja. [13]



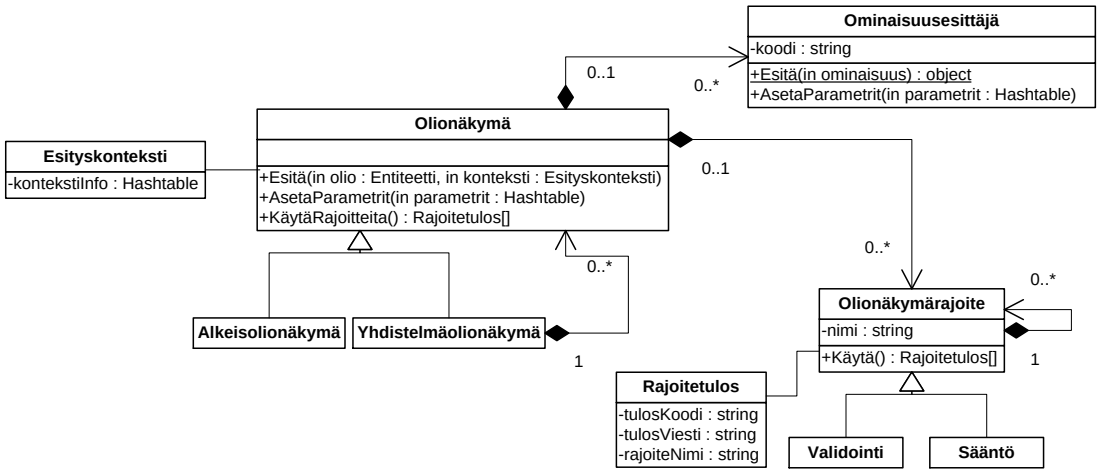
Kuva 5.6 Ominaisuusesittäjä-ratkaisumalli [13]

5.3.2 Olionäkymä-ratkaisumalli

Olionäkymä-ratkaisumallin (engl. Entity View) perusideana on koordinoida olion ominaisuuksien esittäminen monimutkaisempien käyttöliittymäosien muodostamiseksi. Tämä saavutetaan luomalla olio, joka koordinoi useita ominaisuusesittäjä-ratkaisumallin *Ominaisuusesittäjä* –olioita käyttöliittymän muodostamiseksi ja muodostaa täten käyttöliittymän yksittäiselle dynaamisen oliomallin *Entiteetti* –oliolle. [13]

Olionäkymä-ratkaisumallin esitysolio on tietoinen sitä ympäröivästä esityskontekstista, jolloin sen muodostama käyttöliittymä voi vaihdella sen mukaan mihin kyseinen käyttöliittymä on tarkoitus luoda. Esitysolio on myös tietoinen mahdollisista käyttöliittymään vaikuttavista rajoitteista, kuten syötettävien tietojen validoinneista ja säännöistä, jotka liittyvät oleellisesti käyttöliittymän muodostamiseen. [13]

Kuvassa 5.7 on esitetty olionäkymä-ratkaisumallin luokkakaavio. *Olionäkymä* –luokka on kaikkien olioita esittävien luokkien yläluokka ja määrittelee myös niiden rajapinnan. *Olionäkymä* –luokka sisältää metodit, joiden avulla sille voidaan välittää esitettävän *Entiteetti* –luokan ilmentymä ja ympäröivä esityskonteksti sekä mahdollisia esittämiseen liittyviä parametreja. [13]

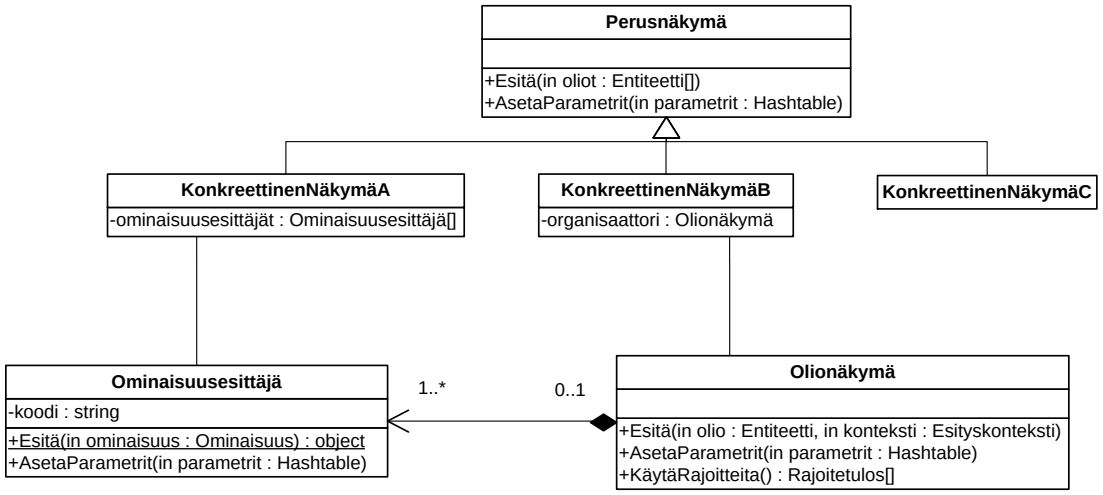


Kuva 5.7 Olionäkymä-ratkaisumalli [13]

5.3.3 Dynaaminen näkymä-ratkaisumalli

Dynaaminen näkymä-ratkaisumallin (engl. Dynamic View) perusideana on esittää joukko olioita käyttöliittymässä ja hallinnoida niiden sommittelu. Tämä saavutetaan luomalla olio, joka koordinoi useita olionäkymä-ratkaisumallin *Olionäkymä* –olioita käyttöliittymän muodostamiseksi ja muodostaa täten käyttöliittymän joukolle dynaamisen oliomallin *Entiteetti* –olioita. [13]

Kuvassa 5.8 on esitetty dynaaminen näkymä-ratkaisumallin luokkakaavio. *Perusnäkymä* –luokka on kaikkien oliojoukkoja esittävien luokkien yläluokka ja määrittelee myös niiden rajapinnan. *Perusnäkymä* –luokka sisältää metodit, joiden avulla sille voidaan välittää joukko esitettäviä *Entiteetti* –luokan ilmentymiä sekä mahdollisia esittämiseen liittyviä parametreja. Perusnäkymä-luokasta perityt konkreettiset esittäjäluokat voivat muodostaa itse käyttöliittymän usealla eri tavalla hyväksi käyttäen *Ominaisuusesittäjä* –luokkia, *Olionäkymä* –luokkia tai muodostamalla käyttöliittymän täysin itsenäisesti. [13]



Kuva 5.8 Dynaaminen näkymä-ratkaisumalli [13]

5.3.4 Esityskerroksen ratkaisumallien yhteenveto

Yllä esitettyjen dynaamisen oliomallin esittämiseen liittyvien ratkaisumallien avulla pystytään erottamaan käyttöliittymä itse dynaamisesta oliomallista. Tämän lisäksi pystytään luomaan käyttöliittymäolioita, jotka ymmärtävät dynaamisen oliomallin ja pystyvät täten mukautumaan dynaamisen oliomallin muutoksiin sekä ilmentämään tapahtuvia muutoksia suoraan käyttöliittymässä ilman käyttöliittymäkomponenttien koodin muuttamista.

5.4 Dynaamisen oliomallin toteutus

Vaikka dynaaminen oliomalli pystytään toteuttamaan ylläkuvatuilla melko yksinkertaisilla ratkaisumalleilla, liittyy itse dynaamisen oliomallin sisältämän sovelluksen toteutukseen useita erilaisia kysymyksiä ja haasteita. Yleisesti katsotaan, että dynaamisen oliomallin sisältämien järjestelmien toteuttaminen on hankalaa ja vaatii kokeneita kehittäjiä. Oliomallin abstrahointi ja metatieto tekevät usein järjestelmistä vaikeaselkoisia vähemmän kokeneille kehittäjille [1]. Seuraavissa alakohdissa on tarkoitus käydä läpi joitain dynaamisen oliomallin sisältämän järjestelmän toteutukseen liittyviä asioita ja katsahtaa myös olemassa oleviin toteutuksiin.

5.4.1 Olemassa olevia toteutuksia

Olemassa olevat dynaamisen oliomallin sisältämät järjestelmät toteuttavat usein jonkin tietyn piirteen järjestelmässä dynaamisen oliomallin avulla. Taulukossa 5.3 on lueteltu tiettyjä olemassa olevia toteutuksia ja niiden dynaamisen oliomallin käyttökohteita.

Taulukko 5.3 Dynaamisen oliomallin käyttökohteita olemassa olevissa toteutuksissa

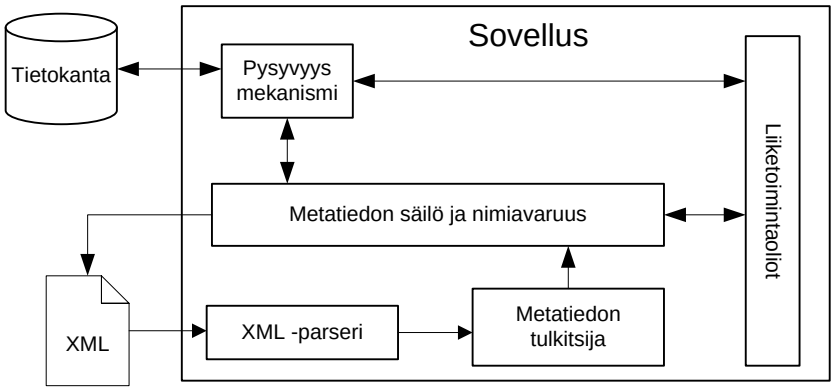
Toteutus	**Dynaamisen oliomallin käyttökohde**
Hartford User-Defined Product Framework	Kuvataan tuotteita järjestelmässä ja niihin liittyvää laskentaa. Käytetty esimerkiksi vakuutusten kuvaamiseen ja niiden hintojen laskemiseen.
Argo Document Workflow Framework	Kuvataan dokumenttien hallinta ja siihen liittyvä historiointi.
Objectiva Telephone Billing System	Kuvataan puhelinlaskun muodostus ja siihen liittyvä laskutus.
IDPH Medical Domain Framework	Kuvataan lääketieteellisiä havaintoja, niiden sisältämiä arvoja sekä kytköksiä toisiinsa.

Dynaamisen oliomallin perustoteutus on yllä olevassa taulukossa olevien toteutusten välillä melko samanlainen. Kaikki toteuttavat dynaamisen oliomallin ratkaisumallit, kuten tyyppi- ja ominaisuus-ratkaisumallin samalla tavalla. Suurimmat eroavaisuudet järjestelmien välillä syntyvät strategia-ratkaisumallin ja sääntöjen toteutuksessa. Sääntöjen kuvaaminen on järjestelmissä liiketoimintakeskeistä ja täten eri liiketoimintoalueille tuotetut sovellukset eriävät sääntöjen osalta toisistaan.

5.4.2 Toteutuksen haasteita

Dynaaminen oliomalli paljastaa järjestelmän oliomallin metatietona, joka voidaan tallentaa tietokantaan helposti tunnettuja menetelmiä käyttäen. Oliotietokannat ovat helpoin tapa hallinnoida olioiden pysyvyys. Muita mahdollisia tallennuspaikkoja ovat relaatiotietokannat ja XML- tiedostot (Extensible Markup Language). Riippumatta siitä kuinka itse malli on tallennettu pitää järjestelmän pystyä lukemaan ja koostamaan

dynaaminen oliomalli tietosäilöstä. Kuvassa 5.9 on esitetty dynaamisen oliomallin sisältämän sovelluksen eräs mahdollinen rakenne. [14]



Kuva 5.9 Dynaamisen oliomallin toteutuksen rakenne [14]

Olioiden pysyvyyden lisäksi on olemassa monia muita dynaamisen oliomallin suunnitteluun ja toteutukseen liittyviä haasteita. Dynaamiseen oliomalliin perustuva järjestelmä voi olla hidas, sillä se on itse asiassa tulkattava sovellussuuntautunut kieli (engl. domain specific language). Tämän uuden kielen määrittelyssä tulee ottaa huomioon uuden kielen vaatimat tukiohjelmat, kuten virheenjäljittimet (engl. debuggers), versionhallinta ja dokumentointivälineet. Mikäli käyttäjien itse annetaan muokata järjestelmää tulee heille myös tarjota tätä varten työkalu. Taulukossa 5.4 on lueteltu dynaamisen oliomallin suunnitteluun ja toteutuksen liittyviä haasteita. [11]

Taulukko 5.4 Dynaamisen oliomallin suunnittelun ja toteutuksen haasteita [11]

Haaste	**Selitys**
Käyttäjien oikeutus	Käyttäjät tulee jakaa ryhmiin, sillä peruskäyttäjien ei tule päästä muokkaamaan metatietoa.
Oliokonfiguraatio	Metatiedon editoimiseksi pitää tarjota työkalu.
Loppukäyttäjäohjelmointi	Sääntöjen luomiseksi tulee tarjota jonkinlainen skriptaus – kieli, joka tulkitaan ajonaikana.
Tehokkuus	Dynaamisen oliomallin ja sen sääntöjen tulkinta ajonaikana

	sekä metatiedon säilöminen muistissa voivat aiheuttaa tehokkuusongelmia.
Yhdenaikaisuus ja tiedon yhtenäisyys	Järjestelmän tulee hallita tilanne, jossa dynaaminen oliomalli muuttuu ajonaikana. Ongelmaksi nousevat muistissa ja ulkoisessa varastossa oleva metatieto ja tieto-oliot, jotka pitää päivittää hallitusti.
Versiointi	Metatiedon muokkaaminen nostaa esiin kysymyksen mahdollisesta metatiedon versioinnista.
Olioiden pysyvyys	Normaalista ohjelmistosta poiketen liiketoimintaolioiden lisäksi tulee tallentaa metatieto johonkin ulkoiseen tietovarastoon. Ongelmakohtina ovat tietovaraston rakenne ja tiedon siirtäminen dynaamisen oliomallin ja tietovaraston välillä.

5.4.3 Toteutuksen selkiyttäminen

Suurimmaksi syyksi dynaamisen oliomallin toteutusten vaikeaselkoisuuteen on esitetty epäyhtenäisyyttä dynaamisen oliomallin ja sitä kehitettävän ohjelmointikielen välillä. Tähän on yritetty löytää ratkaisua dynaamisen oliomallin siirtämiseksi osaksi itse ohjelmointikieltä metaluokkien avulla. Joitain kokeiluja on tehty Smalltalk-80 kielellä, mutta varsinaisia ratkaisuja muille ohjelmointikielille ei ole esitetty. [5]

Toiseksi syyksi dynaamisen oliomallin toteutuksen vaikeaselkoisuuteen on esitetty dynaamisen ja staattisen ohjelmakoodin yhteensovittamisen vaikeutta. Tähän on esitetty ratkaisuksi aspektiohjelmointia (engl. Aspect-Oriented Programming), jossa dynaaminen ja staattinen sovelluksen osa erotetaan toisistaan tiettyjen mukautumispisteiden (engl. Adaptability Points) avulla. [6]

5.5 Dynaamisen oliomallin yhteenveto

Dynaaminen oliomalli tarjoaa vaihtoehdon perinteiselle olio-orientoituneelle suunnittelulle. Perinteinen olio-orientoitunut suunnittelu luo erilaisia luokkia eri tyyppisille liiketoiminnan olioille ja liittää niihin attribuutteja ja metodeita. Nämä muodostavat järjestelmän, joka vaatii koodin muuttamista ja uuden version luomista järjestelmästä, mikäli liiketoiminnan säännöt muuttuvat. Dynaaminen oliomalli ei mallinna näitä liiketoiminnan olioita varsinaisina luokkina. Dynaamisessa oliomallissa liiketoiminnan oliot kuvataan metatietona, joka kuvaa liiketoiminnan rakenteen ja rajoitukset. Tämä metatieto tulkitaan ajonaikana ja se määrää järjestelmän toiminnan. Dynaamisessa oliomallissa liiketoiminnan muutokset aiheuttavat muutoksen metatiedossa, jota voidaan muuttaa ajonaikana ja täten muutokset ilmenevät välittömästi järjestelmän toiminnassa. Dynaamisen oliomallin tärkeimpiä ratkaisumalleja ovat tyyppiolio-, ominaisuus- ja strategia-ratkaisumallit, jotka esiteltiin kohdassa 5.2. Nämä ratkaisumallin yksinkertaistavat liiketoimintaolioiden rakenteen ja mahdollistavat tätä kautta rakenteen kuvaamisen metatietona.

Dynaaminen oliomalli asettaa myös tiettyjä haasteita sitä tukevien järjestelmien esityskerrokseen. Näihin haasteisiin pystytään vastaamaan pilkkomalla käyttöliittymän muodostaminen pienempiin ohjelmakokonaisuuksiin esityskerroksen ratkaisumalleja hyödyntäen, jotka esiteltiin kohdassa 5.3.

Kohdassa 5.4 esiteltiin dynaamisen oliomallin toteutukseen liittyviä muita haasteita. Nämä ongelmakohdat on usein ratkaistu olemassa olevissa järjestelmissä omilla tavoillaan ja niihin ei varsinaisesti löydy valmiita ratkaisumalleja.

Osa B: Toteutus

6 Käytetty teknologia

Tässä luvussa käsitellään yleistä teknologiaa, jota hyödyntäen diplomityön tuotoksena syntynyt SafranOne – ohjelmistoalusta on toteutettu. Tämän luvun tarkoituksena on antaa yleiskuva käytetyistä tuotteista ja arkkitehtuurista SafranOne – ohjelmistoalustan ympärillä.

6.1 Microsoft .NET –arkkitehtuuri

Microsoftin .NET –arkkitehtuuri on ohjelmistokomponentti, joka on asennettavissa Microsoft Windows käyttöjärjestelmille. Arkkitehtuuri sisältää suuren joukon valmiiksi koodattuja luokkakirjastoja yleisiin ohjelmointivaatimuksiin, kuten käyttöliittymien luontiin, tietokannan käsittelyyn ja verkkosovellusten tekemiseen. .NET –arkkitehtuuri perustuu virtuaalikonemalliin, joka piilottaa laitteiston, jonka päällä ohjelmia ajetaan ja huolehtii myös esimerkiksi muistinhallinnasta. Varsinainen ohjelmointikieli käännetään välikielen tavukoodiksi (engl. Byte-code), joka on nimeltään Common Intermediate Language. Itse tavukoodi käännetään konekielelle vasta juuri ennen sen suoritusta (engl. JIT – Just in time).

6.1.1 C#

C# on Microsoftin kehittämä olio-orientoitunut ohjelmointikieli, joka on syntynyt osana Microsoftin .NET –arkkitehtuuria ja se on sekä ECMA:n (European Computer Manufacturers Association) ja ISO:n (International Organization for Standardization) standardoima. C# on proseduraalisuudeltaan ja olio-orientoituneelta syntaksiltaan hyvin läheistä sukua C++ -kielelle. Se on myös saanut paljon vaikutteita muista kielistä kuten JAVA. Yleisin C#:n toteutus on Microsoft Visual C#, joka on sisällytettynä osaksi Microsoftin tuotteita.

6.1.2 Windows Forms

Windows Forms on .NET –arkkitehtuurin sisältämä luokkakirjasto Windows –sovellusten tekemiseen. Se sisältää valmiita käyttöliittymäkomponentteja ikkunointiin, painikkeisiin ja

muihin näyttöelementteihin liittyen. Windows Forms –sovellukset voidaan levittää ja päivittää verkkosivuston kautta käyttäen niiden tukemaa Clickonce SmartClient –asennuspakettia, jolloin ohjelman pystyy avaamaan ja asentamaan WWW -selaimen kautta edellyttäen, että kohdetietokoneessa on asennettuna .NET –arkkitehtuurin ajoympäristö.

6.1.3 ASP.NET

ASP.NET on Microsoft:in verkkosovelluskehys, jonka avulla voidaan luoda verkkosivuja, verkkosovelluksia ja XML -pohjaisia verkkopalveluja (engl. web services). Se on osa Microsoftin .NET -arkkitehtuuria, minkä johdosta kaikki itse .NET -arkkitehtuurin ohjelmointikielet ja ominaisuudet ovat ohjelmoijien käytettävissä.

6.1.4 Web Parts

ASP.NET Web Parts on koottu joukko komponentteja, joiden avulla luodaan loppukäyttäjälle mahdollisuus muuttaa WWW -sivujen sisältöä, sommittelua ja toimintaa suoraan WWW -selaimessa. Muutokset voidaan asettaa koskemaan joko sivuston yksittäisiä tai kaikkia käyttäjiä. Kun käyttäjät muokkaavat sivuja ja niiden sisältämiä kontrolleja, niiden asetukset voidaan tallentaa tulevia selainistuntoja varten. Tätä toiminnallisuutta kutsutaan personoinniksi (engl. personalization). Nämä Web Part:ien ominaisuudet luovat kehittäjille mahdollisuuden antaa loppukäyttäjille valtuudet personoida nettisovelluksia dynaamisesti ilman kehittäjien tai ylläpitäjien väliintuloa. Taulukkoon 6.1 on koottu tärkeimpiä Web Parts –teknologian mahdollistamia asioita.

Taulukko 6.1 Web Parts –teknologian mahdollistamia asioita

Web Part -teknologian tuomia mahdollisuuksia
Sivujen sisällön personointi
Sivujen sommittelun personointi
Kontrollien asetuksien vienti ja tuonti ulkoisen kuvauksen kautta
Mahdollisuus luoda yhteyksiä kontrollien välille
Kontrollien valtuutusten personalisointi ja hallinta

6.1.5 ASP.NET User Control

ASP.NET käyttäjäkontrollit ovat hyvin samanlaisia, kuin ASP.NET –nettisivut. Molemmat koostuvat sekä käyttöliittymäsivusta ja koodista. Käyttäjäkontrolliin voi nettisivun tapaan lisätä sisäisiä kontrolleja ja merkintöjä (engl. markup). Koodilla voidaan tämän lisäksi hallita sisältöä ja suorittaa erinäisiä tehtäviä. ASP.NET –käyttäjäkontrolli ei toimi itsenäisesti, vaan se on aina lisättävä osaksi ASP.NET –nettisivua. ASP.NET käyttäjäkontrollin avulla ASP.NET -nettisivu pystytään jakamaan pienempiin osakokonaisuuksiin.

6.2 Microsoft Office SharePoint Server 2007

Microsoft Office SharePoint Server 2007 on .NET -arkkitehtuuria hyödyntävä porttaalisovellus, joka mahdollistaa intranet-, ekstranet- ja Web-sovellusten tuottamisen. Se sisältää ominaisuuksia sisällönhallintaan ja tiedon etsimiseen liittyen. SharePoint:in avulla voidaan luoda laajoja sivustoja ja mahdollistaa niiden muokkaaminen ja personointi käyttäjien toimesta. SharePoint:in sivut voivat sisältää ASP.NET Web Parts –komponentteja, joita hyödyntäen sivustoille on mahdollista lisätä SharePoint:in ulkopuolisia ohjelmistokomponentteja.

6.2.1 SmartPart

SmartPart on Web Part, joka pystyy esittämään ASP.NET käyttäjäkontrolleja SharePoint sivustolla. Tämä mahdollistaa käyttöliittymäpalojen graafisen suunnittelun, sillä ASP.NET käyttäjäkontrolleja pystytään suunnittelemaan graafisella välineellä Visual Studio 2005 kehittimessä. Itse WebPart:ien kehitykseen ei ole tarjolla graafista kehitintä.

6.3 Nolics.net

Nolics.net on sovellussuuntautuneella kielellä (engl. Domain specific language) varustettu olio- ja relaatiomallin yhteensovittamiseen (engl. Object-relational mapping) tarkoitettu tuote. Nolics.net tuotteen sovellussuuntautuneella kielellä kuvataan asiat, jotka liittyvät oleellisesti olioiden sarjallistamiseen (engl. serialization) relaatiotietokantaan. Sovellussuuntautuneella kielellä luodun mallin avulla Nolics.net luo mallia vastaavan

tietokannan ja C# luokat. Alla on esimerkki Nolics.net tuotteen sovellussuuntautuneesta kielestä, jossa on kuvattu luokka asiakas ja siihen liittyvä osoitetieto.

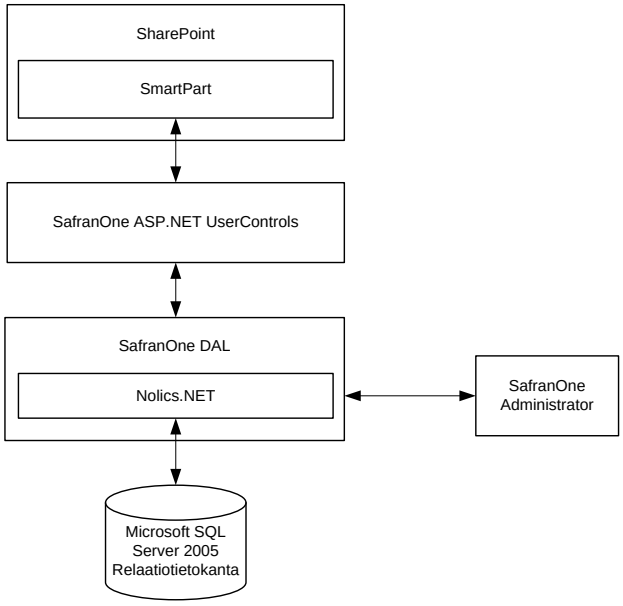
```
dbclass  Asiakas {
  primary key long Koodi[AutoGenID = true];
  string Nimi[100];
}

dbclass Osoite {
    primary key long
      ID[AutoGenID = true];
    string Katuosoite[50];
    string Postinumero[50];
    string Kaupunki[50];

    link Asiakas Asiakas_linkki:
        long AsiakasKoodi;
}
```

7 SafranOne

Tässä luvussa esitellään diplomityön osana toteutetun SafranOne –ohjelmistoalustan dynaamista oliomallia. Aluksi paneudutaan ohjelmistoalustan yleiseen rakenteeseen ja ratkaisujen toteutukseen SafranOne –ohjelmistoalustalla. Tätä seuraavissa kohdissa käydään dynaamista oliomallia läpi asiakkaan kuvitteellisen esimerkkivaatimuksen tukemana ja samalla käsitellään keskeiset dynaamisen oliomallin piirteet, jotka liittyvät kulloinkin käsiteltävään asiaan.

7.1 Yleistä

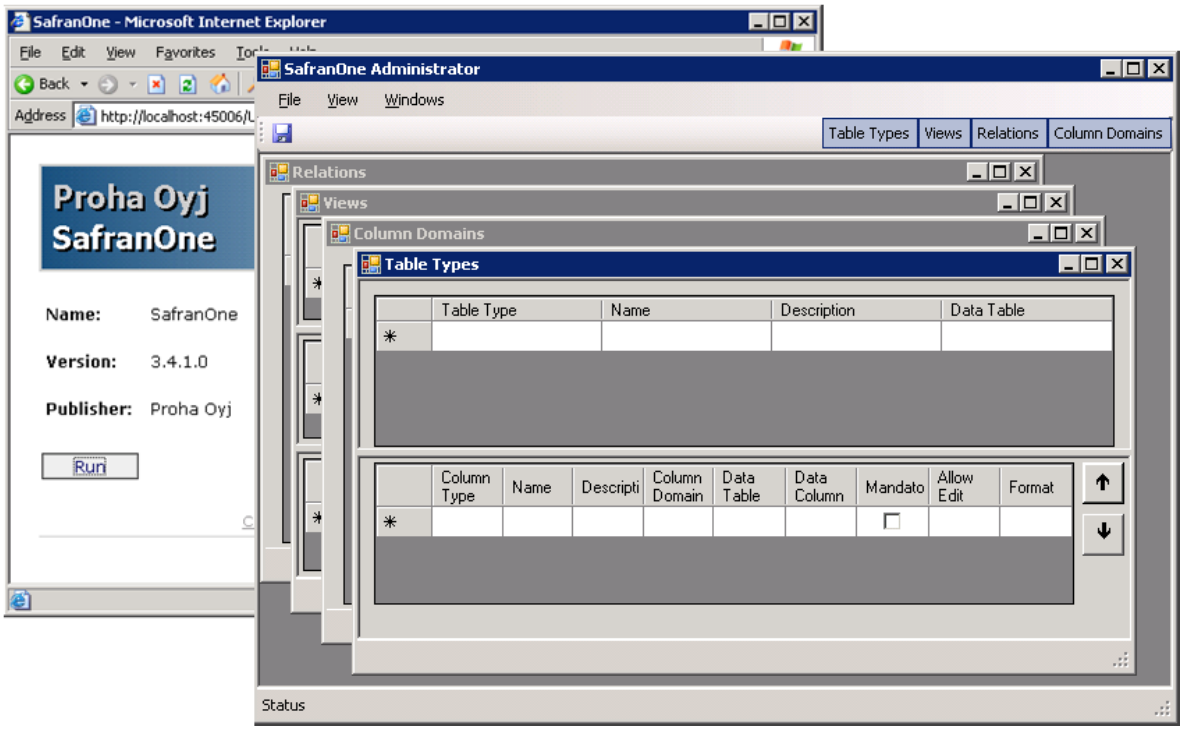
Diplomityössä toteutettu SafranOne –ohjelmistoalusta voidaan jakaa osiin kuvan 7.1 osoittamalla tavalla. SafranOne DAL (data access layer) pitää sisällään itse dynaamisen oliomallin ja tieto-oliot sekä hallinnoi näiden pysyvyyden tietokantaan Nolics.NET –tuotteen avulla. SafranOne DAL toimii palveluna SafranOne –ohjelmistoalustan käyttäjäkontrolleille. Nämä käyttäjäkontrollit esitetään Microsoft SharePoint –porttaalisovelluksessa SmartPart WebPart:n avulla. Itse dynaamista oliomallia muokataan tähän tarkoitukseen toteutetulla SafranOne Administrator –ohjelmalla.



Kuva 7.1 SafranOne –ohjelmistoalustan rakenne

7.1.1 SafranOne Administrator

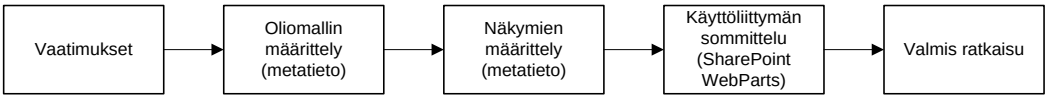
SafranOne Administrator –ohjelman avulla pystytään muokkaamaan dynaamisen oliomallin metatietoa. Sen avulla on mahdollista luoda uusia liiketoimintaolioiden kuvauksia ja linkittämään liiketoimintaolioita toisiinsa. SafranOne Administrator on toteutettu .NET SmartClient ClickOnce –teknologialla, jolloin se pystytään asentamaan ja ajamaan WWW –sivustolta suoraan ilman erillistä asennusta (ks. Kuva 7.2).



Kuva 7.2 SafranOne Administrator –ohjelmisto käynnistyy suoraan WWW –sivulta

7.1.2 Prosessi ratkaisujen toteutukselle

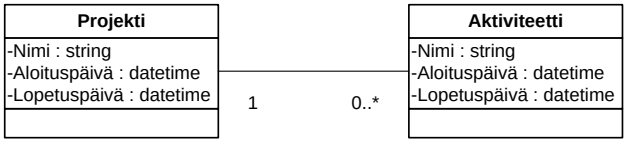
SafranOne –ohjelmistoalustalla muodostettavien ratkaisujen toteutus voidaan ajatella seuraavan kuvan 7.3 esittämän kaavion mukaista prosessia. Vaatimusten perusteella määritellään dynaamisen oliomallin metatieto käyttäen SafranOne Administrator –sovellusta. Tämän jälkeen määritellään metatietona mahdolliset näkymät määriteltyjä liiketoimintaolioita vasten, joita taas hyödyntävät käyttöliittymäkomponentit, jotka sommitellaan SharePoint:ssa halutulla tavalla. Tämän prosessin tuotoksena on valmis ratkaisu, joka sisältää halutut liiketoimintaoliot ja käyttöliittymän niiden käsittelyyn.



Kuva 7.3 SafranOne ratkaisun toteutusprosessi

7.1.3 Esimerkkivaatimus

Selvyyden vuoksi seuraavissa alakohdissa käytetään hyväksi kuvitteellista asiakkaan vaatimusta, joka on esitetty kuvassa 7.4. Asiakkaan tarpeena on saada järjestelmä, johon syötetään projekteja ja niihin liittyviä aktiviteetteja. Projektien aloitus- ja lopetuspäivät ovat laskennallisia kenttiä, jotka tulee laskea aktiviteettien ääripäivämääristä. Tätä esimerkkivaatimusta tullaan käyttämään tämän luvun muissa kohdissa havainnollistamaan SafranOne –dynaamista oliomallia.

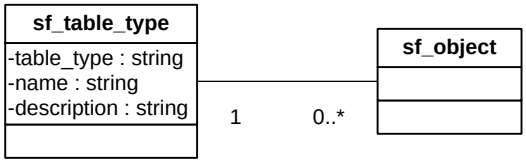


Kuva 7.4 Asiakkaan esimerkkivaatimus

7.2 Tyyppiolio

SafranOne:n dynaamisessa oliomallissa liiketoiminnan luokat on toteutettu alakohdassa 5.2.1 kuvatun tyyppiolio-ratkaisumallin mukaisesti. Liiketoiminnan luokat on kuvattu *sf_table_type* –luokan ilmentyminä (entiteettityyppi), mikä on osa järjestelmän sisältämää

metatietoa. Itse liiketoimintaoliot ovat *sf_object* –luokan (entiteetti) ilmentymiä. Kuvassa 7.5 on esitetty tämä SafranOne –ohjelmistoalustan sisältämä tyyppiolio-ratkaisumalli, jonka avulla uusia liiketoiminnan luokkia voidaan luoda järjestelmään luomalla uusia *sf_table_type* –luokan ilmentymiä.



Kuva 7.5 SafranOne tyyppiolio-ratkaisumalli

Viitaten alikohdan 7.1.3 vaatimukseen, on mahdollista luoda SafranOne Administrator –ohjelmalla kaksi uutta *sf_table_type* –luokan ilmentymää kuvaamaan projektia ja aktiviteettia. Näiden liiketoimintaluokkien määritykset on esitetty kuvassa 7.6.

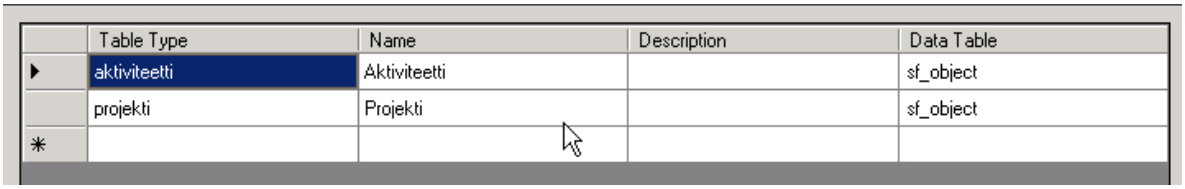


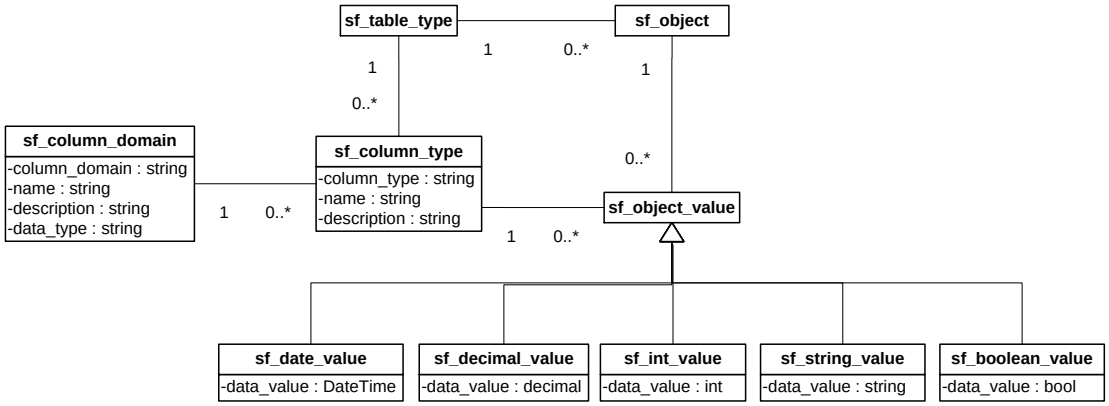
	Table Type	Name	Description	Data Table
▶	aktiviteetti	Aktiviteetti		sf_object
	projekti	Projekti		sf_object
*				

Kuva 7.6 sf_table_type –olioiden luonti SafranOne Administrator –ohjelmalla

7.3 Ominaisuustyyppi

SafranOne:n dynaamisessa oliomallissa liiketoiminnan luokkien ominaisuudet on toteutettu alakohdassa 5.2.2 esitellyn ominaisuusratkaisumallin mukaisesti soveltamalla ominaisuus-ratkaisumallia ja tyyppiolio-ratkaisumallia. Kuvassa 7.7 on esitelty SafranOne:n ominaisuustyyppi-ratkaisumallin toteutus. Liiketoiminnan luokkien *sf_table_type* ominaisuudet on kuvattu luokan *sf_column_type* ilmentyminä ja itse liiketoimintaolioiden *sf_object* ominaisuuksien arvot ovat luokan *sf_object_value* ilmentymiä. Eri tietotyyppiä olevien ominaisuuksien toteuttamiseksi *sf_object_value* –luokasta on peritty eri tietotyyppejä varten omat luokkansa *sf_date_value*, *sf_decimal_value*, *sf_int_value*, *sf_string_value* ja *sf_boolean_value*, jotka sisältävät itse ominaisuuden arvon. Tämän lisäksi *sf_column_type* ominaisuuden tietotyyppi kuvataan erillisessä *sf_column_domain* –

luokan ilmentymässä, jonka avulla eri tietotyypin ilmentymiä voidaan koostaa joukoiksi nimettyjä tyypityksiä, joille voidaan kenties tulevaisuudessa määritellä joitain erityisiä ehtoja. Tällainen ehto voisi olla esimerkiksi ikä, joka on tyyppiä kokonaisluku ja sen arvo voi olla välillä 0-150 ja täten se eroaa muista kokonaislukutietotyypeistä ehtonsa johdosta.



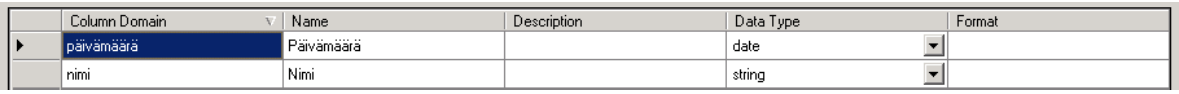
Kuva 7.7 SafranOne:n ominaisuustyyppi-ratkaisumalli

Viitaten alikohdan 7.1.3 esimerkkivaatimukseen, on mahdollista luoda SafranOne Administrator –ohjelmalla *sf_column_type* –luokan ilmentymiä kuvaamaan projekti ja aktiviteetti –luokan ominaisuuksia (Kuva 7.8). Ennen itse ominaisuuksien luontia on pitänyt kuitenkin luoda kaksi *sf_column_domain* –luokan ilmentymää kuvaamaan teksti- ja päivämäärätyyppistä tietoa kuvan 7.9 mukaisesti.

	Column Type	Name	Description	Column Domain	Data Table	Data Column	Mandatory	Allow Edit	Format
▶	aktiviteetti__nimi	Nimi		nimi	sf_string_value	data_value	☑	yes	
	aktiviteetti__aloituspäivä	Aloituspäivä		päivämäärä	sf_date_value	data_value	☑	yes	
	aktiviteetti__lopetupäivä	Lopetuspäivä		päivämäärä	sf_date_value	data_value	☑	yes	
*							☐		

	Column Type	Name	Description	Column Domain	Data Table	Data Column	Mandatory	Allow Edit	Format
▶	projekti__nimi	Nimi		nimi	sf_string_value	data_value	☑	yes	
	projekti__aloituspäivä	Aloituspäivä		päivämäärä	sf_date_value	data_value	☐	no	
	projekti__lopetuspäivä	Lopetuspäivä		päivämäärä	sf_date_value	data_value	☐	no	
*							☐		

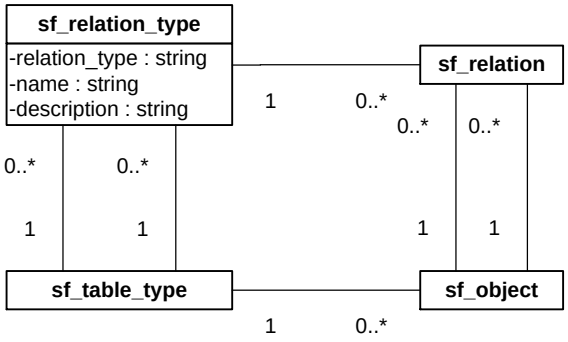
Kuva 7.8 sf_column_type –olioiden luonti SafranOne Administrator –ohjelmalla projekti- ja aktiviteetti –liiketoimintaluokille



Kuva 7.9 sf_column_domain –olioiden luonti SafranOne Administrator –ohjelmalla

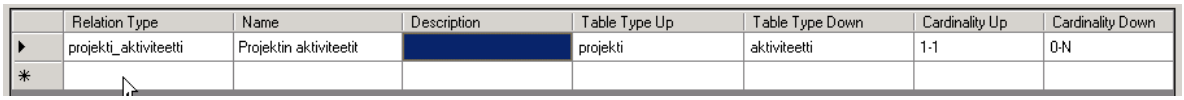
7.4 Olioiden väliset kytkennät

SafranOne:n dynaamisessa oliomallissa liiketoimintaluokkien väliset kytkennät on toteutettu alakohdassa 5.2.3 esitellyn ominaisuus-ratkaisumallin uudelleenkäytöllä. Kuvassa 7.10 on esitelty SafranOne:n olioiden välisten kytkentöjen ratkaisu ominaisuus-ratkaisumallin uudelleenkäytöllä. Liiketoimintaluokkien *sf_table_type* väliset kytkennät on kuvattu *sf_relation_type* –luokan ilmentyminä ja itse liiketoimintaolioiden *sf_object* väliset kytkennät on kuvattu *sf_relation* –luokan ilmentyminä. *Sf_relation_type* –luokan ilmentymä on täten dynaamisen oliomallin metatietoa ja kuvaa kahden liiketoimintaluokan välisen kytkennän tyypin. Kahden *sf_object* –liiketoimintaolion välinen kytkentä kuvataan *sf_relation* –luokan ilmentymänä.



Kuva 7.10 SafranOne:n olioiden väliset kytkennät

Viitaten alikohdan 7.1.3 esimerkkivaatimukseen, on mahdollista luoda SafranOne Administrator –ohjelmalla *sf_relation_type* –luokan ilmentymä kuvaamaan projektin ja aktiviteetin välistä kytkentää kuvan 7.11 mukaisesti.

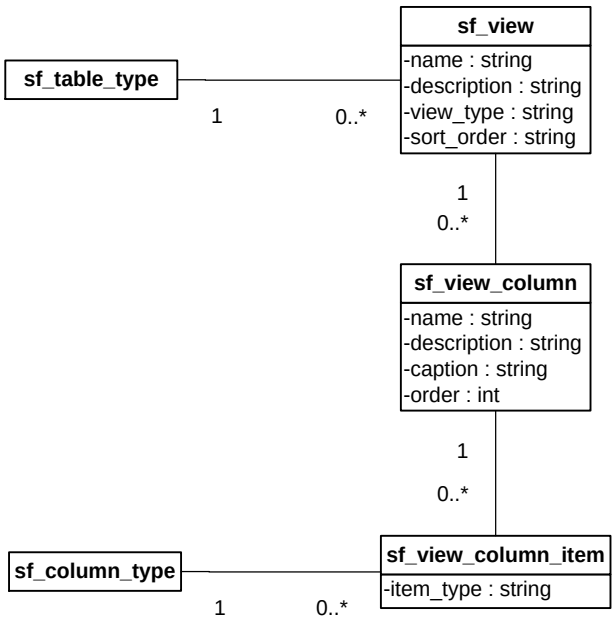


	Relation Type	Name	Description	Table Type Up	Table Type Down	Cardinality Up	Cardinality Down
▶	projekti_aktiviteetti	Projektin aktiviteetit		projekti	aktiviteetti	1-1	0-N
*							

Kuva 7.11 sf_relation_type –olion luonti SafranOne Administrator –ohjelmalla projektin ja aktiviteetin välille

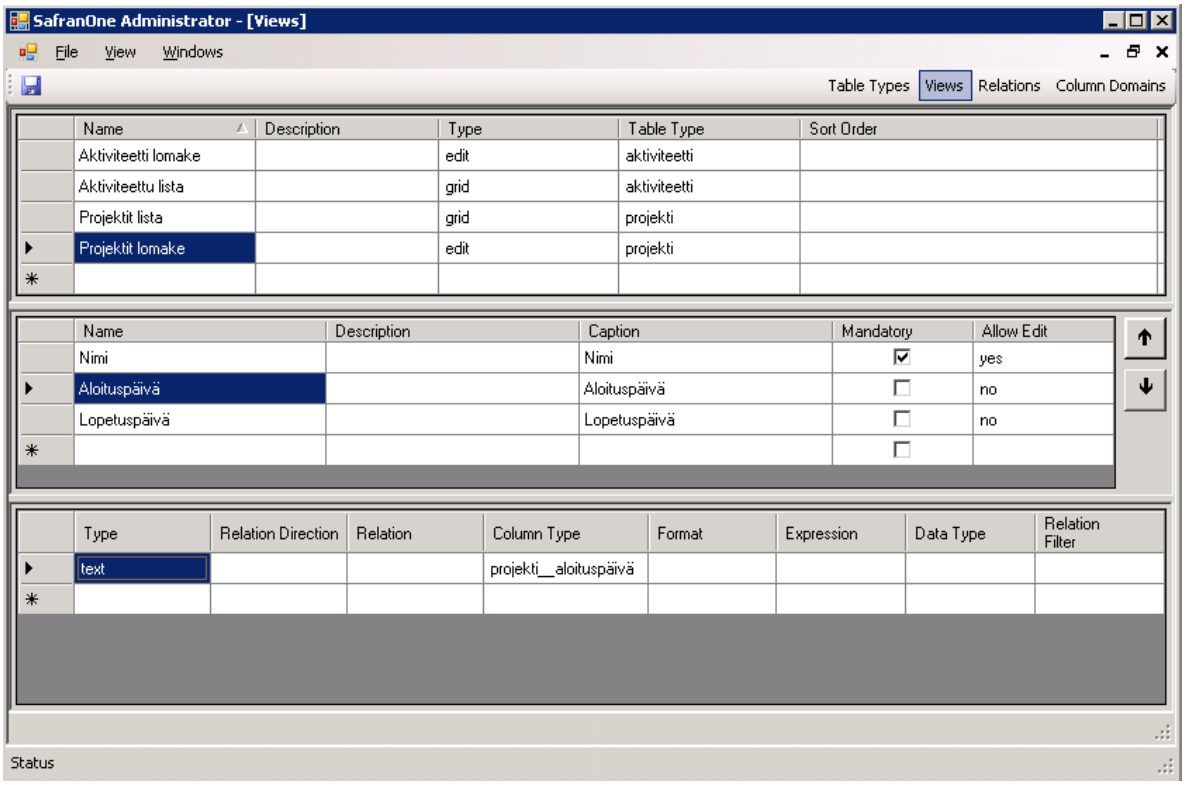
7.5 Näkymät

Näkymät kuvaavat liiketoimintaolion näyttämiseen liittyviä asioita, kuten mitä arvoja näytetään ja miten. SafranOne:n näkymät eivät ole suoranaisesti toteutettu kohdan 5.3 esittämien esityskerroksen ratkaisumallien avulla, vaan näistä ratkaisumalleista on lähinnä otettu ideoita SafranOne –ohjelmistoalustan näkymien ratkaisemiseksi. SafranOne:n näkymien rakenne on esitetty kuvassa 7.12. Yksittäinen näkymä liittyy aina yksittäiseen liiketoiminnan luokkaan *sf_table_type*, jota näkymä esittää. Näkymät ovat tyypiltään (*view_type*) joko taulukoita tai lomakkeita. Näkymiin sisältyy aina joukko sarakkeita, jotka on kuvattu luokan *sf_view_column* ilmentyminä ja näihin sarakkeisiin liittyviä ominaisuuksia, jotka on kuvattu luokan *sf_view_column_item* ilmentyminä. *Sf_view_column* kuvaa täten yksittäisen sarakkeen näyttämiseen liittyviä asioita kuten sarakkeen otsakkeen ja sen järjestyksen muiden otsakkeiden joukossa. Otsakkeisiin liittyvät *sf_view_column_item*:t kuvaavat, mitkä liiketoimintaolioiden arvot (*sf_column_type*) esitetään otsakkeen alla ja miten.



Kuva 7.12 SafranOne näkymien rakenne

Viitaten alikohdan 7.1.3 esimerkkivaatimukseen, on mahdollista luoda SafranOne Administrator –ohjelmalla näkymät projekti ja aktiviteetti –liiketoimintaolioille kuvan 7.13 mukaisesti.



Kuva 7.13 Näkymien luonti SafranOne Administrator –ohjelmalla

7.6 Ohjelmalogiikka ja säännöt

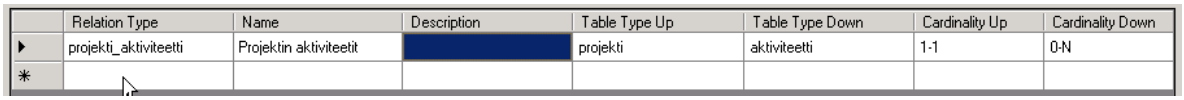
SafranOne –ohjelmistoalustaan ei sääntöjä varten toteutettu erillisiä sääntöolioita, jotka sisältäisivät liiketoimintalogiikan ja olisivat osa järjestelmän sisältämää metatietoa. Sen sijaan tietyt liiketoimintaolioihin liittyvät perussäännöt on asetettu itse liiketoimintamallia kuvaaviin metatieto-olioihin. Tällaisiksi perussäännöiksi voidaan ajatella vaikkapa ominaisuuksien pakollisuutta ja olioiden välisten kytkentöjen kardinaliteettiä (engl. cardinality). Muiden sääntöjen osalta ohjelmistoalustaan toteutettiin mahdollisuus liittää liitännäis-luokkia metatiedon määrittelemiin liiketoiminnanluokkiin. Toinen tapa liiketoimintalogiikan ja rajoitteiden toteuttamiseen ovat staattiset luokat, jotka voidaan yhdistää osaksi SafranOne –ohjelmistoalustaa. Seuraavissa alakohdissa on esitelty näitä kolmea yllämainittua tapaa liiketoiminnan sääntöjen ja rajoitteiden toteuttamiseksi.

7.6.1 Metaolioiden sisältämät säännöt

Metaolioiden sisältämät säännöt pitävät sisällään mahdollisuuden määritellä metatietona sääntöjä, jotka rajoittavat liiketoimintaolioiden käyttäytymistä. SafranOne –ohjelmistoalustan meta-olioille toteutettuja sääntöjä ovat liiketoimintaolioiden välisten kytkentöjen kardinaliteetti, ominaisuuksien pakollisuus ja muokattavuus.

Liiketoimintaolioiden kytkentöjen kardinaalisuus on määritelty *sf_relation_type* –luokan ilmentymällä ilmoittaen kahden liiketoimintaluokan välisen riippuvuuden toisistaan. Kardinaalisuus kertoo kuinka monta liiketoimintaoliota esiintyy liiketoimintaluokkien välisessä suhteessa. Kardinaalisuus on asetettu kytkentöjen molempiin päihin ja dynaaminen oliomalli tarkistaa kardinaalisuuden asettamat rajoitteet uusia kytkentöjä ja liiketoiminnanolioita luotaessa ja täten varmistaa dynaamisen oliomallin eheyden.

Viitaten alikohdan 7.1.3 esimerkkivaatimukseen, on mahdollista määritellä SafranOne Administrator –ohjelmalla kardinaalisuus projekti ja aktiviteetti –liiketoimintaluokkien välille kuvan 7.14 mukaisesti. Projektiin voi kuulua nollasta mielivaltaiseen määrään aktiviteetteja (Kuva 7.14, Cardinality Down:0-N) ja jokaisen aktiviteetti tulee kuulua yhteen ja vain yhteen projektiin (Kuva 7.14, Cardinality Up:1-1).



	Relation Type	Name	Description	Table Type Up	Table Type Down	Cardinality Up	Cardinality Down
▶	projekti_aktiviteetti	Projektin aktiviteetit		projekti	aktiviteetti	1-1	0-N
*							

Kuva 7.14 Kytkentöjen kardinaalisuuden määritteleminen

Liiketoimintaolioiden pakollisuus ja muokattavuus voidaan määritellä liiketoimintaolioiden ominaisuuksille eli *sf_column_type* –luokan ilmentymille. Ominaisuus voi olla joko pakollinen tai sen arvon voi jättää asettamatta. Muokattavuus voidaan määritellä sallituksi aina tai vain silloin kuin ollaan luomassa uutta liiketoimintaoliota tai sen muokkaaminen voidaan estää kokonaan.

Alikohdan 7.1.3 esimerkkivaatimukseen viitaten on mahdollista määritellä SafranOne Administrator –ohjelmalla projekti ja aktiviteetti –liiketoimintaluokkien ominaisuuksien pakollisuudet ja muokattavuudet kuvan 7.15 esittämällä tavalla. Aktiviteetin kaikki ominaisuudet ovat pakollisia (Kuva 7.15, Mandatory) ja niiden muokkaaminen on aina

sallittu (Kuva 7.15, Allow Edit,yes). Projektin ominaisuuksista vain nimi on pakollinen tieto ja se voidaan asettaa vain uutta projektia luotaessa (Kuva 7.15, Allow Edit,new).

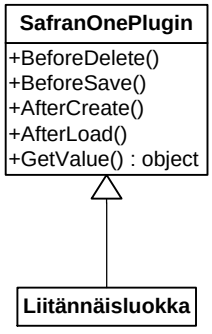
	Column Type	Name	Description	Column Domain	Data Table	Data Column	Mandatory	Allow Edit	Format
▶	aktiviteetti__nimi	Nimi		nimi	sf_string_value	data_value	☑	yes	
	aktiviteetti__aloituspäivä	Aloituspäivä		päivämäärä	sf_date_value	data_value	☑	yes	
	aktiviteetti__lopetupäivä	Lopetuspäivä		päivämäärä	sf_date_value	data_value	☑	yes	
*							☐		

	Column Type	Name	Description	Column Domain	Data Table	Data Column	Mandatory	Allow Edit	Format
	projekti__nimi	Nimi		nimi	sf_string_value	data_value	☑	new	
	projekti__aloituspäivä	Aloituspäivä		päivämäärä	sf_date_value	data_value	☐	no	
	projekti__lopetuspäivä	Lopetuspäivä		päivämäärä	sf_date_value	data_value	☐	no	

Kuva 7.15 Liiketoimintaolioiden pakollisuuden ja muokattavuuden määritteleminen

7.6.2 Liitännäiset

Koska SafranOne:n dynaamisessa oliomallissa ei määritelty metaolioita varsinaisten sääntöjen ja rajoitteiden luomiseen, luotiin dynaamiseen oliomalliin arkkitehtuuri liitännäisluokkien lisäämiseksi jälkikäteen dynaamiseen oliomalliin monimutkaisten sääntöjen, laskentojen ja rajoitteiden toteuttamiseksi. Kuvassa 7.16 on esitetty liitännäisluokkien rakenne, jossa itse liitännäisluokka perii dynaamisessa oliomallissa määritellyn SafranOnePlugin –luokan. Liitännäisluokassa voidaan ylikirjoittaa SafranOnePlugin –luokan metodeja ja täten vaikuttaa itse metatietona määritellyn liiketoimintaluokan toimintaan.



Kuva 7.16 SafranOne -ohjelmistoalustan liitännäisluokkien rakenne

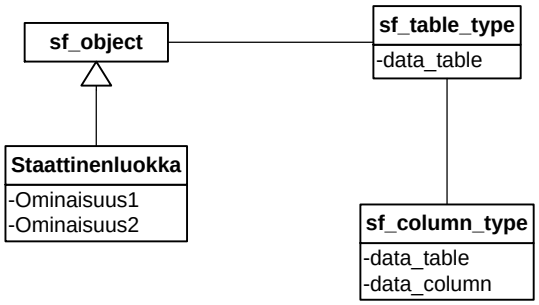
Alikohdan 7.1.3 esimerkkivaatimukseen viitaten on mahdollista luoda projekti –liiketoimintaluokalle liitännäinen, joka muodostaa projektin aloituspäivän ja lopetuspäivän

sen sisältävien aktiviteettien ääripäivistä. Alla on esitetty kyseisen liitännäisluokan yksi mahdollinen toteutus.

```
public class projekti_plugin : SafranOnePlugin
{
    protected override object GetValue(sf_object obj, sf_column_type colType)
    {
        if ("projekti__aloituspäivä" == colType.column_type)
        {
            List<sf_object> ls = obj.getRelationObjects("projekti_aktiviteetti", false);
            DateTime min = DateTime.MaxValue;
            foreach (sf_object o in ls)
            {

                if (min > ((DateTime)o.getValue("aktiviteetti__aloituspäivä")))
                    min = (DateTime)o.getValue("aktiviteetti__aloituspäivä");

            }

            if (!DateTime.MaxValue.Equals(min))
                return min;
        }

        if ("projekti__lopetuspäivä" == colType.column_type)
        {
            List<sf_object> ls = obj.getRelationObjects("projekti_aktiviteetti", false);
            DateTime max = DateTime.MinValue;
            foreach(sf_object o in ls)
            {
                if (max < ((DateTime)o.getValue("aktiviteetti__lopetuspäivä")))
                    max = (DateTime)o.getValue("aktiviteetti__lopetuspäivä");

            }

            if (!DateTime.MinValue.Equals(max))
                return max;
        }

        return Guid.Empty;
    }
}
```

7.6.3 Staattisen ja dynaamisen oliomallin yhdistäminen

Mikäli metaolioiden sisältämät säännöt ja liitännäisluokkien toteuttaminen eivät riitä liiketoimintaolioiden sääntöjen, laskentojen ja rajoitteiden määrittelemiseen voi staattisten luokkien toteuttaminen olla ainoa ratkaisu tarpeiden tyydyttämiseksi. Tällaisissa tapauksissa SafranOne:n dynaaminen oliomalli mahdollistaa staattisen ja dynaamisen oliomallin yhdistämisen sekä staattisen luokan kuvauksen määrittelemisen metatietona. Staattinen luokka voidaan lisätä osaksi dynaamista oliomallia periyttämällä se dynaamisen oliomallin liiketoiminnan olioita kuvaavasta luokasta *sf_object* kuvan 7.17 mukaisesti.



Kuva 7.17 Rakenne staattisen luokan yhdistämiseksi osaksi dynaamista oliomallia

Staattisen luokan ollessa peritty *sf_object* –luokasta, voidaan se määritellä omaksi liiketoimintaluokakseen metatietoon luomalla sitä vastaava *sf_table_type* –luokan ilmentymä. Staattisen luokan sisältämät sisäiset ominaisuudet voidaan määritellä *sf_column_type* –luokan ilmentyminä dynaamiseen oliomalliin. Staattiselle luokalle voidaan myös määritellä dynaamisen oliomallin muiden liiketoimintaluokkien tapaan dynaamisesti uusia ominaisuuksia ja kytkentöjä toisiin dynaamisen oliomallin liiketoiminnan luokkiin. Staattisen luokan liittämisen osaksi dynaamista oliomallia mahdollistaa *sf_table_type* ja *sf_column_type* –luokilla oleva ominaisuus *data_table* (ks. Kuva 7.17), joka kertoo viittaako metatieto johonkin muuhun luokkaan kuin sf_object –luokkaan. *Sf_column_type* –luokalla on tämän lisäksi ominaisuus *data_column*, joka kertoo mihin mahdolliseen staattisen luokan muuttujaan kyseinen metatieto määritys viittaa. Kuvassa 7.18 on esitetty kuvitteellisen staattisen luokan metatiedon määritteleminen siten, että liiketoiminnanluokan ominaisuudet viittaavat dynaamisen oliomallin sijasta staattisen luokan ominaisuuksiin.

	Column Type	Name	Description	Column Domain	Data Table	Data Column	Mandatory	Allow Edit	Format
	staattinenluokka__ominaisu...	Ominaisuus1		nimi	staattinenluokka	Ominaisuus1	☐	yes	
	staattinenluokka__ominaisu...	Ominaisuus2		nimi	staattinenluokka	Ominaisuus2	☐	yes	

Kuva 7.18 Staattisen luokan ominaisuuksien määritteleminen SafranOne Administrator –ohjelmalla

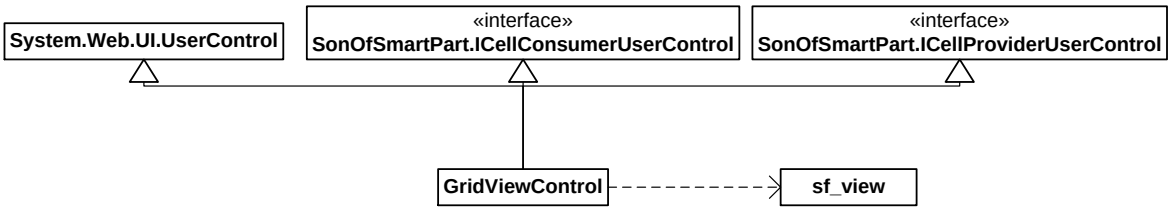
7.7 Käyttöliittymä

Dynaamista oliomallia tukemaan toteutettiin tämän diplomityön puitteissa kaksi käyttöliittymäkomponenttia; puu- ja taulukkokomponentti. Puukomponentti pystyy

esittämään mitä tahansa dynaamisessa oliomallissa määritettyä kytkentää kahden liiketoimintaluokan välillä esittäen oliot hierarkkisena rakenteena. Taulukkokomponentti pystyy esittämään dynaamisen oliomallin liiketoimintaoliot ja niiden ominaisuuksia taulukkona ja mahdollistaa näiden muokkaamisen lomakkeella.

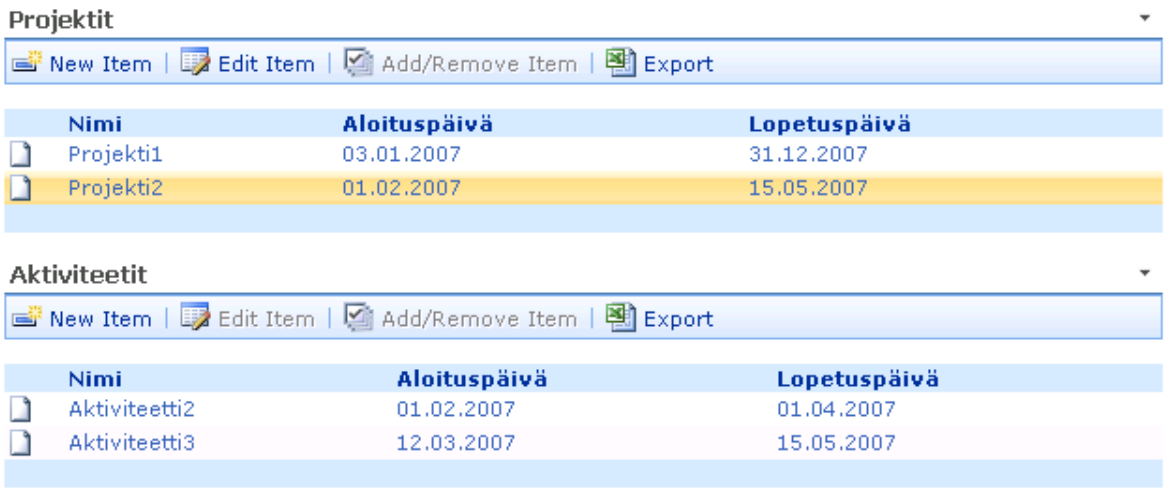
7.7.1 Taulukkokomponentti

Dynaamista oliomallia tukeva taulukkokomponentti pystyy esittämään liiketoimintaolioita taulukkona ja lomakkeena hyödyntäen metatiedossa kuvattuja näkymiä. Toisin sanoen taulukkokomponentti tulkitsee näkymiä muodostaakseen niitä vastaavan käyttöliittymän sarakkeineen ja liiketoimintaolioineen. Taulukkokomponentti on toteutettu ASP.NET –käyttäjäkontrollina ja se toteuttaa SmartPart –tuotteen sisältämät rajapinnat. Rajapinnat mahdollistavat taulukkokomponentin kytkemisen toisiin komponentteihin sen ollessa lisättynä SharePoint –sivustolle SmartPart WebPart:n avulla. Kytkemisen avulla taulukkokomponentti pystyy kuuntelemaan muiden komponenttien valintaa tai kertomaan oman valintansa muille. Kuvassa 7.19 on esitetty taulukkokomponentin rakenne.

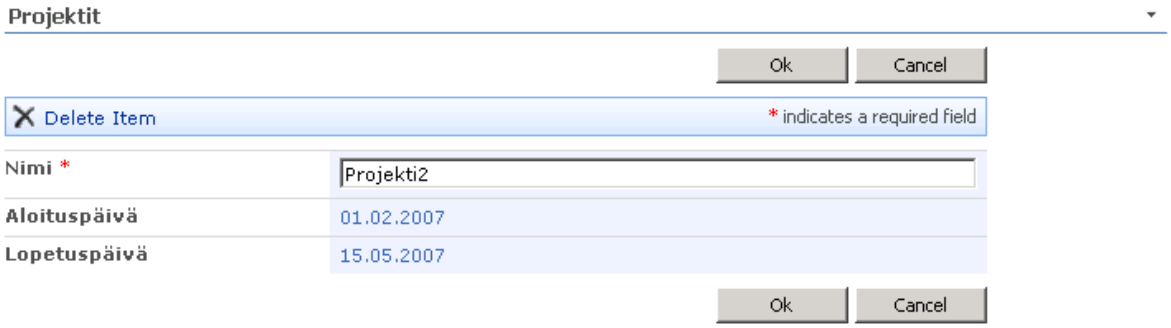


Kuva 7.19 Taulukkokomponentin rakenne

Alikohdan 7.1.3 esimerkkivaatimuksen perusteella muodostettiin kohdassa 7.5 näkymät esimerkkivaatimuksen liiketoimintaolioille. Näitä näkymiä hyödyntävät taulukkokomponentit on esitetty kuvassa 7.20. Kuvassa projektilista on kytkettynä sen alla olevaan aktiviteettilistaan, joka esittää vain valitun projektin sisältämiä aktiviteetteja. Kuvassa 7.21 taulukkokomponentti näyttää valitun projektin lomakkeena, joka mahdollistaa kyseisen liiketoimintaolion muokkaamisen.



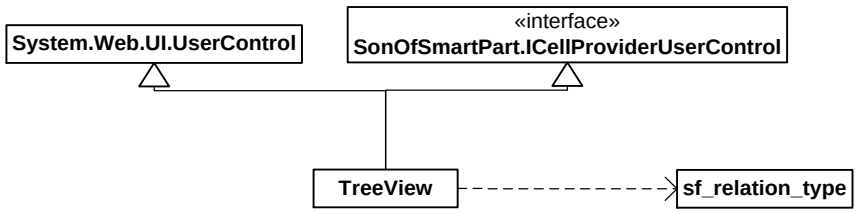
Kuva 7.20 Projektien ja aktiviteettien esittäminen taulukkokomponentilla



Kuva 7.21 Projektin muokkaaminen lomakkeella

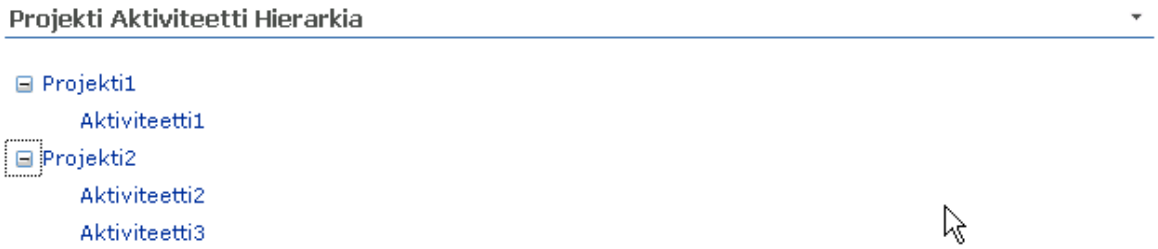
7.7.2 Puukomponentti

Dynaamista oliomallia tukeva puukomponentti pystyy esittämään liiketoimintaolioiden välisiä kytkentöjä hierarkkisena rakenteena hyödyntäen metatiedossa kuvattua kytkentää kahden liiketoimintaolion välillä. Taulukkokomponentin tapaan puukomponentti on toteutettu ASP.NET –käyttäjäkontrollina. Se toteuttaa myös SmartPart:n rajapinnan, jonka avulla se voi kertoa omasta valinnastaan muille komponenteille SharePoint –sivulla. Puukomponentin rakenne on esitetty kuvassa 7.22.



Kuva 7.22 Puukomponentin rakenne

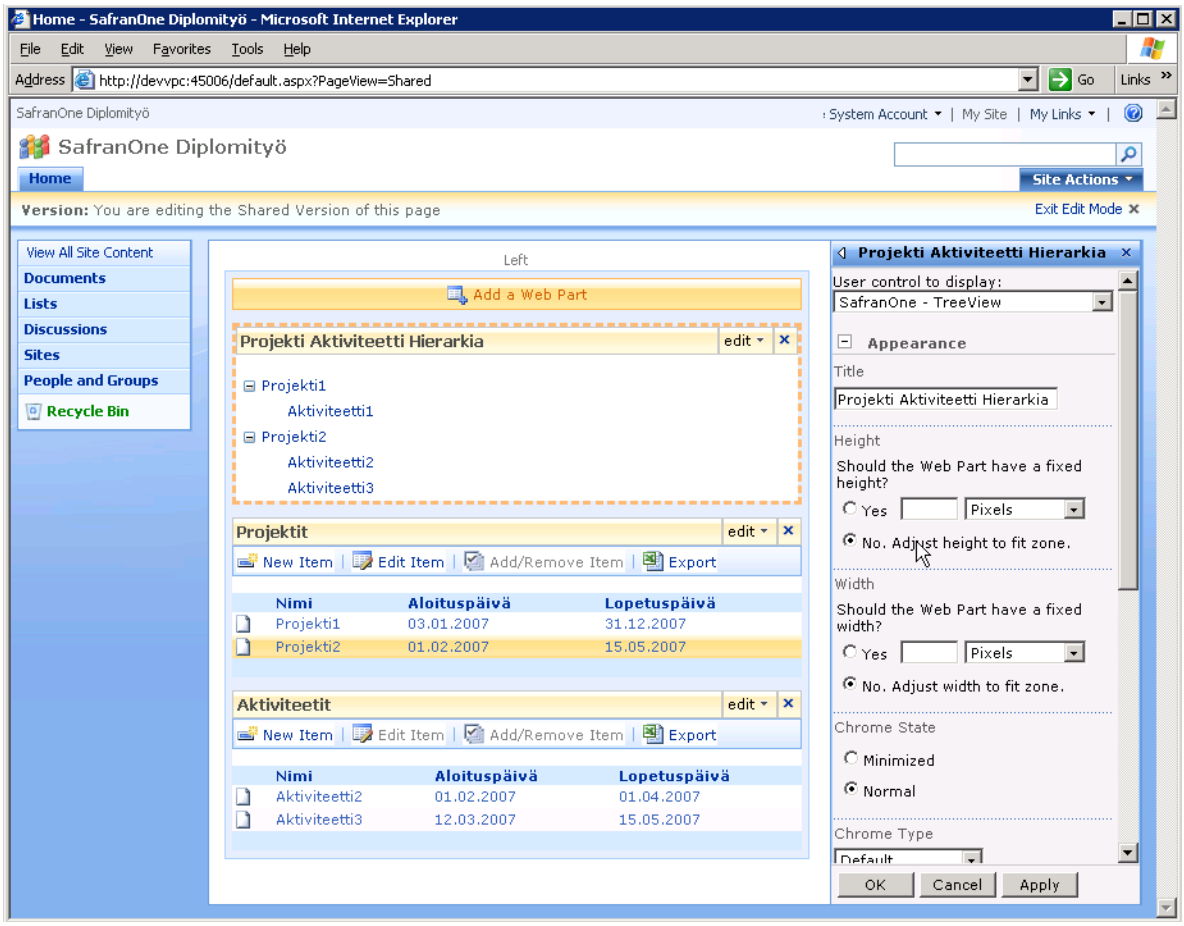
Alikohdan 7.1.3 esimerkkivaatimuksen perusteella muodostettiin kohdassa 7.4 kytkentä esimerkkivaatimuksen projekti ja aktiviteetti –liiketoimintaolion välille. Tätä kytkentää hyödyntävä puukomponentti on esitetty kuvassa 7.23.



Kuva 7.23 Projektin ja aktiviteetin kytkennän esittäminen puukomponentilla

7.7.3 Komponenttien käyttö

Dynaamista oliomallia tukevien komponenttien ollessa ASP.NET –käyttäjäkontrolleja ne voidaan esittää Microsoft SharePoint –porttaalisovelluksen sivulla SmartPart WebPart:n avulla. Kuvassa 7.24 on Sharepoint:n muokkaustilassa oleva sivu, jolle on lisätty kolme kappaletta SmartPart WebPart:eja. Valitun WebPart:n konfigurointi onnistuu kuvan oikeassa reunassa olevan lomakkeen avulla. Lomakkeella voidaan valita näytettävä komponentti ja muokata sen vaatimia asetuksia. Näitä asetuksia ovat esimerkiksi taulukkokomponentin esittämät näkymät ja puukomponentin esittämä liiketoimintaolioiden välinen kytkentä.



Kuva 7.24 Käyttöliittymäkomponenttien konfigurointia SharePoint:n sivulla

Osa C: yhteenveto

8 SafranOne:n dynaamisen oliomallin analyysi

Tässä luvussa analysoidaan toteutetun SafranOne -ohjelmistoalustan dynaamista oliomallia. Oliomallia verrataan kirjallisuudessa ja tiedeyhteisössä esitettyihin dynaamisen oliomallin ratkaisuihin ja listataan niiden välillä olevia eroavaisuuksia ja yhtäläisyyksiä. Tämän lisäksi käsitellään joitain mahdollisia tulevaisuuteen liittyviä parannuksia SafranOne:n dynaamiseen oliomalliin.

8.1 Eroavaisuudet ja yhtäläisyydet kirjallisuuteen

SafranOne:n dynaaminen oliomalli seuraa perusrakenteeltaan tieteellisissä artikkeleissa esitettyjä ratkaisumalleja. Tyyppiolio-ratkaisumallia on käytetty sellaisenaan yleistämään liiketoimintaoliot yhden luokan ilmentymiksi. Niin liiketoimintaolioiden sisältämät ominaisuudet, kuin olioiden väliset kytkennätkin on toteutettu ominaisuus-ratkaisumallia hyväksi käyttäen. Olioiden ominaisuudet on tosin toteutettu sillä eriäväisyydellä, että itse ominaisuuksien tietotyyppi on yleistetty vielä kertaalleen *sf_column_domain* –luokan ilmentymäksi. Tämä ilmentymä mahdollistaa tiettyjen käsitteellisten tietotyyppien koostaminen kokonaisuuksiksi, joita voidaan myöhemmässä vaiheessa hyödyntää mahdollisen ohjelmalogiikan yhdistämiseksi näihin käsitteellisiin asioihin.

Suurimmat eroavaisuudet kirjallisuudessa esitettyjen ratkaisumallien ja SafranOne:n välillä syntyvät ohjelmalogiikan ja sääntöjen toteutuksissa dynaamiseen oliomalliin. Kirjallisuudessa säännöt toteutetaan dynaamiseen oliomalliin erillisten strategia-ratkaisumallin mukaisten sääntöolioiden avulla. SafranOne:n dynaamisessa oliomallissa säännöt ovat osa liiketoimintaolioiden sisältämää metatietoa ilman erillisiä sääntöolioita tai ne on koodattu staattisesti käyttäen liitännäisluokkia tai staattisia luokkia, jotka on peritty dynaamisen oliomallin luokista.

8.2 Tulevaisuus

Vaikka SafranOne:n dynaamisella oliomallilla pystytään jo tällaisenaan määrittelemään melko kattavia tietojärjestelmiä, on diplomityön aikana noussut kuitenkin joitain ideoita tulevaisuuden kehityskohdiksi dynaamiseen oliomallin.

Metatiedon historiointi

Metatiedon historiointi mahdollistaisi liiketoimintaolioiden säilyttämisen sellaisenaan metatiedon muuttuessa ja mahdollistaisi tiedon esittämisen ja tutkimisen sellaisena, kuin se on järjestelmään menneisyydessä syötetty.

Ehdollinen metatieto

Jotta järjestelmällä pystyttäisiin tukemaan monimuotoisia liiketoimintaprosesseja, tulisi jossain tapauksissa pystyä antamaan metatiedolle ehdollisuuksia, joiden mukaan liiketoimintaoliot määräytyisivät. Tällainen tilanne syntyy esimerkiksi, kun liiketoimintaolioiden sisältämä tieto eroaa yrityksen osastojen välillä ja täten metatiedon avulla pitäisi pystyä määrittelemään ehdollisuuksia osastokohtaisesti.

Liiketoimintaolioiden välisten kytkentöjen riippuvuudet

Metatiedon avulla tulisi pystyä ilmaisemaan miten olioiden väliset kytkennät riippuvat toisistaan. Tämän avulla voitaisiin rajoittaa liiketoimintaolioiden välisiä kytkentöjä toisten järjestelmässä olevien olioiden ja kytkentöjen kautta.

Näkymien sisäkkäisyys

Tällä hetkellä SafranOne tukee näkymiä vain tiettyyn liiketoimintaolioon. Jotta mahdollistettaisiin monimuotoisempien näkymien laatiminen, tulisi sallia näkymien asettaminen sisäkkäin. Täten pystyttäisiin esittämään useita eri tyyppisiä liiketoiminnan olioita saman näkymän kautta. Näkymä muodostuisi siis useasta erillisestä näkymästä rekursiivisesti ja sen muodostamisessa käytettäisiin hyväksi liiketoimintaolioiden välisiä kytkentöjä.

XML tekniikoiden hyödyntäminen käyttöliittymäkontrolleissa

Erilaisten XML –tekniikoiden käyttö käyttöliittymäkontrollien muodostamisessa voisi olla järkevää. Esimerkiksi XSLT –muunnosten avulla voitaisiin muodostaa liiketoimintaolioiden tiedosta näkymien avulla HTML –käyttäjäkontrolleja.

9 Tavoitteiden saavuttaminen

Tässä luvussa käsitellään luvussa kolme diplomityölle asetettuja tavoitteita ja arvioidaan niiden onnistumista. Tärkeimmiksi tavoitteiksi diplomityölle asetettiin dynaamiseen oliomalliin tutustuminen ja tähän malliin pohjautuvan SafranOne –ohjelmistoalustan toteuttaminen. Diplomityötä sivuavina tavoitteina oli Microsoft –teknologioihin perehtyminen ja tiedon saattaminen yrityksen tuotekehityksen tietoon.

9.1 Dynaaminen oliomalli

Dynaamisen oliomallia tutkittiin kirjallisuudessa ja tieteellisissä artikkeleissa esitettyjen ratkaisumallien avulla. Aiheesta löytyi paljon myös tietoa, joka on jäänyt tälle diplomityölle asetettujen rajojen ulkopuolelle. Tärkeänä havaintona on myös, että uutta tietoa ja uusia ratkaisumalleja dynaamiseen oliomalliin julkaistaan jatkuvasti ja aihetta käsitellään edelleen kansainvälisissä konferensseissa. Tulevaisuuden kannalta dynaaminen oliomalli vaikuttaa lupaavalta tavalta kehittää sovelluksia ja sen suunnittelun ratkaisumalleja on kenties mahdollista hyödyntää myös muissa yrityksen tuotekehityshankkeissa.

9.2 SafranOne –ohjelmistoalusta

SafranOne –ohjelmistoalusta toteutettiin hyödyntäen dynaamista oliomallia, jossa metatiedolla pystytään määrittelemään liiketoimintaolioita ja niihin liittyviä sääntöjä. Tämän diplomityön asettamien vaatimusten puitteissa voidaan katsoa, että kehitetty SafranOne –ohjelmistoalusta toteuttaa sille asetetut alkuperäiset tavoitteet. Myynnin ja konsulttien tehtäväksi jää todentaa pystytäänkö SafranOne –ohjelmistoalustalla toteuttamaan asiakkaiden vaatimukset täyttäviä projektihallintajärjestelmiä Microsoft SharePoint –portaalisovellukseen.

9.3 Teknologiset tavoitteet

SafranOne –ohjelmistoalusta toteutettiin Microsoft –teknologioita hyödyntäen ja näistä teknologioista sekä niiden suomista mahdollisuuksista saatiin arvokasta tietoa kehitystyön ohessa. Tärkeimpinä saavutuksina olivat .NET –sovellusympäristön hyödyntäminen

Windows –sovellusten ja verkkosovellusten tekemiseen. Tämän lisäksi saatiin tietoa kuinka mahdollistetaan ASP.NET käyttäjäkontrollien integrointi Microsoft SharePoint –portaalisovellukseen SmartPart:n avulla. Näitä saavutettuja tietoja pystytään mahdollisesti käyttämään hyödyksi myös yrityksen tulevaisuuden Microsoft –teknologioihin pohjautuvissa tuotekehityshankkeissa.

10 Loppulause

Diplomityö ei varsinaisesti esitä mitään uutta keksintöä, vaan se esittää kuinka tunnettujen suunnittelun ratkaisumallien avulla saadaan toteutettua dynaamiseen oliomalliin perustuva ohjelmisto, jonka liiketoiminnan luokkia voidaan dynaamisesti muuttaa metatietoa muuttamalla. Diplomityö tutkii dynaamisen oliomallin ratkaisumalleja kirjallisuuden avulla ja esittää kuinka tämän tiedon avulla saadaan kehitettyä dynaamiseen oliomalliin perustuva ohjelmistoalusta Microsoft –teknologioita käyttäen.

Yritykselle diplomityön tuomia hyötyjä ovat uusi dynaamiseen oliomalliin perustuva ohjelmistoalusta SafranOne sekä kehitystyön ohessa syntynyt osaaminen Microsoft –teknologioista ja itse dynaamisesta oliomallista. Tätä syntynyttä osaamista ja tietoa voidaan tulevaisuudessa käyttää yrityksen muissa tuotekehityshankkeissa ja SafranOne –ohjelmistoalustan jatkokehityksessä.

Lähdeluettelo

[1] Johnson, R., 5.11.1998, Dynamic Object Model. [Viitattu 01.10.2007]. Saatavissa: http://st-www.cs.uiuc.edu/users/johnson/papers/dom/DynamicObjectModel.pdf

[2] Ralph, E., Johnson, E., Oakes, J. The User-Defined Product Framework. Ralph E. Johnson, E., Documenting Frameworks using Patterns. OOPSLA '92, ACM SIGPLAN Notices 27, 10 (1992). s. 63-70.

[3] Yoder, J. W., Foote, B., Riehle, D., Tilman, M. Metadata and active object-models. In Addendum To the 1998 Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (Addendum) (Vancouver, British Columbia, Canada). J. Haungs, Ed. OOPSLA '98 Addendum. ACM, New York, NY, 1998

[4] Riehle, D., Tilman, M., Johnson, R., Dynamic Object Model. Manolescu, D., Völter, M., Noble, J. Pattern Languages of Program Design 5. Addison-Wesley, 2005.

[5] Razavi, R., Bouraqadi, N., Yoder, J. W., Perrot, J., Johnson, R. Language support for adaptive object-models using metaclasses. Computer Languages, Systems & Structures 31, 3-4 (2005). s. 199-218

[6] Dastas, A., Yoder, J. W., Borba, P., Johnson, R. Using Aspects to Make Adaptive Object-Models Adaptable. Cazzola, W., Chiba, S., Saake, G. Proc. ECOOP'04 Workshop on Reflection, AOP, and Meta-Data for SoftwareEvolution (RAM-SE), 2004. s. 9-19

[7] Yoder, J. W., Balaguer, F., Johnson, R., Architecture and Design of Adaptive Object-Models. ACM SIGPLAN Notices 36, 12 (2001). s. 50-60

[8] Crous, T., Danzfuss, T., Liebenberg, A., Moolman, A., Adaptive Object Modelling using the .NET Framework. Journal of .NET Technologies 3, 1-3 (2005). s. 177-182

[9] Revailt, N., Yoder, J. W., Adaptive Object-Models and Metamodeling Techniques. Workshop Results; ECOOP 2001, Budapest, 2001. Springer Lecture Notes in Computer Science 2323 (London, 2002). s. 57-71

[10] Yoder, J. W., Balaguer, F., Johnson, R., Adaptive Object-Models for Implementing Business Rules. Position Paper for Third Workshop on Best-Practices for Business Rules

Design and Implementation, OOPSLA 2001. [Viitattu 01.10.2007]. Saatavissa: http://www.adaptiveobjectmodel.com/OOPSLA2001/AOM_OOPSLABusinessRulesWorkshp.pdf

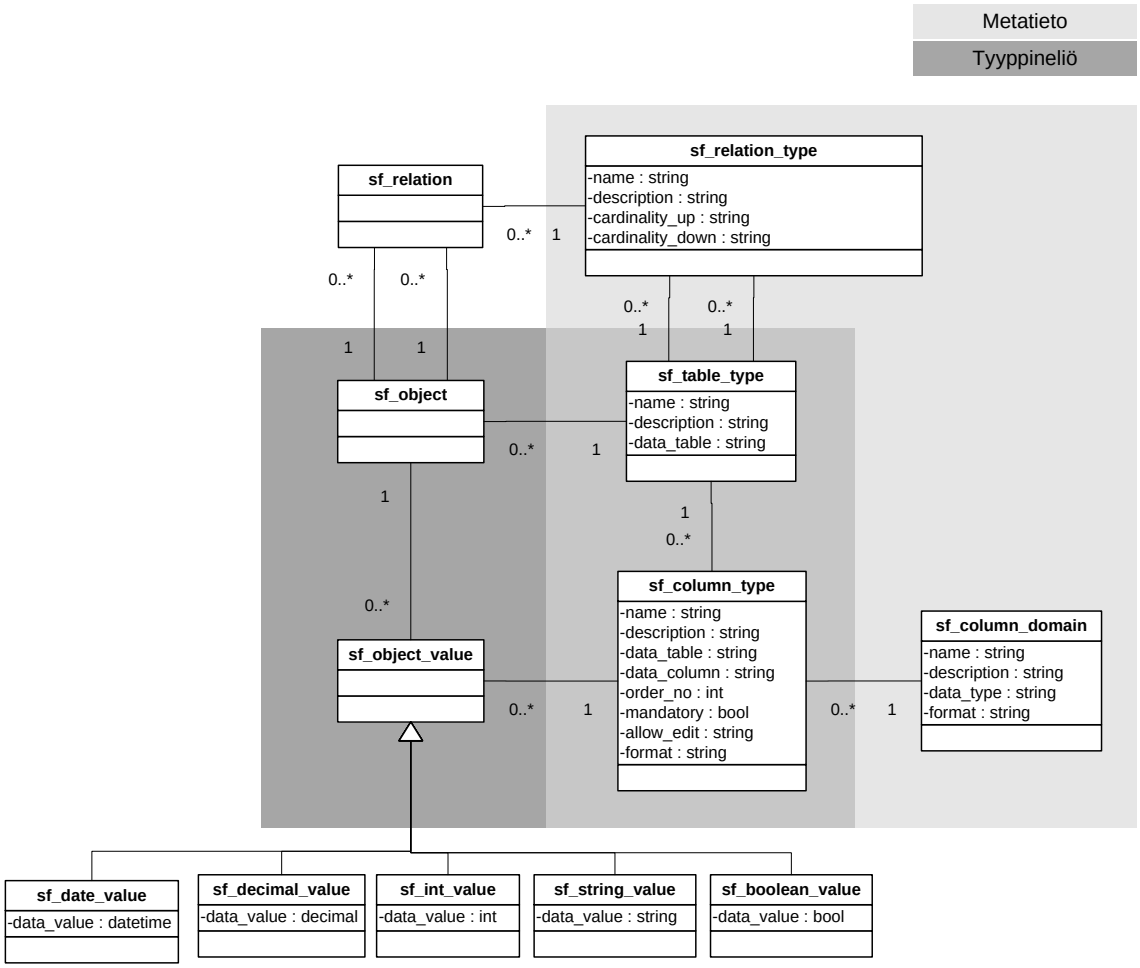
[11] Mai, Y., Li, J., Butler, G., Difficult Issues in Designing Adaptive Object Model Systems. ICEIS 2004, Proceedings of the 6th International Conference on Enterprise Information Systems, Porto, Portugal, 2004. s. 295-302

[12] Süloglu, S., 5.12.2006, A Way to Model Dynamic Systems. [Viitattu 01.10.2007]. Saatavissa: http://www.ceng.metu.edu.tr/~coseml/AWayToModelDynamicSystems.pdf

[13] Welicki, L., Yoder, J. W., Wirfs-Brock, R., A Pattern Language for Adaptive Object Models: Part I - Rendering Patterns. 14th Pattern Language of Programs of Programs Conference - PLoP 2007, Monticello, Illinois, USA, 2007. [Viitattu 01.10.2007] Saatavissa: http://www.hillside.net/plop/2007/papers/PLoP2007_WelickiEtAl.pdf

[14] Yoder, J. W., Jonhson, R., The Adaptive Object-Model Acrhitectural Style. WICSA 3: Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture, 2002. s. 3-27

Liite 1: SafranOne:n dynaamisen oliomallin luokkakaavio



Liite 2: SafranOne:n dynaamisen oliomallin tietokannan ER -kaavio