TIETOMALLIGENERAATTORI

Turo Soisenniemi

Opinnäytetyö
Toukokuu 2013
Tietotekniikan
koulutusohjelma
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

TURO SOISENNIEMI:
Tietomalligeneraattori

Opinnäytetyö 59 sivua, joista liitteitä 24 sivua
Toukokuu 2013

Tämän työn tavoitteena on ollut suunnitella ja tuottaa tietomalligeneraattori Insta DefSec Oy:n eräässä tuotekehitysprojektissa kehitettävän tuoteperheen käyttöön. Projektissa käytössä ollut tietomalli vaati generoinnilta erityispiirteitä, joita markkinoilla olleet ratkaisut eivät tarjonneet.

Työn aikana kartoitettiin generaattorin vaatimuksia. Vaatimuksiin perustuen luotiin toteutussuunnitelma. Suunnitelma katselmoitiin ja hyväksytyn katselmoinnin jälkeen generaattori toteutettiin työn aikana. Toteutusta varten perehdyttiin erilaisiin tekniikoihin, joilla generaattori voitaisiin toteuttaa. Työssä esitellään valitut tekniikat. Työssä käsiteltiin myös generaattorin käyttöönottoprosessi sekä esitellään kehitysehdotuksia generaattorin uusiksi ominaisuuksiksi.

Generaattori otettiin käyttöön välittömästi sen valmistuttua. Generaattori saavutti kaikki sille asetetut tavoitteet.

Asiasanat: generointi, Java, tietotekniikka, ohjelmistotekniikka, tietomalli, generointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information Technology
Software Engineering

TURO SOISENNIEMI:
Data Model Generator

Bachelor's thesis 59 pages, appendices 24 pages
May 2013

The purpose of this thesis was to design and implement data model generator for a project family at Insta DefSec Oy. Data model of the project required special features from data model generator. There were no possible options in the market.

During the thesis requirements for data model generator were made. Plan for implementing the generator were made based on requirements. Plan was reviewed and after successful review, generator was implemented. During the implementation phase different generator technologies were explored. Chosen technologies are introduced in this thesis. Deployment of data model generator is covered in the thesis. The thesis contains thoughts how to improve generator later on.

Generator was implemented during the thesis. All requirements were met.

Key words: Java, information technology, software engineering, data model, generating

SISÄLLYS

1 JOHDANTO ... 7
2 TIETOMALLI ... 8
2.1 Projektin tietomallin ongelma ... 9
2.2 Projektin tietomallin rakenne ... 9
2.3 Tietomalliobjektien generoinnin tarve ... 11
2.4 Generaattorin edeltäjä ... 11
3 VAATIMUKSET ... 13
3.1 Periytyminen ... 13
3.2 Pakettihierarkia ... 14
4 SUUNNITTELU ... 16
4.1 Määritystiedosto ... 17
4.2 Parsinta ... 18
4.3 Generointi ... 19
5 TOTEUTUS ... 22
5.1 Rakenne ... 22
5.1.1 Parsinta ... 22
5.1.2 Generointi ... 23
5.2 Käytetyt tekniikat ... 23
5.2.1 Reflektio-standardikirjasto ... 24
5.2.2 JCodeModel-kirjasto ... 24
5.2.3 Option-luokka ... 25
5.3 Toiminta ... 25
5.3.1 Määritystiedostojen parsinta ... 25
5.3.2 Dataluokkien generointi ... 26
5.3.3 Malliluokkien generointi ... 27
5.3.4 Tehdasluokkien generointi ... 27
5.3.5 Varastoluokkien generointi ... 27
5.3.6 Viitteiden generointi ... 28
5.3.7 Apuluokka ... 28
6 KÄYTTÖÖNOTTO ... 29
6.1 Vanhan mallin siirto ... 29
6.2 Uuden generoinnin käyttöönotto ... 30
7 KEHITYSEHDOTUKSIA ... 31
7.1 Oletusparametrit ... 31
7.2 Tuotteistaminen ... 31
7.3 Rajapintojen toteutus ... 32

7.4 Poisto ja muuttaminen 32
7.5 Versionhallintatuki 33
7.6 UML-luokkakaavio tuki 34
LÄHTEET 35
LIITTEET 36
Liite 1. Käyttöohje 36
Liite 2. Esimerkkejä määritystiedostoista ja generoinnista 40

LYHENTEET JA TERMIT

Java	Java on Sunin kehittämä ohjelmointikieli. Kielen uusin versio on tällä hetkellä 7. Nykyään Javaa hallinnoi Oracle.
Annotaatio	Annotaatio on keino määritellä metadataa, jota voidaan myös lukea ajonaikaisesti. Annotaatiot on lisätty Javaan versiossa 5. Annotaatio merkitään @-merkillä.
Java Generics	Java-kielen versiossa 5 on lisätty tuki geneerisille tyypeille.
sisäluokka	Javan luokan sisällä määritetty luokka (”inner class”).
UML	Unified Modeling Language. Standardi graafinen mallinnuskieli. Sisältää erilaisia kaaviotyyppejä ohjelmistojen kuvaamiseen.
Hajautettu versionhallinta	Hajautettu versionhallinta tarkoittaa, että jokaisella kehittäjällä on kopio koko historiasta. Muutokset tehdään paikalliseen kantaan ja mahdollisesti julkaistaan muille kehittäjille.
Haara	Versionhallintaohjelma saattaa tukea rinnakkaista kehitystä. Toisistaan erillään olevia kopioita, joihin voidaan tehdä muutoksia, kutsutaan haaroiksi (englanniksi branch). Haaroja voidaan aloittaa olemassa olevasta haarasta ja yhdistää takaisin tai johonkin toiseen haaraan.

1 JOHDANTO

Työn tarkoituksena oli tuottaa toimiva ratkaisu tietomallin generointiin käynnissä olevaan projektiin. Luvussa 2 esitellään tietomalleja yleisesti ja projektissa käytössä olevaa tietomallia. Samassa luvussa kerrotaan miksi tietomalli oli generoitava. Seuraavassa luvussa käsitellään vaatimuksia ja tarpeita generaattorille sekä pohditaan yleisiä generoinnin ongelmia. Samassa luvussa esitellään hieman edellistä ratkaisua, joka oli käytössä ennen tätä työtä.

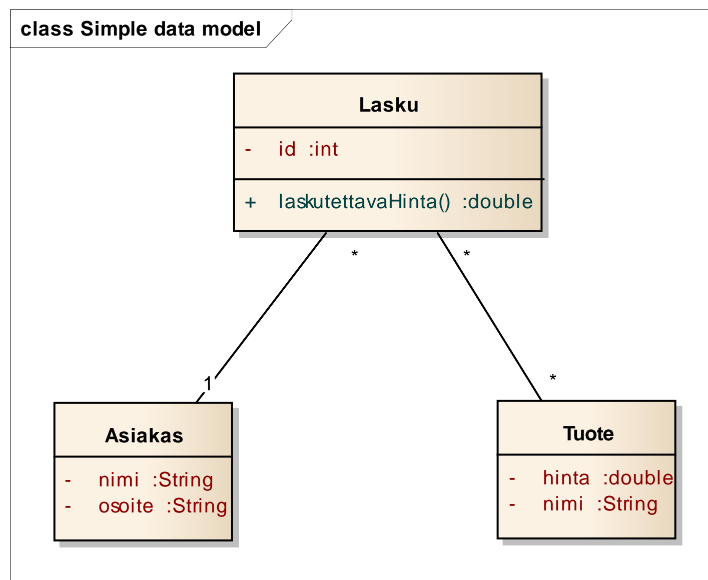
Tavoitteena oli suunnitella ja toteuttaa generaattori, jonka avulla voidaan generoida tietomallin käsitteet Javan luokiksi. Työn alkuvaiheessa on esitelty suunnitelma generaattorin toteuttamiseksi (luku 4). Generaattorin toteutukseen valittuja tekniikoita käsitellään luvussa 5. Samassa luvussa myös esitellään toteutusratkaisuja ja generaattorin toimintaa.

Luvussa 6 käydään läpi käyttöönotto. Käyttöönoton osana käsitellään olemassa olevan mallin siirtoa generoitavaksi. Samalla perehdytään kokemuksiin käyttöönotosta.

Työn lopussa on koottu kokonaiskuva ja tarkastellaan saavutettiinko tavoitteet. Painopiste on kuitenkin kehitysehdotuksilla. Kehitysehdotukset liittyvät suurimmaksi osaksi käytännössä havaittuihin tarpeisiin.

2 TIETOMALLI

Ohjelmistoprojekti käsittelee jotakin oikean maailman ongelmaa. Käsitteitä, jotka kuvaavat oikean maailman asioita, sanotaan tietomalliksi. Esimerkiksi laskutusjärjestelmän tietomallin käsitteitä voisivat olla ”Lasku”, ”Asiakas” ja ”Tuote”. Kuvassa 1 on esitetty yksinkertainen mahdollinen tietomalli.



KUVA 1. Yksinkertainen tietomalli

Tietomallin käsitteet ovat yleensä yksinkertaisia. Niitä vastaavat luokat eivät välttämättä sisällä logiikkaa. Yksinkertaisuutensa vuoksi tietomallin luokat voidaan usein generoida. Generointi voidaan esimerkiksi suorittaa UML:n luokkakaaviosta (kuva 1). Useimmat generaattorit eivät tarjoa logiikan generointia vaan ainoastaan tynkien metodien, asetus- ja hakumetodien generoinnin.

Yleisesti saatavilla olevat tietomalligeneraattorit tarjoavat ratkaisuja tietynlaisiin ongelmiin. Generaattoria ei välttämättä voi määrittää vaan tuotos on kaikille samanlaista. Toisaalta generaattori saattaa tarjota oman kielen, jonka avulla generaattorin toimintaa voi määrittää. Generaattorin käyttö saattaa siis vaatia kokonaan uuden kielen opettelua. Jälkimmäisessä tapauksessa generaattorilla saatetaan määritellä generoitavia logiikkaa sisältäviä metodeja.

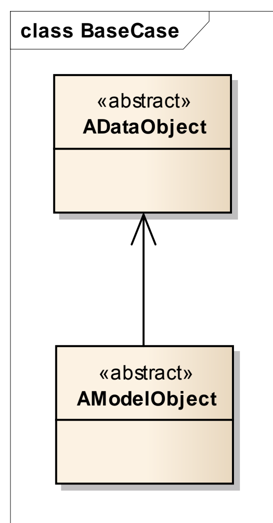
2.1 Projektin tietomallin ongelma

Käytettävä tietomalli sisältää paljon loogisia metodeja, jotka on toteutettava käsitettä vastaaviin luokkiin. Logiikka on yhtenevää, joten se voitaisiin generoida. Logiikan generointi ei kuitenkaan ole mahdollista kaikilla markkinoilla olevilla generaattoreilla.

Projektissa koodi on yhteisomistuksessa. Tietomallin käsitteet ovat siis jokaisen ohjelmoijan muokattavissa. Koodin yhteisomistajuus johtaa siihen, että jokaisen ohjelmoijan tulisi osata muokata tietomallin käsitteitä. Tämä rajaa valmiiden ratkaisuiden käyttöä entisestään.

2.2 Projektin tietomallin rakenne

Projektissa on käytössä kaksiosainen tietomalli (kuva 2). Dataobjekti (ADataObject) käsittää perustietotyypit ja viitteet muihin dataobjekteihin. Malliobjekti (AModelObject) sisältää ohjelmoijan tarvitsemat rajapinnat datan päivittämiseen. Dataobjektit ovat muuttumattomia, mutta niiden päivitys luo uuden version. Malliobjekti on suunniteltu niin, että sillä voi hakea saman version tietoa dataobjektista. Tietomallin kaikki käsitteet periytyvät näistä kahdesta kantaluokasta.



KUVA 2. Tietomallin kantaluokat

Tietomallin objekteja luodaan erillisellä tehdasluokalla. Tehdasluokkaan (AObjectFactory) on kerätty kaikkien mahdollisten malliobjektien luontimetodit. Luontimetodi sisältää kaikki tarvittavat parametrit luokan ja sen sisäluokkien luontiin.

Näissä metodeissa siis luodaan sekä malli että dataobjekti ja palautetaan malliobjekti. Tietyn käsitteen kaikkien esiintymien haku tapahtuu varastoluokan (AModelStorage) kautta, jossa on hakumetodit tyypeittäin. Kuvassa 3 on esitetty esimerkin avulla haku- ja luontimetodi.

```
AObjectFactory factory = getFactory();
Human human = factory.createHuman("First","Last");

AModelStorage storage = getStorage();
Set<Human> humans = storage.getHumans();
```

KUVA 3. Haku- ja luontimetodi esimerkit

Malli- ja dataobjekti tunnistetaan yksilöllisellä tunnisteella. Tunniste on tallessa sekä malli- että dataobjektissa ja mallilla on tiedossa oma dataobjektinsa. Tunnisteen on täsmättävä objektien välillä.

Tietomallissa on vielä erikoispiirteenä se, että kokoelmat tallennetaan oman tyyppisinä malli- ja dataobjekteinaan. Käytännössä nämä on toteutettu Javan sisäluokkina. Tällä varmistetaan, että kokoelma on oikeaa tyyppiä (kuva 4).

```
class DepartmentData {
    …
    class Employees extends DList {
    …
    }
}
```

KUVA 4. Sisäluokat tietomallissa

Jotta viittaukset voisivat toimia, on olemassa luokka, joka sitoo staattisesti malliobjektin ja dataobjektin toisiinsa. Samoin varastoluokassa on staattinen sidonta tietyn sisäluokan ja hakumetodin välillä. Kuvassa 5 on esimerkin avulla esitetty sidonta.

```
static {
    ModelObjectMapping.registerMapping(Department.class,
        DepartmentData.class);
}
```

KUVA 5. Staattinen sidonta

Staattisen sidonnan tarkoituksena on se, että voidaan yhdistää oikea malliobjekti dataobjektiin. Malli- ja dataobjektien sisäluokat rekisteröidään samalla tavalla. Tämä liittyy olennaisesti tietomallin sisäiseen toimintaan, mutta uusia käsitteitä lisättäessä tai vanhoja muokatessa, staattinen sidonta on päivitettävä. Rekisteröinti on siis generoitavissa.

2.3 Tietomalliobjektien generoinnin tarve

Käytössä oleva tietomalli on työlästä kirjoittaa ja se on lähes identtisesti toistuvaa monilta osin. Sen takia on todettu, että on tuottavampaa ja virheettömämpää generoida tietomalliin liittyvät objektit. Ilman generointia ohjelmoijan tulisi muuttaa useaa luokkaa lisätessään tietomalliin uuden käsitteen. Ohjelmoijan käyttämä aika ei kuitenkaan varsinaisesti toisi lisäarvoa projektiin vaan olisi käsin tehtynä virhealtista ja mekaanista työtä.

Generoimalla tietomalli voidaan ohjata resurssit pois mekaanisesta työstä. Tietomallin yhtenäisyys on myös taattu, koska generointi tapahtuu samalla tavalla riippumatta ohjelmoijasta. Yhtenäisyys takaa sen, että jos tietomallin luokista löytyy virhe, se on korjattavissa generoinnista. Yksittäiselle tietomallin luokalle väärin toteutettua metodia ei siis pääse syntymään.

2.4 Generaattorin edeltäjä

Tarve generaattorille ilmeni samalla, kun tietomalli muotoutui nykyiseen muotoonsa. Tietomallin yhteydessä tehtiin generaattori, jota laajennettiin tarpeen mukaan. Käytännön kokemukset osoittivat, että tarpeiden mukaan syntyneessä generaattorissa oli ongelmia. Heikkouksia oli muun muassa periytymisen puuttuminen. Periytyminen oli toteutettu Javan rajapinnoilla niin, että ainoastaan lehtiluokat olivat oikeasti luokkia eikä

kantaluokkia ollut olemassa. Metodit, jotka olisivat kuuluneet kantaluokkaan, kopioitin jokaiseen lehtiluokkaan.

Toinen suuri heikkous oli pakettihierarkian puuttuminen. Kaikki malliobjektit generoitiin samaan kansioon. Ennen pitkää luokkia olisi ollut valtava määrä samassa polussa. Luokkia ei pystynyt ryhmittelemään kokonaisuuksiin ja tietomallin hahmottaminen koodissa oli hankalaa.

3 VAATIMUKSET

Generaattorin on pystyttävä generoimaan koodi, niin että ohjelmoija kirjoittaa ainoastaan tarvittavat kentät, mahdolliset määritteet kentille ja luokalle sekä logiikan. Generaattorin käyttäjän ei tulisi kirjoittaa yhtään tietomalliin liittyvää uudelleen käytettävää koodia.

Määritystiedostossa ei saa olla riippuvuuksia generoituihin tiedostoihin, jotta ei syntyisi kehäriippuvuutta. Kehäriippuvuus estäisi generaattorin ajamisen generoitujen tiedostojen puuttuessa.

Generaattorin on havaittava mahdolliset virheet ja ei-sallitut toimenpiteet ennen generointia. Muun muassa malliobjektien säilöminen muihin kuin listoihin tai joukkoihin on kiellettyä tietomallin rakenteen vuoksi. Virheet on havaittava ennen generointia, jotta generaattorilla ei tuoteta kääntymätöntä koodia. Esimerkiksi hakurakenteen käyttö on kiellettyä dataobjektissa: ”`private Map<String, User> lempinimi;`”. Kielto johtuu tietomallin toteutuksesta ja rakenteesta.

Kentille ja luokille on voitava antaa erilaisia määrityksiä. Luokan tapauksessa on voitava määritellä sen tallentamisesta tietokantaan ja käyttäytymisestä poistettaessa objekti. Kenttien tapauksessa voidaan määritellä kenttä muutettavaksi tai määrittää sen riippuvuussuhteita muihin generoitaviin luokkiin. Generaattorin on myös tarkastettava, että määritys on sallittu. Osa määrityksistä suoritetaan oletusarvoisesti, jos ei ole ylikirjoittavaa määritystä.

3.1 Periytyminen

Jo muutamalla käsitteellä saadaan aikaiseksi tietomalli, jossa käsitteillä on yhteistä pohjaa. Tästä syystä generaattorin on tuettava olio-ohjelmointia, jotta voidaan noudattaa olio-ohjelmoinnin periaatteita. Generaattorille on asetettu vaatimus, että jokaisen kantaluokan on oltava abstrakti. Lehtiluokat ovat ainoita luotavia olioita. Lehtiluokka tarkoittaa luokkaa, joka ei ole abstrakti ja Javassa se on merkitty final-avainsanalla. Final-avainsana estää kyseisestä luokasta periyttämisen. Kuvassa 6 on esitetty abstrakti- ja lehtiluokka.

```
public abstract class Example {
}

public final class LeafClass extends Example{

  }

//Compile Error
Example example = new Example();

//Compiles
Example example = new LeafClass();
```

KUVA 6. Abstrakti- ja lehtiluokka.

Javaa generoidessa generaattorin on tuettava Javan genericseja. Genericsit määrittävät käännösaikaisia tyyppejä ja periaatteessa niitä voi olla rajaton määrä. Esimerkkinä voidaan käyttää Javan List-rajapintaa. Ilman genericsia List-rajapinta sallii minkä tahansa tyypin. Genericsien avulla voidaan määrittää, että List-rajapinta sallii vain String tyyppisiä objekteja. Generics määritetään kirjoittamalla sallittu luokka kulmasulkeisiin: ”`List<String> listOfStrings`”. String toimii tässä tapauksessa tyyppiparametrina.

Generoitavia luokkia halutaan käyttää kuten muitakin luokkia, joten ne voivat olla tyyppiparametreina. Generaattorin on osattava parsia määritystiedostot genericseista ja korvattava ne generoitavilla luokilla. Generoitu koodi on siis tyyppivarmaa (type safe).

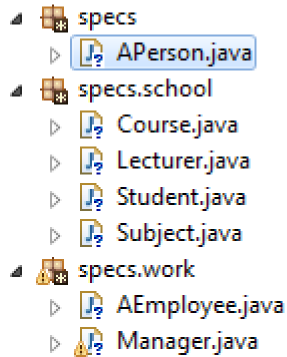
Koska tietomallin kaikilla käsitteillä on yhteinen kantaluokka, on se lisättävä periytymisketjuun. Lehtiluokkien on toteutettava yhteisestä kantaluokasta sekä muista samaan käsiteketjuun kuuluvista kantaluokista periytyvät metodit.

3.2 Pakettihierarkia

Tietomallin käsitteistö voi laajentua valtavaksi, joten käsitteistöä tulisi pystyä järjestelemään hierarkkisesti. Saman otsikon alle kuuluvat käsitteet tulisikin pystyä

jakamaan omiin paketteihinsa. Tämä mahdollistaa vastuualueiden jaon esimerkiksi pakettikohtaisesti. Pakettihierarkia ei kuitenkaan saa estää periytymästä käsitteestä tai esittelemästä käsitettä, joka sijaitsee eri paketissa. Kuitenkin on otettava huomioon, että esimerkiksi generoidusta luokasta ei saa periyttää kehäriippuvuuden estämiseksi.

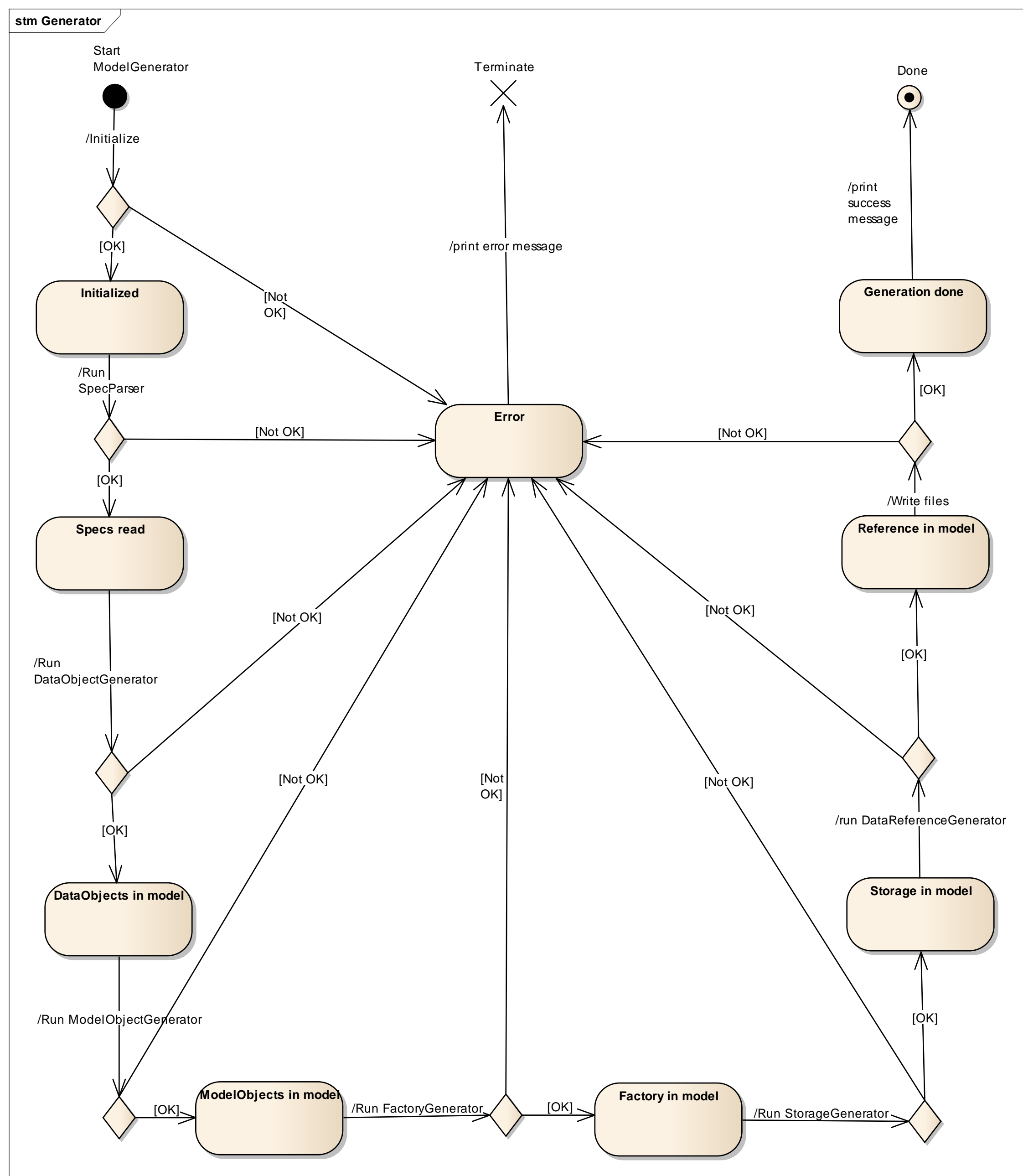
Pakettihierarkian on säilyttävä vastaavana kohdekansiossa. Kuvassa 7 nähdään esimerkki vaaditusta kansiorakenteesta. Kuvasta havaitaan, että yleiset luokat voidaan jättää päätasolle ja muille on luotu oma paketti.



KUVA 7. Pakettihierarkia esimerkki.

4 SUUNNITTELU

Työn aluksi hahmoteltiin sekvenssikaavion ja tilakaavion avulla generaattorin rakenne ja toiminta. Generaattorin ajettava luokka on nimeltään ModelGenerator, jonka tehtävänä on hallita eri generaattorin osia. Vastuualueita on kuusi: määrittelytiedostojen parsinta, data- ja tietomalliobjektien generointi, varasto- ja tehdasmetodien generointi sekä staattiset viitteet. Kuvassa 8 on esitelty edellä mainittu tilakaavio.



KUVA 8. Tilakaavio generaattorin tiloista ja toiminnasta

4.1 Määritystiedosto

Ensimmäisenä oli suunniteltava tiedostomalli, jonka perusteella generoidaan. Tavoitteena oli pitää tiedosto mahdollisimman yksinkertaisena, jotta sen käyttö on helppoa ja nopeaa. Päädyttiin ratkaisuun, jossa yhtä käsitettä vastaa yksi määritystiedosto. Ratkaisun ongelmaksi tulee se, että laajassa käsitteistössä tiedostojen määrä kasvaa. Etuna taas on vastuualueiden jakaminen luokille.

Määrittelytiedostoon luodaan Java-luokka samaan tapaan kuin loisi tavallisenkin luokan. Java-luokka valittiin osittain käyttäjäkunnan perusteella: käyttäjät ovat Java-ohjelmistosuunnittelijoita. Luokalle ei aseteta rakentajaa vaan määrätään ainoastaan kenttiä. Kentän näkyvyys määrittää hakumetodin näkyvyyden. Esimerkiksi kentästä ”`public String name`” generoitaisiin kenttä ”`private String name`” ja hakumetodi ”`public String getName()`”. Kuvassa 9 esimerkki yksinkertaisesta määritystiedostosta.

Jos kentälle halutaan generoida myös asetusmetodi, annetaan sille annotaatio @Mutable, jolla ilmaistaan, että kenttää voidaan muuttaa. Annotaatiolle annetaan enumeraatio EScope parametrina. EScopella on kaksi arvoa, PUBLIC ja PROTECTED. Kuvassa 8 on esimerkki @Mutablen käytöstä. Kyseisestä esimerkistä generoituu asetusmetodi ”`public void setName(String name)`”

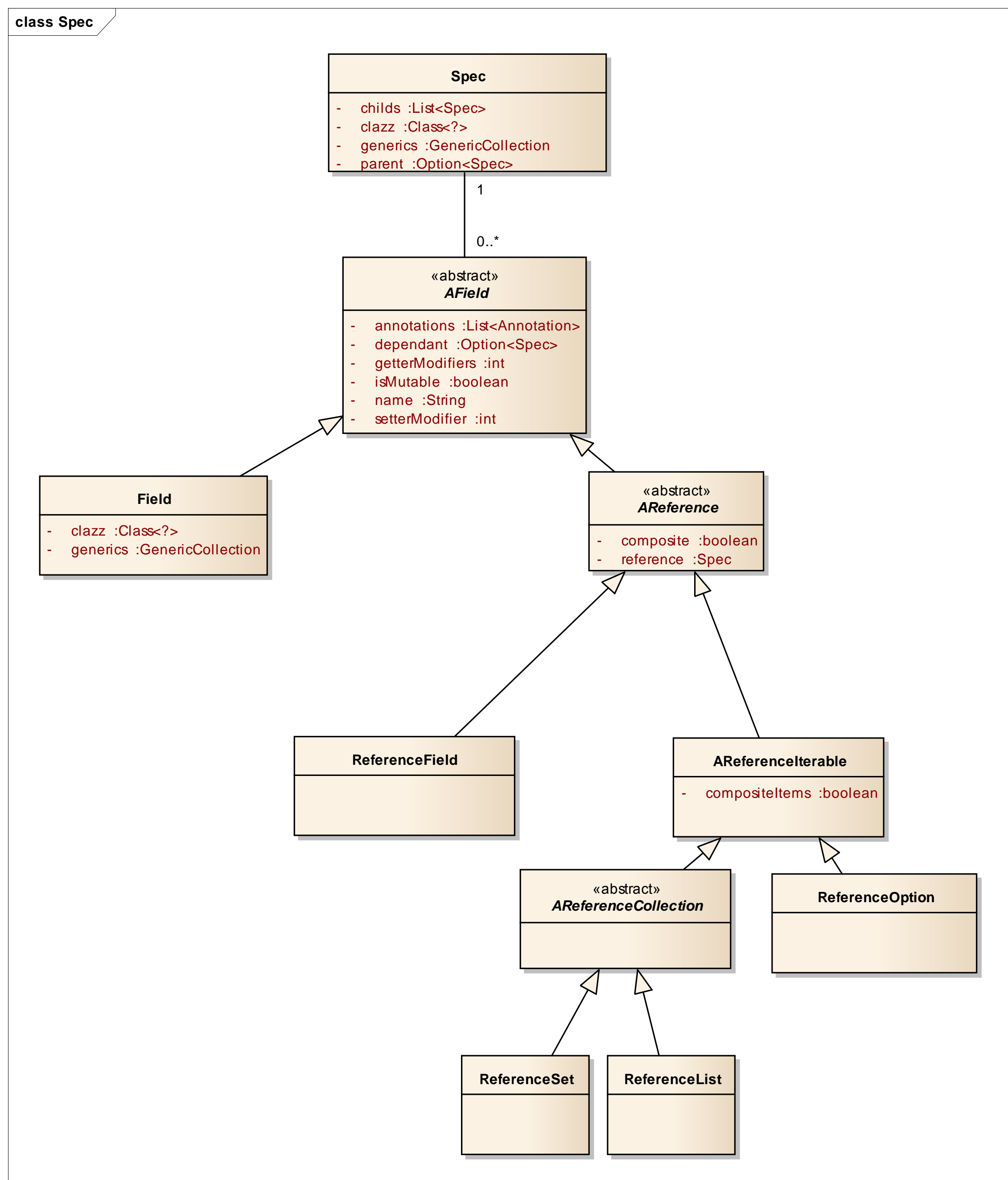
```
class Example {
    @Mutable(EScope.PUBLIC)
    public String name;
}
```

KUVA 9. Yksinkertainen esimerkki määritystiedostosta.

Määrittelytiedostoon liittyy myös muita annotaatioita ja niistä johtuvia sääntöjä. Nämä on selitetty liitteessä 1. Liitteessä 2 on esitetty muutama esimerkki määrittelytiedostosta ja niistä syntyneet generoidut tiedostot.

4.2 Parsinta

Ensimmäinen generoinnin vaihe on käsitellä käyttäjän tekemät määrittelytiedostot ja tarkistaa onko niissä virheitä. Tällä taataan, että generoidaan ainoastaan sallitun muodon mukaisia tiedostoja, jotka ovat käännettävissä. Parsinta vaiheessa myös kootaan generoinnin kannalta oleelliset tiedot tietorakenteeseen (Spec-luokka), jota voidaan välittää myöhemmin generaattorin muille osille. Kuvassa 10 on esitetty luokan suunnitelma.



KUVA 10. Spec-luokka ja kentät

Spec-luokka soveltaa rekursiokooste suunnittelumallia: lapsien lisäksi tiedetään mahdollinen omistaja. (www.oodesign.com: Composite Pattern 2013). Tämän rakenteen avulla voidaan käydä läpi periytymishierarkia ja tarkistaa kehäriippuvuudet.

Luokan sisäisiin rakenteisiin tallennetaan määritystiedostosta luetut kentät. AField-luokassa on tallessa kenttiin liittyvät määritteet. Sekä kentällä että luokkaa kuvaavalla Spec-oliolla on oltava viite alkuperäiseen luokkaansa (Javan Class-luokka). Class-luokan avulla saadaan tarvittavaa tietoa generointia varten.

AField-luokka on yhteinen kantaluokka erityyppisille määritystiedoston kentille. Lehtiluokka Field kuvaa kenttää, johon ei liity toinen määritystiedosto (Spec). Muussa tapauksessa kentän luokka periytyy AReferenceField-luokasta. Tässä tapauksessa kentästä on viite toiseen Spec-olioon.

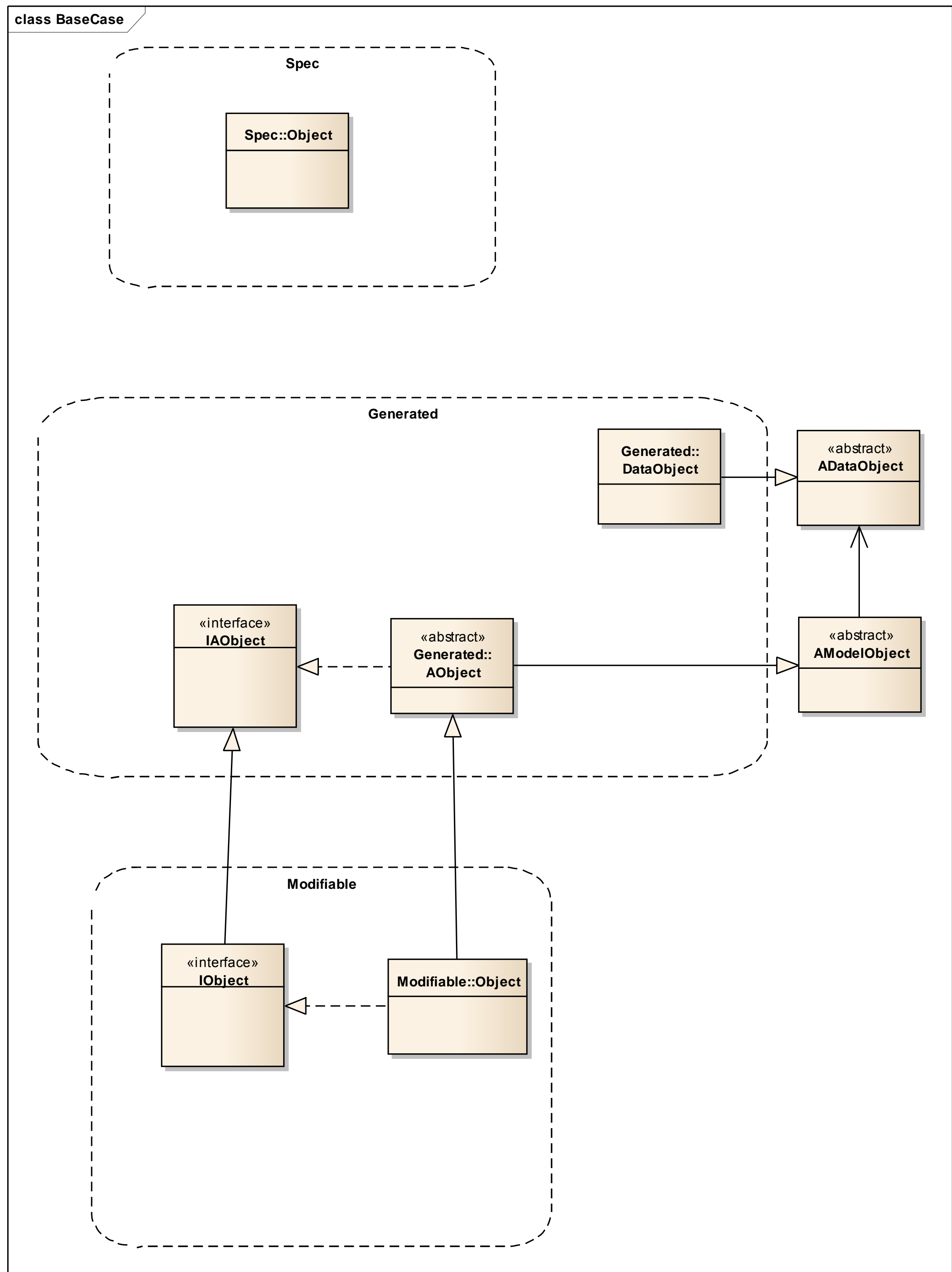
ReferenceField-luokka kuvaa kenttää, jossa kenttä on suoraan jossakin toisessa määritystiedossa määritelty kenttä. Muussa tapauksessa kentän on oltava iteroitava AReferenceIterable-tyyppiä. Erottelu on tehtävä, koska tietomallin rakenne vaatii erilaista käsittelyä tietomallin omille luokille, jotka tallennetaan kokoelmaan tai Option-luokkaan. Option-luokkaa on käsitelty tarkemmin luvussa 5.

4.3 Generointi

Generoinnin jälkeen käyttäjä haluaa kirjoittaa generoituun luokkaan logiikkaa. Ongelmaksi voikin muodostua se, että miten säilyttää kirjoitettu logiikka uudelleen generoitaessa. Asia ratkaistiin generoimalla kaksi luokkaa: abstrakti luokka, joka generoidaan aina ja toteutusluokka, joka generoidaan vain, jos sitä ei ole olemassa (Kuva 10). Tämä mahdollistaa sen, että käyttäjän kirjoittama logiikka ei tuhoudu generoitaessa ja malli voidaan muuttaa generoimalla tiedostot uudestaan. Koska generaattori ei voi havaita muutoksia, uudelleen generointi saattaa aiheuttaa muutoksien jälkeen käännösvirheitä koodissa, jossa on käytetty generoituja luokkia. Jokaiselle generoidulle ja toteutusluokalle tehdään myös rajapinta, jota voidaan käyttää esimerkiksi yksikkötesteissä.

Generaattorin vaatimuksena oli periytyminen. Periytyminen sovitettiin edellä mainittuun malliin niin, että määritystiedostot voivat periytyä toisistaan. Tätä hyväksi

käyttämällä voidaan generoida abstrakti luokka periytymään toisen määritystiedoston kerran generoituvasta luokasta. Periytymisen vaatimuksena on, että määritystiedostossa kantaluokka on asetettu abstraktiksi (kuva 11).



KUVA 11. Generoinnin perustapaus.

Vastaavasti rajapinnat periytyvät. Rajapintoja on ajateltu käytettävän erityisesti testeissä, jolloin projektissa käytössä oleva testausrajapinta mahdollistaisi rajapintojen käytön ilman objektin luontia. Käsitettä kohden luodaan aina kaksi tiedostoa: joka ajokerta generoituva tiedosto ja vain kerran generoituva tiedosto. Ajatuksena on, että jälkimmäiseen käyttäjä tekee muutokset.

5 TOTEUTUS

Generaattorin toteutus aloitettiin suunnittelun katselmoinnin jälkeen välittömästi. Toteutuksen aluksi tutkittiin mahdollisia tekniikoita, joilla toteuttaa generaattori. Samalla rakennetta ja luokkia hahmoteltiin yksityiskohtaisemmiksi.

5.1 Rakenne

Generaattorin toteutus aloitettiin hahmottamalla ensin kokonaiskuva. Ajettavaan tiedostoon (ModelGenerator) koottiin eri toiminnat eli eri osien generoinnille omat metodit. Käytännössä siis toteutettavaksi jäivät yksitellen generoinnin eri osat ja itse generaattori vain käytti varsinaisia toiminnallisia osia oikeassa järjestyksessä.

5.1.1 Parsinta

Ensimmäisenä tietysti on suoritettava parsinta, jotta tiedettäisiin mitä generoidaan. Koska haluttiin, että määritystiedostossa voidaan viitata toiseen määritystiedostoon, toteutettiin parsinta (SpecParser) niin, että aluksi käytiin kaikki tiedostot läpi ja kerättiin niistä ”tyhjät” Spec-tyyppiset oliot, jotka sisälsivät ainoastaan viittauksen alkuperäiseen luokkaan (määritystiedostoon).

Viittauksen avulla alkuperäiseen luokkaan pystyttiin seuraavaksi lukemaan määritystiedostosta kentät, annotaatiot ja muut parsinnan kannalta oleelliset tiedot. Kun tiedossa oli tässä vaiheessa kaikki määritystiedostot, pystytään käsittelemään ne eri tavalla. Generoidessa ei voida enää viitata määritystiedostoon, vaan on viitattava generoituun tiedostoon.

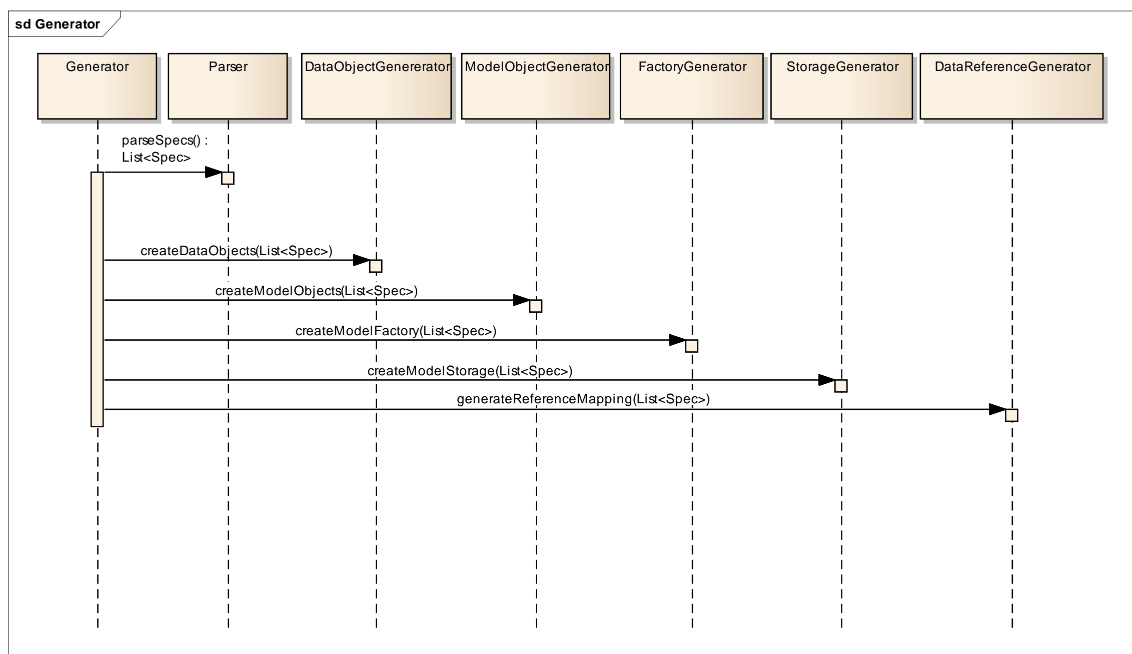
Spec-luokka sisältää kaiken tiedon mitä generointiin tarvitaan. Se sisältää tiedon mihin määritystiedostoon se viittaa. Periytymisen kannalta se sisältää tiedon onko sillä kantaluokkaa ja tiedon kaikista tarvittavista kentistä. Lisäksi Spec-luokassa on säilyttynä mahdolliset annotaatiot sekä määritteet. Määritteet ja annotaatiot saadaan haettua reflektion avulla (luku 5.2.1) Spec-olion Class-tyyppisen kentän avulla.

Parsinnan jälkeen jokainen parsittu Spec-olio tarkistetaan. Tarkistusvaiheessa varmistetaan, että annotaatioita ei ole käytetty väärin ja että kaikki lehtiluokat periytyvät

abstrakteista kantaluokista. Virheen löytyessä generointi keskeytetään ja heitetään poikkeus, joka ilmaisee missä oli virhe, jotta käyttäjä voi korjata määritystiedoston.

5.1.2 Generointi

Vastuualueet jaettiin generaattorille oheisen sekvenssikaavion mukaisesti (kuva 12). Jokainen generaattorin osa suorittaa eri vastuualueen generoinnin. Jokaiselle osa-alueelle on mahdollista määrittää erikseen generoitavien tiedostojen kohdekansio/-paketti.



KUVA 12. Generaattorin toiminta ja vastuualueet

5.2 Käytetyt tekniikat

Tekniikoiden valintaan vaikutti kirjastojen helppokäyttöisyys. Toinen valintakriteeri oli rajoittavuus. Valittu tekniikka ei saisi sulkea pois vaihtoehtoja. Päädyttiin käyttämään kahta kirjastoa sekä projektin omasta toteutuksesta löytyvää luokkaa.

5.2.1 Reflektio-rajapinta

Parsintaan käytettiin Javan tarjoamaa reflektio-rajapintaa (Oracle: Trail: The Reflection API, 2012). Rajapinta tarjoaa keinoja, joilla voidaan ajon aikana lukea luokkien ominaisuuksia, esimerkiksi tarkastella minkä nimisiä kenttiä luokka sisältää ja mikä niiden tyyppi on. Ilman reflektio-rajapintaa olisi jouduttu toteuttamaan Java-koodin lukija tai määrittelemään kokonaan oma määrittelysyntaksi ja lukija sille. Jälkimmäisestä tavasta on projektissa kokemusta vuosien takaa.

Reflektion käytössä on etuna se, että generoitavat tiedostot ovat samaa tyyppiä kuin määrittelytiedostot. Määrittelytiedoston kirjoittajan on siis helppo omaksua syntaksi, koska se on normaalia Javan syntaksia. Suurin etu tulee kuitenkin siitä, että ei tarvitse määritellä kieltä ja kouluttaa käyttäjiä. Generaattorin käyttöön riittää yksinkertainen käyttöohje (liite 1) ja Javan tuntemus.

5.2.2 JCodeModel-kirjasto

Generoinnin apuna päädyttiin käyttämään JCodeModel nimistä generointirajapintaa tarjoavaa kirjastoa (Oracle: JCodeModel (Codemodel Core 2.6 API) 2012). Kirjastossa on täysi tuki Javan kaikille ominaisuuksille, joten teknisiä rajoitteita ei synny. JCodeModel on Sunin kehittämä. Kirjastosta käytettiin versiota 2.6. Kuvassa 13 on esimerkki kirjaston käytöstä. Kirjaston avulla generoitiin kaikki tarvittavat tiedostot. Kirjaston suurimpia etuja oli sen laajuus - kaikki Javan ominaisuudet on tuettuna - ja pakettiriippuvuuksien automaattinen generointi. Olemassa olevaan tai generoitavaan tiedostoon viitattaessa rajapinta osasi itse lisätä import-lauseen generoitavaan tiedostoon.

```
JCodeModel model = new JCodeModel();
JDefinedClass nc = model._class(“modelobject.Example”)

//Generates new files to given folder
model.build(new File(“.”));
```

KUVA 13. JCodeModel kirjaston käyttöesimerkki

Generaattorin edeltäjässä käytettiin JavaParser nimistä kirjastoa (Google Project Hosting: UsingThisParser 2013). JavaParserin heikkoudeksi osoittautui erityisesti pakettiriippuvuuksien generointi. Suurin syy vaihtoon olikin tuo ja lisäksi JCodeModelin käyttö oli intuitiivista dokumentaation niukkuudesta riippumatta.

5.2.3 Option-luokka

Yleinen ongelma ohjelmoinnissa on nolla-osoitin. Nolla-osoittimesta ei voi suoraan päätellä onko tila haluttu. Tähän käytettiin ratkaisuna Option-luokkaa. Option-luokan piirre on, että sen avulla voidaan välttää viittaukset nolla-osoittimeen. Option on geneerinen luokka, joka voi sisältää tyyppiparametrinsa mukaisen arvon. Optionilta voidaan kysyä isDefined metodilla, sisältääkö se arvon. Jos arvo on olemassa, se voidaan pyytää get-metodilla (kuva 14).

```
Option<Department> department = getDepartment();
if(department.isDefined()) {
   raiseSalaryOfDepartment(department.get());
}
```

KUVA 14. Option-luokan käyttö

5.3 Toiminta

Generaattorilla on pääluokka (ModelGenerator), jonka ajamalla tiedostot generoidaan. Generoinnin vastuualueet on jaettu kuuteen eri luokkaan. Jokaisella luokalla on oma vastuualue. Vastuualueet on jaettu parsintaan ja generointiin. Generointi on jaettu generoinnin osa-alueiden mukaan omiin tiedostoihinsa.

5.3.1 Määritystiedostojen parsinta

Parsinnan avuksi toteutettiin SpecParser-luokka, joka tarjoaa kaksi julkista metodia parsimiseen: parseFile(File file) ja parseSpecs(). Ensimmäinen lukee vain minimitiedot ja jälkimmäinen kaiken halutun tiedon. Näiden lisäksi parsitut Spec-oliot saadaan metodilla getSpecs(), joka palauttaa listan Spec-olioita.

Ensimmäinen metodi mahdollisti periyttämisen toteuttamisen: määritystiedostoa parsittaessa SpecParserilla on viittaus mahdolliseen kantaluokkaan tai tässä tapauksessa kantaluokan määritystiedostosta luotuun Spec-olioon. Viittaukset ovat minimissään Spec-olio, jolla on reflektio-rajapinnan Class-luokka muistissa.

ParseSpecs() metodi taas suorittaa varsinaisen parsinnan. Edellisessä kappaleessa mainitun Class luokan avulla voidaan hakea määritystiedoston sisältämät kentät ja mahdollinen kantaluokka. Spec-olioita päivitetään niin, että ne sisältävät tiedon mahdollisesta kantaluokastaan. Kantaluokkaan viitataan Spec-oliona, joka on kapseloitu Option luokan sisään. Kantaluokan lisäksi tallennetaan tiedot luokan kentistä, tyyppiparametreista, annotaatioista sekä tyyppiparametreistä, joilla määritetään kantaluokan tyyppi.

5.3.2 Dataluokkien generointi

Käytetyssä tietomalllissa jokaista käsitettä vastaa ADataObject-luokasta periytyvä tiedon kapselointiin käytetty luokka. Toisaalta taas käsitettä vastaa myös AModelObject luokasta periytyvä luokka, joka tarjoaa käytettävät rajapinnat. DataObjectGenerator (DOG) tarjoaa käytännössä yhden ainoan rajapinnan: void generateDataObjects(List<Spec> spec). DOG alustetaan niin, että sillä on JCodeModel-olio, johon generoitava tieto lisätään.

Käytännössä DOG luo tarvittavat rakentajat ja toteutettavat rajapinnat käsitteen omalle dataobjektille. Spec-olio määrittää mitä kenttiä dataobjektin tulee sisältää ja niille kenttien lisäksi hakumetodit. Jos kenttä on määritetty muutettavaksi, sille luodaan myös asetusmetodi.

Jos käsite sisältää viitteitä muihin käsitteisiin, käsite dataobjekti sisältää ainoastaan viitteen toisen käsitteen dataobjektin tunnisteeseen. Listoille, joukoille ja Option-luokalle, joiden tyyppiparametri on jokin toinen käsite, generoidaan omat malli- ja dataluokat isäntäluokkiensa sisäluokiksi. Dataobjektit sisältävät näissä tapauksissa myös viitteen tunnisteeseen. Muissa tapauksissa dataobjekti sisältää normaalit kokoelmat, esimerkiksi ”`private final List<String> osallistujat`”.

5.3.3 Malliluokkien generointi

ModelObjectGenerator(MOG) luo käsitteistä AModelObject-luokasta periytyvät luokat sekä IModelObject-rajapinnasta periytyvät rajapinnat. Rajapinta sisältää kaikki julkiset asetus- ja hakumetodit, jotka määritystiedostossa on kentille annettu. Itse olioon generoidaan haku- ja asetusmetodit, jotka kutsuvat kyseisen käsitteen dataobjektin vastaavia metodeja. Malliluokka siis sisältää tunnisteen, joka vastaa dataluokan tunnistetta ja itse dataolion.

Malliluokkiin käyttäjä haluaa mahdollisesti kirjoittaa logiikkaa itse, sen sijaan että käyttäisi vain generoituja metodeja. Tästä seuraa ongelma: jos käyttäjä on kirjoittanut logiikkaa, muuttaa määritystiedostoa ja generoi uudestaan luokat, niin kirjoitettu logiikka katoaisi. Ongelma ratkaistiin sillä, että malliluokat ja –rajapinnat ovat kaksiosaisia. Jokaista käsitettä vastaa generoitu malliluokka ja –rajapinta, jotka luodaan aina uudestaan generoitaessa sekä pysyvä malliluokka ja rajapinta, jotka generoidaan ainoastaan, jos niitä ei ole olemassa.

5.3.4 Tehdasluokkien generointi

Projektin mallissa tehdasluokkiin generoitiin rajapinta ja abstrakti luokka, joka toteutti kyseisen rajapinnan. Rajapintaan generoidaan kaikkien lehtiluokkien luontimetodit. Toteutuksien lisäksi abstraktiin luokkaan generoidaan staattinen kutsu, joka rekisteröi kaikki tietomalliin liittyvät käsitteet ladatessa luokka käyttöön.

Erityistä huomiota vaati sisäluokkien luonti. Luontimetodissa pidetään huolta, että kaikki tarvittavat data- ja malliobjektit luodaan sekä kutsutaan tarvittavia rekisteröintimetodeita. Tehdasmetodien luonti onkin generaattorin pienimpiä osia, sillä se ei varsinaista logiikkaa sisällä juurikaan.

5.3.5 Varastoluokkien generointi

Varastoluokan generoinnissa otettiin huomioon myös abstraktit luokat. Vastaavasti kuin tehdasluokkien generoinnissa, varastoa generoitaessa generoitiin rajapinta ja abstrakti luokka. Erona kuitenkin se, että varastoluokan avulla voidaan hakea myös abstrakteja käsitteitä.

Varastoa generoitaessa otettiin käyttöön luokka, jonka avulla käsitteen nimi muunnettiin monikkoon. Esimerkiksi generoitaessa Person-nimistä käsitettä sen varastometodi onkin getPeople. Normaaleissa monikoissa lisätään tarvittaessa s-pääte: Cat-käsitteestä saadaan metodi getCats. Kyseinen ominaisuus on tuettu ainoastaan englanninkielisillä termeillä.

5.3.6 Viitteiden generointi

Viitteiden generointi on generaattorin yksinkertaisin osa. Viitegeneraattori (DataReferenceGenerator) generoi kutsut DataReference-luokan staattisiin metodeihin registerRemovePolicy ja registerReference. Ensimmäisellä kerrotaan käsitteen poistokäytäntö, joka on mahdollisesti määritelty määritystiedostossa käsitteelle. Oletuspoistokäytäntö on, että poistoa ei estetä eikä siihen reagoida mitenkään.

Toinen metodi sitoo malliobjektit dataobjekteihin. Jokaiselle käsitteelle ja niiden sisältämille sisäluokille kutsutaan kyseistä metodia. Sisäluokkia ovat siis kokoelmat ja Optionit, jotka viittaavat mallista löytyvään käsitteeseen.

5.3.7 Apuluokka

Generaattoria varten luotiin apuluokka (GeneratorHelper), koska lähes jokaisessa generaattorin osassa toistui samanlaiset tarpeet. Esimerkiksi hakumetodien nimeäminen on yhtenäistä ja eri generaattorin osat viittaavat ainoastaan JCodeModelin mallissa sijaitseviin olioihin. Apuluokka välitetään muille generaattorin osille luotaessa niitä.

Apuluokan avulla voi hakea viittauksen muun muassa jo käsiteltyyn luokkaan tai luoda sille pohjan. Apuluokan avulla generaattorin eri osat siis keskustelevat toisiensa kanssa. Apuluokassa on myös rajapinnat Spec-olioiden tarkasteluun. Generointiin vaikuttaa esimerkiksi se, jos Spec:ssä on määritetty olion poistoon liittyviä asetuksia tai jos luokka on määrätty abstraktiksi.

6 KÄYTTÖÖNOTTO

Kun generaattori oli saatu testidatalla toimivaan vaiheeseen, aloitettiin käyttöönottoprosessi. Käyttöönottoon oli varattu muutama päivä aikaa ja se tehtiin kehitystyön rinnalla. Käyttöönottoa helpotti se, että projektissa oli käytössä hajautettu versionhallinta.

6.1 Vanhan mallin siirto

Vanha generaattori oli käytössä sillä välin kun kehitettiin uutta. Uuden generaattorin käyttöönottamiseksi vanhalla generaattorilla luodut käsitteet oli siirrettävä uuteen määritystiedostomuotoon. Jotta kehittämiselle ei tulisi pysähdystä, siirtoa varten luotiin versionhallinnassa uusi haara.

Vanhassa mallissa oli kirjoitettu logiikkaa määritystiedostoon, joka sitten generoitaessa kopioitiin generoituihin tiedostoihin. Uudessa mallissa logiikka kirjoitetaan ensimmäisellä kerralla generoitavaan pysyvään tiedostoon. Siirtäessä tietomallin määritystiedostoja uuteen muotoon, oli säilytettävä logiikka. Aluksi määriteltiin uuteen generaattoriin uusi paikka generoitaville tiedostoille, jotta vanha ja uusi malli voisivat elää väliaikaisesti rinnan.

Seuraavaksi oli eroteltava kaikki tieto vanhoista määritystiedostoista niin, että ainoastaan kentät tulevat uusiin määritystiedostoihin. Samalla oli jaettava tiedostot pakettihierarkiaan, sillä vanhassa mallissa pakettihierarkiaa ei ollut. Kun kaikki oli siirretty, suoritettiin ensimmäinen generointi.

Ensimmäisen generoinnin jälkeen pysyvät malliobjektit oli luotu. Tämän jälkeen logiikan siirtäminen oli mahdollista. Logiikan siirron jälkeen oli korjattava viitteet vanhoihin malliobjekteihin. Kun projekti oli saatu kääntyvään tilaan, ajettiin yksikkötestit ja testattiin manuaalisesti toiminnallisuutta.

Kun projektin toiminnallisuus oli testattu, lukittiin pääkehityshaara versionhallinnassa. Lukituksen jälkeen haara, jossa generaattorin käyttöönotto oli tehty, yhdistettiin pääkehityshaaraan. Samalla tarkastettiin vielä mahdolliset muutokset vanhoihin määritystiedostoihin ja tehtiin tarvittavat korjaukset.

6.2 Uuden generoinnin käyttöönotto

Koska kehitystyö projektissa jatkui samanaikaisesti, oli ennen generaattorin ottamista käyttöön yleisesti tarkistettava, että muutoksia ei ole tapahtunut. Käytännössä siis uusimmat muutokset otettiin pääkehityshaarasta ja tarkastettiin muuttuneet määritystiedostot ja lisätyt tiedostot, joissa viitattiin vääriin paketteihin.

Varsinaisesti ongelma ei ollut uuden generoinnin käyttöönotto vaan ennen yhdistämistä avattujen haarojen kanssa toimiminen. Koska lähes kaikki tietomalliin liittyvä oli jollakin tavalla muuttunut, aiheutti se konflikteja haaroja yhdistäessä versionhallinnassa. Päivitettäessä pääkehityshaaraa muihin haaroihin, useat kehittäjät joutuivat käsittelemään samat konfliktit. Ja mahdollisesti vielä käsittelemään niitä uudestaan yhdistäessään oman haaransa pääkehityshaaraan, jos kehittäjät olivat ratkoneet omissa haaroissaan konflikteja eri tavalla.

Käytettäessä generaattoria on huomattu joitakin käytännön ongelmia. Kehitysympäristöt eivät välttämättä siedä kovinkaan hyvin ulkopuolisia muutoksia ympäristöön. Toinen ongelma syntyi, kun generaattori luo tiedostoja, joista käyttäjän ei tarvitse olla tietoinen. Tiedostot on kuitenkin lisättävä versionhallintaan, jotta generaattoria ei tarvitse ajaa paikallisesti otettaessa päivityksiä versionhallinnasta.

7 KEHITYSEHDOTUKSIA

Generaattori on ollut projektissa käytössä valmistumisestaan lähtien. Generaattorissa oli potentiaalia tukea työskentelyä enemmän. Tähän lukuun on kerätty joitakin kehitysehdotuksia.

7.1 Oletusparametrit

Javassa ei ole olemassa käsitettä oletusparametri. Luokan rakentajassa on annettava kaikki tarvittavat parametrit, vaikka ne olisivat suurimman osan kerroista samat. Tässä tietomallissa käytännössä kuitenkin useasti kokoelmat halutaan antaa tyhjinä, Option ei määriteltynä ja useissa tapauksissa perustietotyypit joinakin vakioina. Tällä hetkellä käsitteet, jotka sisältävät useita kenttiä, ovat kohtalaisen työläitä luoda.

Jokaiselle käsitteelle voitaisiin generoida Builder-luokka, joka oletuksena asettaa tyhjän kokoelman tai asetetut oletusarvot rakennettaessa objekti. Jos halutaankin jokin muu arvo oletusarvon sijasta, asetetaan se erillisellä metodilla. Kun halutut arvot, jos yhtään, on saatu asetettua, kutsutaan build-metodia, joka rakentaa varsinaisen objektin.

Määritystiedostoon lisättäessä kenttä, muutoksia saattanee tulla hieman vähemmän tai ei ollenkaan uudelleen generoitaessa. Nykyisellään uuden kentän lisääminen aiheuttaa käännösvirheitä. Jokainen tehdasluokan metodikutsu kyseiselle käsitteelle on väärin, koska lisätty kenttä puuttuu parametreista. Kehittäjän työ myös helpottuu entisestään, kun objekteja luotaessa ei tarvitse luetella kaikkia parametreja.

Vaarana tietysti on, että oletusarvo ei välttämättä olekaan sallittu tietyissä tapauksissa. Objektien sisältämillä tarkistusfunktioilla voidaan ajonaikana varmistaa eheys, mutta parempi olisi, jos virhe tulisi ilmi jo käännösvaiheessa. Tähän tarkoitukseen voisi harkita erillistä annotaatiota, esimerkiksi @Required, jolla ilmaistaan generaattorille, että ei generoida oletusparametria.

7.2 Tuotteistaminen

Koska määritystiedostojen luku onnistuu jo tietystä paikasta ja generaattori on jaettu hyvin pieniin osiin, voisi harkita generaattorin muokkaamista niin, että

generointisäännöt olisi myös säädettäviä. Generaattoria tulisi muokata niin, että se tarjoaa rajapinnat, joilla voidaan säätää generaattorin toimintaa.

Tuotteena generaattoria voisi harkita niin, että sille annetaan eri generaattorin osille tarvittavat sääntötiedostot. Oletuksena olisi jokin yksinkertainen generointisääntö, esimerkiksi dataobjektien generointi ilman yhteistä kantaluokkaa ja yksinkertaisilla asetus- ja hakumetodeilla toteutettuna. Sääntötiedostot olisivat Java-koodia, jotka toteuttaisivat tarjotut rajapinnat.

Käytännössä näiden muutoksien jälkeen generaattorilla voidaan generoida mihin tahansa Java-projektiin tietomalli. Tuotetta varten tarvitsisi lisäksi tuottaa sääntöjä varten dokumentaatio esimerkkeineen. Asetuksia varten voitaisiin luoda erillinen tiedosto, josta generaattori lukisi halutut säännöt ja muut asetukset. Jos halutaan käyttää useita rinnakkaisia generointikokoonpanoja, komentoriviparametrilla voisi asettaa käytettävän asetustiedoston.

7.3 Rajapintojen toteutus

Tällä hetkellä malliobjekti voi toteuttaa ulkopuolisen rajapinnan niin, että ensimmäisen generoinnin jälkeen asetetaan malliobjekti toteuttamaan rajapinta. Koska malliobjektit periytyvät aidosti, tämä toimii kantaluokkien kanssa oikein. Ongelmana on, jos haluttaisiin generoida vastaava malli, toteutettavat rajapinnat eivät säilyisi.

Generaattoriin voitaisiin lisätä tuki rajapintojen toteutukselle. Toteutuksessa pitäisi ottaa huomioon, että generoituja rajapintoja ei saa lisätä määritystiedostoon, jotta ei synny kehäriippuvuutta. Mahdollisesti voitaisiin jopa lisätä sääntöjä, jolla generoidaan rajapinnan metodeja generoituun luokkaan. Tämä olisi toki mahdollista vain yksinkertaisissa tai yhtenevissä toteutuksissa.

7.4 Poisto ja muuttaminen

Näin ketterien menetelmien aikakaudella tietomallin käsitteistö voi elää. Joskus on poistettava käsite tai halutaan vaihtaa käsitteen nimeä. Ongelma syntyy generoidessa siitä, että käyttäjä ei välttämättä ole tietoinen kaikista generoiduista tiedostoista. Toisaalta generaattorikaan ei voi poistaa tiedostoja, jos määritystiedosto poistetaan.

Käyttäjä saattaa haluta kopioida logiikkaa vielä talteen malliobjektista. Uudelleen nimetessä taas generaattori ei voi tietää edellistä nimeä ja toisaalta taas import-lauseiden eheyttä ei voida taata.

Poistoon ratkaisu on helppo. Generaattoriin toteutetaan komentoriviparametri, jolla määrätään poistettava käsite eli määritystiedosto. Generaattori ei tällöin generoi uudestaan vaan siivoa kaikki kyseiseen käsitteeseen liittyvät tiedostot. Muun koodin siivoaminen jätetään käyttäjän vastuulle.

Jos käsitteen nimeä halutaan muuttaa, on suoritettava enemmän toimintoja. Generaattorin olisi etsittävä kaikki tiedostot, jossa kyseistä malliobjektia (ja dataobjektia) käytetään. Kaikki viitteet vanhaan tulisi korvata viitteellä uuteen. Korvattaessa tulisi ottaa huomioon, että kyseisessä tiedostossa ei ole päällekkäin nimettyjä luokkia. Päällekkäin nimettyjä luokkia voidaan käyttää Javassa samassa tiedostossa, jos toiseen (tai molempiin) viitataan koko pakettinimellä luokannimen sijasta. Koska viittaaminen on kuitenkin loogista, korvaaminen voitaisiin kuitenkin tehdä.

Ongelma näissä ratkaisuissa on se, että käyttäjän tulisi tietää näistä ominaisuuksista. Haittaa tosin ominaisuuksilla tuskin saa aikaan, koska versionhallinta hoitaa versioinnin nykyohjelmistoprojekteissa.

7.5 Versionhallintatuki

Generaattoriin voisi lisätä tuen yleisimmille versionhallintaohjelmille. Lisätessä, muutettaessa ja poistettaessa voitaisiin samalla ilmoittaa versionhallinnalle tapahtumista niin, että käyttäjän ei tarvitse siitä huolehtia. Yksinkertaisimmillaan tuki voitaisiin tehdä niin, että generaattorille voidaan määrittää käytettävä versionhallinta, jolloin generaattori olettaa versionhallinnan binääritiedoston löytyvän käyttöjärjestelmän ympäristömuuttujasta. Tarvittaessa generaattori kutsuu kyseistä versionhallintaa oikeilla parametreilla.

7.6 UML-luokkakaavio tuki

Generaattorin toiminnassa on jo olemassa käsitteistöä parsitulle luokalle erilaisine määritteineen. Käytännössä tietomallista voitaisiin generoida käsitteistön perusteella luokkakaavio. Toisaalta tätä voitaisiin soveltaa myös niin, että luokkakaaviota voitaisiin käyttää lähdeaineistona määritystiedostojen sijasta.

Etuna generoitavassa luokkakaaviossa olisi se, että se on ajan tasalla. Toisaalta, jos generoidaan luokkakaaviosta, luokkakaavio olisi varmasti ajan tasalla. Joka tapauksessa generaattorin käyttö varmistaisi sen, että dokumentaatio tietomallista on ajan tasalla.

Ajatuksen voisi myös kääntää toisinpäin. Piirtämällä kaavion määritystiedostoista, UML-luokkakaaviosta voisi generoida määritystiedostot. Etuna on erityisesti määritystiedostojen yksinkertaisuus. Kyseinen generaattori tarvitsisi tuen generoida Javan annotaatioita ja arvoja niille.

LÄHTEET

www.oodesign.com
Composite Pattern
Luettu 16.04.2013
http://www.oodesign.com/composite-pattern.html

Oracle
Trail: The Reflection API
Luettu 03.09.2012
http://docs.oracle.com/javase/tutorial/reflect/

Oracle
JCodeModel (Codemodel Core 2.6 API)
Luettu 05.09.2012.
http://www.jarvana.com/jarvana/view/com/sun/codemodel/codemodel/2.6/codemodel-2.6-javadoc.jar!/com/sun/codemodel/JCodeModel.html

Google Project Hosting
UsingThisParser
Luettu 20.3.2013
http://code.google.com/p/javaparser/wiki/UsingThisParser

LIITTEET

Liite 1. Generaattorin käyttöohje

Tietomalligeneraattorin käyttö

Tiivistelmä

Generaattori ajetaan, jos specs-paketissa on tehty muutoksia. Logiikka kirjoitetaan model-objektiin,. Logiikkaa voi lisätä ajamatta generaattoria. Abstraktien luokkien nimeäminen Coding Standardsin mukaisesti (abstract luokka = ALuokka).

Specin kirjoitus

Spec kirjoitetaan insta-model-generator-projektissa specs-pakettiin tai sen alipakettiin. Esim:

specs.ObjectA
specs.test.ObjectB

Kentät

Kentän esittely:
Kenttä esitellään niin, että sille generoitava getteri luetaan esittelyn näkyvyydestä:

```
protected String name;
public int value;
```

Speceissä toisiin speceihin viitataan normaalisti:

```
public Task task;
```

Jos halutaan luoda MList, MOption, MSet, ne esitellään samaan tapaan kuin muutkin iteroitavat:

```
public Set<String> names;
```

```
public List<Task> tasks;
```

Annotaatiot

@Composite

@CompositeItems(dependency)

Composite ja CompositeItems annotaatiolla kerrotaan, että malliobjekti poistetaan samalla kun omistaja poistetaan. Compositea käytetään yksittäisten malliobjektikenttien kanssa ja CompositeItems-annotaatiota iteroitavien kanssa(Option, List, Set).

CompositeItems:in dependency-arvo on oletusarvoisesti false. Jos kentän objektit ovat riippuvaisia kyseisestä specistä/objektista, annetaan dependency-arvo true:ksi. Esimerkiksi:

```
public class ObjectA {

    @CompositeItems(dependency=true)
    public Set<ObjectB> objects;

}
```

@Dependant

Dependant-annotaatio ilmaisee, minkä objektin on saatava päivityksiä tämän objektin päivittyessä. Dependant-annotaatioita voi olla yksi/spec. Alla esimerkki liittyen edelliseen:

```
public class ObjectB{

@Dependant
public ObjectA objectA;

}
```

@Mutable(EScope)

Tällä annotaatiolla kentälle määrätän setteri. EScopella on kaksi eri arvoa, protected ja public:

```
@Mutable(EScope.PUBLIC)
public String description;
```

Iteroitaville malliobjekteille ei voi/saa asettaa Mutable annotaatiota.

Luokka
Periytyminen toimii normaalisti. Kantaluokkien on oltava abstrakteja:

```
public abstract class AObject{

}

public class ImplementationObject extends AObject {

}
```

Abstraktien luokkien nimeäminen on käyttäjän vastuulla:
Projektin Coding Standards (abstrakti luokka alkaa A-kirjaimella)

Oletusarvoisesti luokka on:

@OnRemoveNoAction

Vaihtoehtoisesti luokalle voi antaa annotaation

@OnRemoveCascade, jolloin kaikki objektin kentät (objektit) poistetaan, kun objekti poistetaan.

Logiikka
Generaattori generoi seuraavat tiedostot (HUOM. jos niitä ei ole olemassa), joihin logiikka kirjoitetaan:

```
specs.ObjectA
=>
desired.package.modelobject.ObjectA
desired.package.modelobject.IObjectA

specs.test.ObjectB
=>
desired.package.modelobject.test.ObjectB
desired.package.modelobject.test.IObjectB
```

Nämä tiedostot eivät muutu ensimmäisen generoinnin jälkeen. Uudelleen generoitaessa generoidut luokat muuttuvat, jolloin muutokset heijastuvat perimisen kautta toteutusluokkiin.

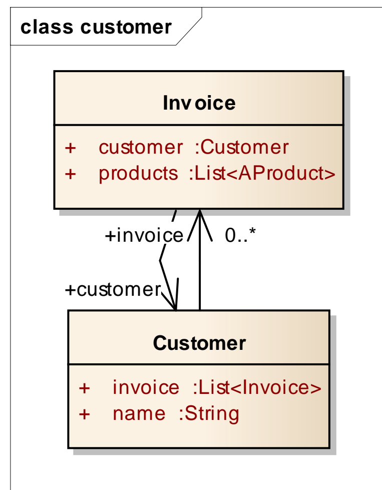
Alla esimerkki custom-metodin kirjoittamisesta.

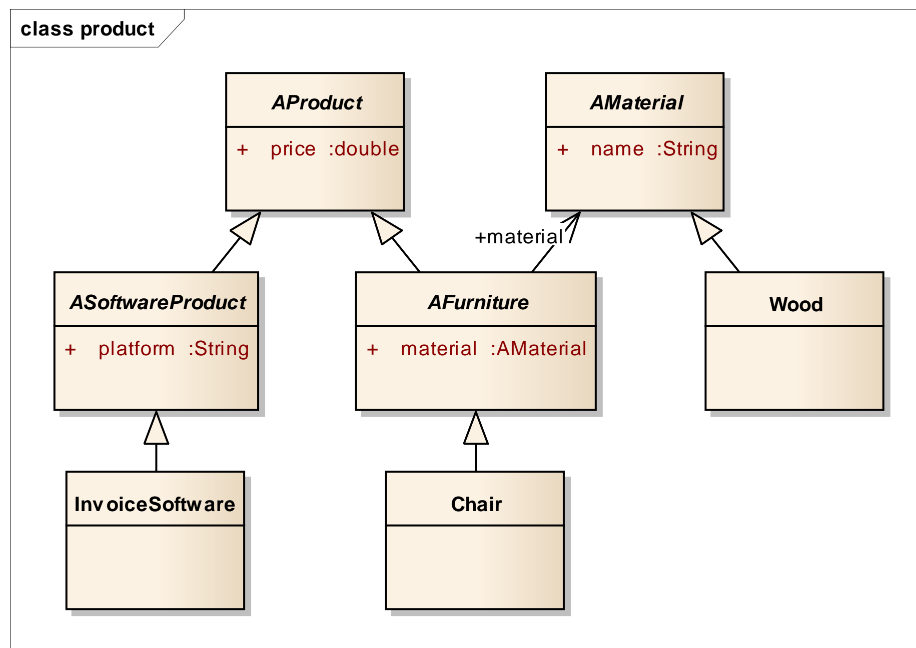
```
public final class ObjectA extends AGeneratedObjectA<ObjectA> implements IObjectA<ObjectA> {
   public EState getState() {
   //Do something
   }
}
public interface IObjectA<MO extends IObjectA<?>> extends IGeneratedObjectA<MO> {
   EState getState();
}
```

Liite 2. Esimerkkejä määritystiedostoista ja generoinnista

1 (20)

Lähtötiedostot (kaavio):





Lähtötiedostot (nimet):

specs.customer.Customer
specs.customer.Invoice
specs.product.AFurniture
specs.product.AMaterial
specs.product.AProduct
specs.product.ASoftwareProduct
specs.product.Chair
specs.product.InvoiceSoftware
specs.product.Wood

Generoidut tiedostot (nimet):

Yleiset:

AModelStorage
AObjectFactory
GeneratedDataReferenceMapping
IObjectFactory
IGeneratedModelStorage

Dataobjektit:

CustomerData
InvoiceData
AFurnitureData
AMaterialData
AProductData
ASoftwareProductData
ChairData
InvoiceSoftwareData
WoodData

Uudelleen generoitavat mallitiedostot:

AGeneratedCustomer
AGeneneratedInvoice
AGeneneratedAFurniture
AGeneneratedAMaterial
AGeneneratedAProduct
AGeneneratedASoftwareProduct
AGeneneratedChair
AGeneneratedInvoiceSoftware
AGeneneratedWood

IGeneratedCustomer
IGeneneratedInvoice
IGeneneratedAFurniture
IGeneneratedAMaterial
IGeneneratedAProduct
IGeneneratedASoftwareProduct
IGeneneratedChair
IGeneneratedInvoiceSoftware
IGeneneratedWood

Kerran generoitavat tiedostot:

Customer	ICustomer
Invoice	IInvoice
AFurniture	IAFurniture
AMaterial	IAMaterial
AProduct	IAProduct
ASoftwareProduct	IASoftwareProduct
Chair	IChair
InvoiceSoftware	IInvoiceSoftware
Wood	IWood

Lähtötiedostot:

```
package specs.customer;

import java.util.List;

import generator.spec.marker.CompositeItems;
import generator.spec.marker.EScope;
import generator.spec.marker.Mutable;

public class Customer {

    @Mutable(EScope.PUBLIC)
    public String name;

    @CompositeItems(dependency=true)
    public List<Invoice> invoice;
}
```

```
package specs.customer;

import java.util.List;

import specs.product.AProduct;

import generator.spec.marker.Dependant;

public class Invoice {

    @Dependant
    public Customer customer;

    public List<AProduct> products;
}
```

```
package specs.product;

public abstract class AProduct {
    public double price;
}

package specs.product;

public abstract class AMaterial {
    public String name;
}

package specs.product;

public abstract class AFurniture extends AProduct {
    public AMaterial material;
}

package specs.product;

public abstract class ASoftwareProduct {
    public String platform;
}
```

```
package specs.product;

public class Chair extends AFurniture {
}
```

```
package specs.product;

public class InvoiceSoftware extends ASoftwareProduct {

}
```

```
package specs.product;

public class Wood extends AMaterial {

}
```

Yleiset tiedostot:

```
package ont.model;

import generator.IModelStorage;
import ont.data.modelobject.customer.Customer;
import ont.data.modelobject.customer.Invoice;
import ont.data.modelobject.product.AFurniture;
import ont.data.modelobject.product.AMaterial;
import ont.data.modelobject.product.AProduct;
import ont.data.modelobject.product.ASoftwareProduct;
import ont.data.modelobject.product.Chair;
import ont.data.modelobject.product.InvoiceSoftware;
import ont.data.modelobject.product.Wood;

public abstract class AModelStorage
    extends AObjectFactory
    implements IModelStorage
{


    public AModelStorage(IRUUIDProvider ruuidProvider) {
        super(ruuidProvider);
    }

    public IModelSet<Customer> getCustomers() {
        return getMAO(Customer.class).getLatestObjects();
    }

    public IModelSet<Customer> getCustomers(StorageVersion version) {
        return getMAO(Customer.class).getObjects(version);
    }

    public IModelSet<Invoice> getInvoices() {
        return getMAO(Invoice.class).getLatestObjects();
    }

    public IModelSet<Invoice> getInvoices(StorageVersion version) {
        return getMAO(Invoice.class).getObjects(version);
    }

    @SuppressWarnings("unchecked")
    public IModelSet<AFurniture<?>> getAFurnitures() {
        return ((IModelSet<AFurniture<?>> )((Object)
           getMAO(AFurniture.class).getLatestObjects()));
    }

    @SuppressWarnings("unchecked")
    public IModelSet<AFurniture<?>> getAFurnitures(StorageVersion
           version) {
        return ((IModelSet<AFurniture<?>> )((Object)
           getMAO(AFurniture.class).getObjects(version)));
    }

    @SuppressWarnings("unchecked")
    public IModelSet<AMaterial<?>> getAMaterials() {
        return ((IModelSet<AMaterial<?>> )((Object)
           getMAO(AMaterial.class).getLatestObjects()));
    }
```

```
    @SuppressWarnings("unchecked")
    public IModelSet<AMaterial<?>> getAMaterials(StorageVersion
            version) {
        return ((IModelSet<AMaterial<?>> )((Object)
            getMAO(AMaterial.class).getObjects(version)));
    }

    @SuppressWarnings("unchecked")
    public IModelSet<AProduct<?>> getAProducts() {
        return ((IModelSet<AProduct<?>> )((Object)
            getMAO(AProduct.class).getLatestObjects()));
    }

    @SuppressWarnings("unchecked")
    public IModelSet<AProduct<?>>getAProducts(StorageVersion version){
        return ((IModelSet<AProduct<?>> )((Object)
            getMAO(AProduct.class).getObjects(version)));
    }

    @SuppressWarnings("unchecked")
    public IModelSet<ASoftwareProduct<?>> getASoftwareProducts() {
        return ((IModelSet<ASoftwareProduct<?>> )((Object)
            getMAO(ASoftwareProduct.class).getLatestObjects()));
    }

    @SuppressWarnings("unchecked")
    public IModelSet<ASoftwareProduct<?>>
            getASoftwareProducts(StorageVersion version) {
        return ((IModelSet<ASoftwareProduct<?>> )((Object)
            getMAO(ASoftwareProduct.class).getObjects(version)));
    }

    public IModelSet<Chair> getChairs() {
        return getMAO(Chair.class).getLatestObjects();
    }

    public IModelSet<Chair> getChairs(StorageVersion version) {
        return getMAO(Chair.class).getObjects(version);
    }

    public IModelSet<InvoiceSoftware> getInvoiceSoftwares() {
        return getMAO(InvoiceSoftware.class).getLatestObjects();
    }

    public IModelSet<InvoiceSoftware>
            getInvoiceSoftwares(StorageVersion version) {
        return getMAO(InvoiceSoftware.class).getObjects(version);
    }

    public IModelSet<Wood> getWoods() {
        return getMAO(Wood.class).getLatestObjects();
    }

    public IModelSet<Wood> getWoods(StorageVersion version) {
        return getMAO(Wood.class).getObjects(version);
    }

}
```

```
package ont.model;

import java.util.List;
import ont.data.generated.dataobject.customer.CustomerData;
import ont.data.generated.dataobject.customer.InvoiceData;
import ont.data.generated.dataobject.product.ChairData;
import ont.data.generated.dataobject.product.InvoiceSoftwareData;
import ont.data.generated.dataobject.product.WoodData;
import ont.data.modelobject.customer.Customer;
import ont.data.modelobject.product.AMaterial;
import ont.data.modelobject.product.AProduct;
import ont.data.modelobject.product.Chair;
import ont.data.modelobject.product.InvoiceSoftware;
import ont.data.modelobject.product.Wood;

public abstract class AObjectFactory
    implements IObjectFactory
{

    private final IRUUIDProvider ruuidProvider;

    static {
        ModelObjectMapping.registerMapping(Customer.class,
           CustomerData.class);
        ModelObjectMapping.registerMapping(
           ont.data.generated.modelobject.customer.AGeneratedCustomer.
           Invoice.class,ont.data.generated.dataobject.customer.
           CustomerData.Invoice.class);
        ModelObjectMapping.registerMapping(ont.data.modelobject.
           customer. Invoice.class, InvoiceData.class);
        ModelObjectMapping.registerMapping(
           ont.data.generated.modelobject.customer.AGeneratedInvoice.
           Products.class,
           ont.data.generated.dataobject.customer.InvoiceData.Products
           .class);
        ModelObjectMapping.registerMapping(Chair.class,
           ChairData.class);
        ModelObjectMapping.registerMapping(InvoiceSoftware.class,
           InvoiceSoftwareData.class);
        ModelObjectMapping.registerMapping(Wood.class,
           WoodData.class);
    }

    public AObjectFactory(IRUUIDProvider ruuidProvider) {
        this.ruuidProvider = ruuidProvider;
    }

    public abstract ILowLevelModelStorage getLowLevelModelStorage();

    private List<RUUID> getIds(Iterable<? extends IUnique> uniques) {
        return F.map(uniques, new Function<IUnique,RUUID>() {


            @Override
            public RUUID apply(IUnique param) {
                return param.getId();
            }

        }
        );
    }
```

```
private Option<RUUID> getOptionId(Option<? extends IUnique>
       unique) {
    if (unique.isDefined()) {
        return Option.<RUUID>some(unique.get().getId());
    }
    return Option.<RUUID>none();
}

public final Customer createCustomer(String name,
    @CompositeItems
    List<ont.data.modelobject.customer.Invoice> invoice) {
    ont.data.generated.dataobject.customer.CustomerData.Invoice
       invoiceData =
       new ont.data.generated.dataobject.customer.
                  CustomerData.Invoice(ruuidProvider.getNewRUUID(),
                  ObjectVersion.getNew(), SystemTime.dateTime(),
                  getIds(invoice));

    Tx.store(invoiceData);

    ont.data.generated.modelobject.customer.AGeneratedCustomer.
       Invoice invoiceModel = new ont.data.generated.modelobject.
                  customer.AGeneratedCustomer.
                  Invoice(invoiceData.getId(),
                  StorageVersion.getNew(),
                  getLowLevelModelStorage(),
                  invoiceData);

    Tx.store(invoiceModel);

    CustomerData data = new  CustomerData(
                  ruuidProvider.getNewRUUID(),
                  ObjectVersion.getNew(),
                  SystemTime.dateTime(), name,
                  invoiceData.getId());

    Tx.store(data);
    Customer model = new Customer(data.getId(),
       StorageVersion.getNew(), getLowLevelModelStorage(), data);

    model.add();
    return model;
}

public final ont.data.modelobject.customer.Invoice
       createInvoice(Customer customer,
    @CompositeItems
    List<AProduct<?>> products) {
    ont.data.generated.dataobject.customer.InvoiceData.Products
       productsData = new ont.data.generated.dataobject.customer.
                  InvoiceData.Products(ruuidProvider.getNewRUUID(),
                  ObjectVersion.getNew(), SystemTime.dateTime(),
                  getIds(products));

    Tx.store(productsData);
    ont.data.generated.modelobject.customer.
       AGeneratedInvoice.Products productsModel = new ont.
                  data.generated.modelobject.customer.
                  AGeneratedInvoice.Products(productsData.getId(),
                  StorageVersion.getNew(),
                  getLowLevelModelStorage(), productsData);
    Tx.store(productsModel);
```

```
InvoiceData data = new InvoiceData(
                      ruuidProvider.getNewRUUID(),
                      ObjectVersion.getNew(), SystemTime.dateTime(),
                      customer.getId(), productsData.getId());

        Tx.store(data);
        ont.data.modelobject.customer.Invoice model = new
           ont.data.modelobject.customer.Invoice(data.getId(),
           StorageVersion.getNew(), getLowLevelModelStorage(), data);
        model.add();
        return model;
    }

    public final Chair createChair(double price, AMaterial<?>
material) {
        ChairData data = new ChairData(ruuidProvider.getNewRUUID(),
                      ObjectVersion.getNew(), SystemTime.dateTime(),
                      price, material.getId());
        Tx.store(data);
        Chair model = new Chair(data.getId(), StorageVersion.getNew(),
                      getLowLevelModelStorage(), data);
        model.add();
        return model;
    }

    public final InvoiceSoftware createInvoiceSoftware(String
platform) {
        InvoiceSoftwareData data = new      InvoiceSoftwareData(
                      ruuidProvider.getNewRUUID(),
                      ObjectVersion.getNew(), SystemTime.dateTime(),
                      platform);
        Tx.store(data);
        InvoiceSoftware model = new InvoiceSoftware(
                      data.getId(), StorageVersion.getNew(),
                      getLowLevelModelStorage(), data);
        model.add();
        return model;
    }

    public final Wood createWood(String name) {
        WoodData data = new WoodData(ruuidProvider.getNewRUUID(),
                      ObjectVersion.getNew(), SystemTime.dateTime(),
                      name);
        Tx.store(data);
        Wood model = new Wood(data.getId(), StorageVersion.getNew(),
                      getLowLevelModelStorage(), data);
        model.add();
        return model;
    }

}
```

```java
package ont.model;

import ont.data.modelobject.customer.Customer;
import ont.data.modelobject.customer.Invoice;
import ont.data.modelobject.product.AFurniture;
import ont.data.modelobject.product.AMaterial;
import ont.data.modelobject.product.AProduct;
import ont.data.modelobject.product.ASoftwareProduct;
import ont.data.modelobject.product.Chair;
import ont.data.modelobject.product.InvoiceSoftware;
import ont.data.modelobject.product.Wood;

public interface IGeneratedModelStorage {


    public IModelSet<Customer> getCustomers();

    public IModelSet<Customer> getCustomers(StorageVersion version);

    public IModelSet<Invoice> getInvoices();

    public IModelSet<Invoice> getInvoices(StorageVersion version);

    public IModelSet<AFurniture<?>> getAFurnitures();

    public IModelSet<AFurniture<?>> getAFurnitures(StorageVersion
           version);

    public IModelSet<AMaterial<?>> getAMaterials();

    public IModelSet<AMaterial<?>> getAMaterials(StorageVersion
           version);

    public IModelSet<AProduct<?>> getAProducts();

    public IModelSet<AProduct<?>> getAProducts(StorageVersion
           version);

    public IModelSet<ASoftwareProduct<?>> getASoftwareProducts();

    public IModelSet<ASoftwareProduct<?>>
           getASoftwareProducts(StorageVersion version);

    public IModelSet<Chair> getChairs();

    public IModelSet<Chair> getChairs(StorageVersion version);

    public IModelSet<InvoiceSoftware> getInvoiceSoftwares();

    public IModelSet<InvoiceSoftware>
           getInvoiceSoftwares(StorageVersion version);

    public IModelSet<Wood> getWoods();

    public IModelSet<Wood> getWoods(StorageVersion version);

}
```

```
package ont.model;

import java.util.List;
import ont.data.modelobject.customer.Customer;
import ont.data.modelobject.customer.Invoice;
import ont.data.modelobject.product.AMaterial;
import ont.data.modelobject.product.AProduct;
import ont.data.modelobject.product.Chair;
import ont.data.modelobject.product.InvoiceSoftware;
import ont.data.modelobject.product.Wood;

public interface IObjectFactory {


    public Customer createCustomer(String name,
        @CompositeItems
        List<Invoice> invoice);

    public Invoice createInvoice(Customer customer,
        @CompositeItems
        List<AProduct<?>> products);

    public Chair createChair(double price, AMaterial<?> material);

    public InvoiceSoftware createInvoiceSoftware(String platform);

    public Wood createWood(String name);

}

package ont.model;

import ont.data.generated.dataobject.customer.CustomerData;
import ont.data.generated.dataobject.customer.InvoiceData;
import ont.data.generated.dataobject.product.ChairData;
import ont.data.generated.dataobject.product.InvoiceSoftwareData;
import ont.data.generated.dataobject.product.WoodData;

public final class GeneratedDataReferenceMapping {


    protected static void init() {
        DataReferenceMapping.registerRemovePolicy(CustomerData.class,
           new OnRemoveNoAction());
        DataReferenceMapping.registerReference(new
           CollectionRef(ont.data.generated.dataobject.customer.
           CustomerData.Invoice.class), InvoiceData.class);

        DataReferenceMapping.registerRemovePolicy(ont.data.
           generated.dataobject.customer.CustomerData.Invoice.class,
           new OnRemoveNoAction());
        DataReferenceMapping.registerRemovePolicy(InvoiceData.class,
           new OnRemoveNoAction());
        DataReferenceMapping.registerReference(new
           FieldRef(InvoiceData.class, "getCustomer"),
           CustomerData.class);
        DataReferenceMapping.registerReference(new
           CollectionRef(ont.data.generated.dataobject.customer.
           InvoiceData.Products.class), ChairData.class);
        DataReferenceMapping.registerRemovePolicy(ont.data.generated.
           dataobject.customer.InvoiceData.Products.class, new
           OnRemoveNoAction());
```

```
 DataReferenceMapping.registerReference(new FieldRef(ChairData.class,
           "getMaterial"), WoodData.class);
        DataReferenceMapping.registerRemovePolicy(ChairData.class, new
           OnRemoveNoAction());

DataReferenceMapping.registerRemovePolicy(InvoiceSoftwareData.class,
           new OnRemoveNoAction());
        DataReferenceMapping.registerRemovePolicy(WoodData.class, new
           OnRemoveNoAction());
    }

}
```

Esimerkkinä generoidut Customer ja Invoice tiedostot:

```
package ont.data.generated.dataobject.customer;

import java.util.List;
import org.joda.time.DateTime;

public class CustomerData
    extends ADataObject
{

    private final static long serialVersionUID = 1L;
    private final String name;
    private final RUUID invoice;

    public CustomerData(RUUID id, ObjectVersion objectVersion,
           DateTime timestamp, String name, RUUID invoice) {
        super(id, objectVersion, timestamp);
        this.name = name;
        this.invoice = invoice;
    }

    public String getName() {
        return name;
    }

    public RUUID getInvoice() {
        return invoice;
    }

    public CustomerData setName(String name) {
        return new CustomerData(getId(), getObjectVersion(),
           getTimestamp(), name, getInvoice());
    }

    @Override
    public CustomerData setObjectVersion(ObjectVersion objectVersion)
{
        return new CustomerData(getId(), objectVersion,
           getTimestamp(), getName(), getInvoice());
    }

    @Override
    public CustomerData setTimestamp(DateTime timestamp) {
        return new CustomerData(getId(), getObjectVersion(),
           timestamp, getName(), getInvoice());
    }

    public final static class Invoice
        extends DList
    {

        private final static long serialVersionUID = 1L;

        public Invoice(RUUID id, ObjectVersion objectVersion, DateTime
           timestamp, List<RUUID> data) {
            super(id, objectVersion, timestamp, data);
        }
```

```
        public CustomerData.Invoice copy(RUUID id, ObjectVersion
           objectVersion, DateTime timestamp, List<RUUID> data) {
            return new Invoice(id, objectVersion, timestamp, data);
        }

        @Override
        public Class<? extends IModelObject<?>> getItemType() {
            return (Class<? extends IModelObject<?>>) (Object)
           ont.data.modelobject.customer.Invoice.class;
        }

        @Override
        public boolean isCompositeItems() {
            return true;
        }

    }

}

package ont.data.generated.dataobject.customer;

import java.util.List;
import org.joda.time.DateTime;

public class InvoiceData
    extends ADataObject
{

    private final static long serialVersionUID = 1L;
    private final RUUID customer;
    private final RUUID products;

    public InvoiceData(RUUID id, ObjectVersion objectVersion, DateTime
           timestamp, RUUID customer, RUUID products) {
        super(id, objectVersion, timestamp);
        this.customer = customer;
        this.products = products;
    }

    public RUUID getCustomer() {
        return customer;
    }

    public RUUID getProducts() {
        return products;
    }

    @Override
    public InvoiceData setObjectVersion(ObjectVersion objectVersion) {
        return new InvoiceData(getId(), objectVersion, getTimestamp(),
           getCustomer(), getProducts());
    }

    @Override
    public InvoiceData setTimestamp(DateTime timestamp) {
        return new InvoiceData(getId(), getObjectVersion(), timestamp,
           getCustomer(), getProducts());
    }
```

```
public final static class Products
        extends DList
    {
        private final static long serialVersionUID = 1L;

        public Products(RUUID id, ObjectVersion objectVersion,
           DateTime timestamp, List<RUUID> data) {
            super(id, objectVersion, timestamp, data);
        }

        public InvoiceData.Products copy(RUUID id, ObjectVersion
           objectVersion, DateTime timestamp, List<RUUID> data) {
            return new Products(id, objectVersion, timestamp, data);
        }

        @Override
        public Class<? extends IModelObject<?>> getItemType() {
            return (Class<? extends IModelObject<?>>) (Object)
           ont.data.modelobject.product.AProduct.class;
        }

        @Override
        public boolean isCompositeItems() {
            return false;
        }

    }

}

package ont.data.generated.modelobject.customer;

import ont.data.modelobject.customer.Customer;
import ont.data.modelobject.product.AProduct;

public interface IGeneratedInvoice<MO extends IGeneratedInvoice<?> >
    extends IModelObject<MO>
{


    public Customer getCustomer();

    public MList<AProduct<?>> getProducts();

}


package ont.data.generated.modelobject.customer;

import ont.data.modelobject.customer.Invoice;

public interface IGeneratedCustomer<MO extends IGeneratedCustomer<?> >
    extends IModelObject<MO>
{

    public String getName();

    public void setName(String name);

    public MList<Invoice> getInvoice();

}
```

```
package ont.data.generated.modelobject.customer;

import ont.data.generated.dataobject.customer.InvoiceData;
import ont.data.modelobject.customer.Customer;
import ont.data.modelobject.product.AProduct;

public abstract class AGeneratedInvoice<MO extends
AGeneratedInvoice<?> >
    extends AModelObject<MO>
    implements IGeneratedInvoice<MO>
{

    public final static String CUSTOMER = "customer";
    public final static String PRODUCTS = "products";
    private final ModelRef<Customer> customer;
    private final
           ModelRef<ont.data.modelobject.customer.Invoice.Products>
           products;

    public AGeneratedInvoice(RUUID id, StorageVersion version,
           ILowLevelModelStorage storage, InvoiceData data) {
        super(id, version, storage, data);
        customer = new ModelRef<Customer>(version,
           getLowLevelModelStorage(), Customer.class);
        products = new
           ModelRef<ont.data.modelobject.customer.Invoice.Products>(
           version, getLowLevelModelStorage(),
           ont.data.modelobject.customer.Invoice.Products.class);
    }

    @Dependant
    public Customer getCustomer() {
        return        this.customer.get(this.<InvoiceData>getData().
           getCustomer());
    }

    public MList<AProduct<?>> getProducts() {
        return        this.products.get(this.<InvoiceData>getData().
           getProducts());
    }

    @Override
    protected void addReferencedObjects() {
        this.customer.add(this.<InvoiceData>getData().getCustomer());
        this.products.add(this.<InvoiceData>getData().getProducts());
    }

    @Override
    protected void removeCompositeObjects() {
           this.products.remove(this.<InvoiceData>getData().
           getProducts());
    }

    @Override
    protected Option<IModelObject<?>> getDependant() {
        return Option.<IModelObject<? extends
           Object>>some(getCustomer());
    }
```

```
public final static class Products
        extends MList<AProduct<?>>
    {

        public Products(RUUID id, StorageVersion version,
           ILowLevelModelStorage storage,
           ont.data.generated.dataobject.customer.InvoiceData.Products
           data) {
            super(id, version, storage, data);
        }

    }

}


package ont.data.generated.modelobject.customer;

import ont.data.generated.dataobject.customer.CustomerData;

public abstract class AGeneratedCustomer<MO extends
AGeneratedCustomer<?> >
    extends AModelObject<MO>
    implements IGeneratedCustomer<MO>
{

    public final static String NAME = "name";
    public final static String INVOICE = "invoice";
    private final
           ModelRef<ont.data.modelobject.customer.Customer.Invoice>
           invoice;

    public AGeneratedCustomer(RUUID id, StorageVersion version,
           ILowLevelModelStorage storage, CustomerData data) {
        super(id, version, storage, data);
        invoice = new
           ModelRef<ont.data.modelobject.customer.Customer.Invoice>(
           version, getLowLevelModelStorage(),
           ont.data.modelobject.customer.Customer.Invoice.class);
    }

    public String getName() {
        return this.<CustomerData>getData().getName();
    }

    public MList<ont.data.modelobject.customer.Invoice> getInvoice() {
        return         this.invoice.get(this.<CustomerData>getData().
           getInvoice());
    }

    public void setName(final String name) {
           saveToTransaction((AGeneratedCustomer.this).<CustomerData>
           getData().setName(name));
    }

    @Override
    protected void addReferencedObjects() {
        this.invoice.add(this.<CustomerData>getData().getInvoice());
    }
```

```
@Override
    protected void removeCompositeObjects() {
           this.invoice.remove(this.<CustomerData>getData().
           getInvoice());
    }

    @Override
    protected Option<IModelObject<?>> getDependant() {
        return Option.none();
    }

    public final static class Invoice
        extends MList<ont.data.modelobject.customer.Invoice>
    {


        public Invoice(RUUID id, StorageVersion version,
           ILowLevelModelStorage storage,
           ont.data.generated.dataobject.customer.CustomerData.
           Invoice data) {
            super(id, version, storage, data);
        }

    }

}


package ont.data.modelobject.customer;

import ont.data.generated.modelobject.customer.IGeneratedCustomer;

public interface ICustomer<MO extends ICustomer<?> >
    extends IGeneratedCustomer<MO>
{


}


package ont.data.modelobject.customer;

import ont.data.generated.modelobject.customer.IGeneratedInvoice;

public interface IInvoice<MO extends IInvoice<?> >
    extends IGeneratedInvoice<MO>
{


}
```

```
package ont.data.modelobject.customer;

import ont.data.generated.dataobject.customer.InvoiceData;
import ont.data.generated.modelobject.customer.AGeneratedInvoice;

public final class Invoice
    extends AGeneratedInvoice<Invoice>
    implements IInvoice<Invoice>
{

    public Invoice(RUUID id, StorageVersion version,
           ILowLevelModelStorage storage, InvoiceData data) {
        super(id, version, storage, data);
    }

}

package ont.data.modelobject.customer;

import ont.data.generated.dataobject.customer.CustomerData;
import ont.data.generated.modelobject.customer.AGeneratedCustomer;

public final class Customer
    extends AGeneratedCustomer<Customer>
    implements ICustomer<Customer>
{


    public Customer(RUUID id, StorageVersion version,
           ILowLevelModelStorage storage, CustomerData data) {
        super(id, version, storage, data);
    }

}
```