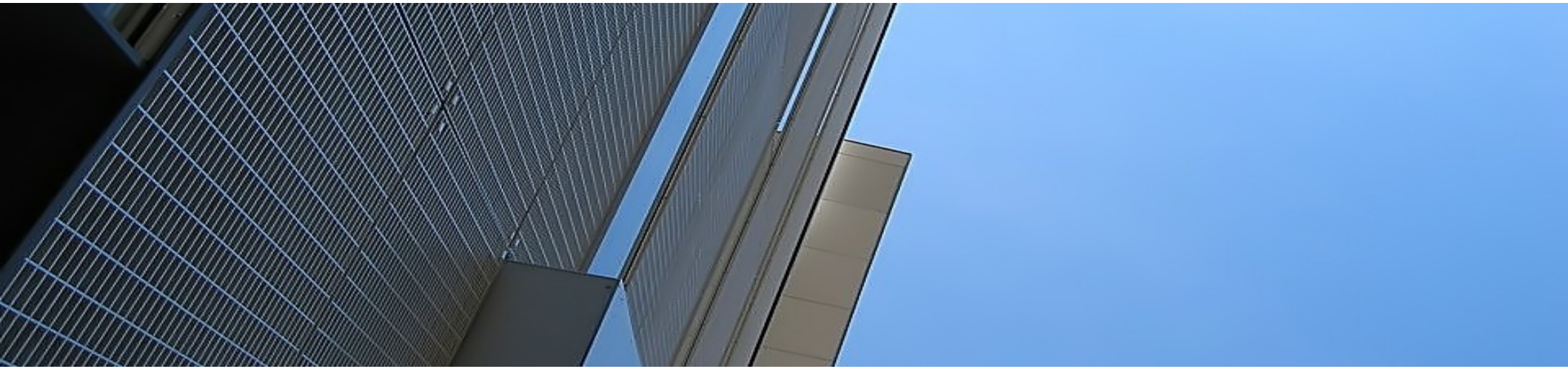
CINCO

Collaborative and interoperable computing



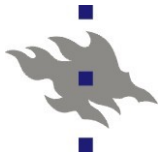
Palvelusuuntautunut ohjelmistotuotanto

Luento 7: Mallimuunnokset

Toni Ruokolainen, 09.02.2010

Luennon runko

- Mallimuunnosten perusteet
- Mallista malliin -muunnokset
- Mallista tekstiin -muunnokset
- Mallien kudonta
- Esimerkkejä



Mallimuunnosten roolista

- Mallimuunnos on prosessi, jossa joukko malleja muunnetaan toisiksi malleiksi
  - Prosessin toimintojen sisältö on annettu muunnosmäärittelyssä
  - Muunnosmäärittelyt koostuvat joukosta muunnossääntöjä
  - Muunnosmäärittelyt mallinnetaan jollain muunnoskielellä
  - Muunnosmäärittely suoritetaan muunnoskehyksen ominaisuuksien mukaisesti
- Muunnosmäärittelyt ovat itsessään malleja
  - Kuvaavat mallien välisiä riippuvuuksia
  - Dokumentoivat toteuttamistapoja
- Mallimuunnokset mahdollistavat
  - Abstraktiotasojen ja näkökulmien väliset siirtymät
  - Ohjelmistoartefaktien tuottamisen malleista
  - Mallien uudelleenkäytettävyyden ja modularisoinnin
    - Erityisesti mallien kutominen

Mallimuunnokset

- Mallista malliin -muunnokset
  - Syöte: 1-n mallia
  - Tulos: 1-n uutta mallia
  - Yleensä 1-1 muunnoksia
- Mallista tekstiin -muunnokset
  - Syöte: 1-n mallia
  - Tulos: ohjelmistoartefakti
    - Esim. WSDL-kuvaus, Java-luokka, konfigurointitiedosto..
- Mallien kudonta
  - Määrittelee vastaavuuksia mallien välille
  - Voidaan käyttää esimerkiksi ei-toiminnallisten ja toiminnallisten piirteiden välisten suhteiden määrittelemiseen

Mallimuunnosten luokittelua

- Mallimuunnoksia voidaan luokitella
  - Lähtö- ja kohdemallien kielen perusteella
    - Endogeeninen muunnos: saman kielen sisällä
    - Eksogeeninen muunnos: eri kielten välillä
  - Lähtö- ja kohdemallien abstraktiotason perusteella
    - Horisontaalinen: saman abstraktiotason sisällä
    - Vertikaalinen: eri abstraktiotasojen välillä
  - Toteutustavan perusteella
    - Imperatiivinen: määrittele miten muunnos tehdään
    - Deklaratiivinen: määrittelee mitä muunnoksessa tehdään

	Horisontaalinen	Vertikaalinen
Endogeeninen	Refaktorointi	Rikastaminen
Eksogeeninen	Kielimigraatio	Koodin generointi

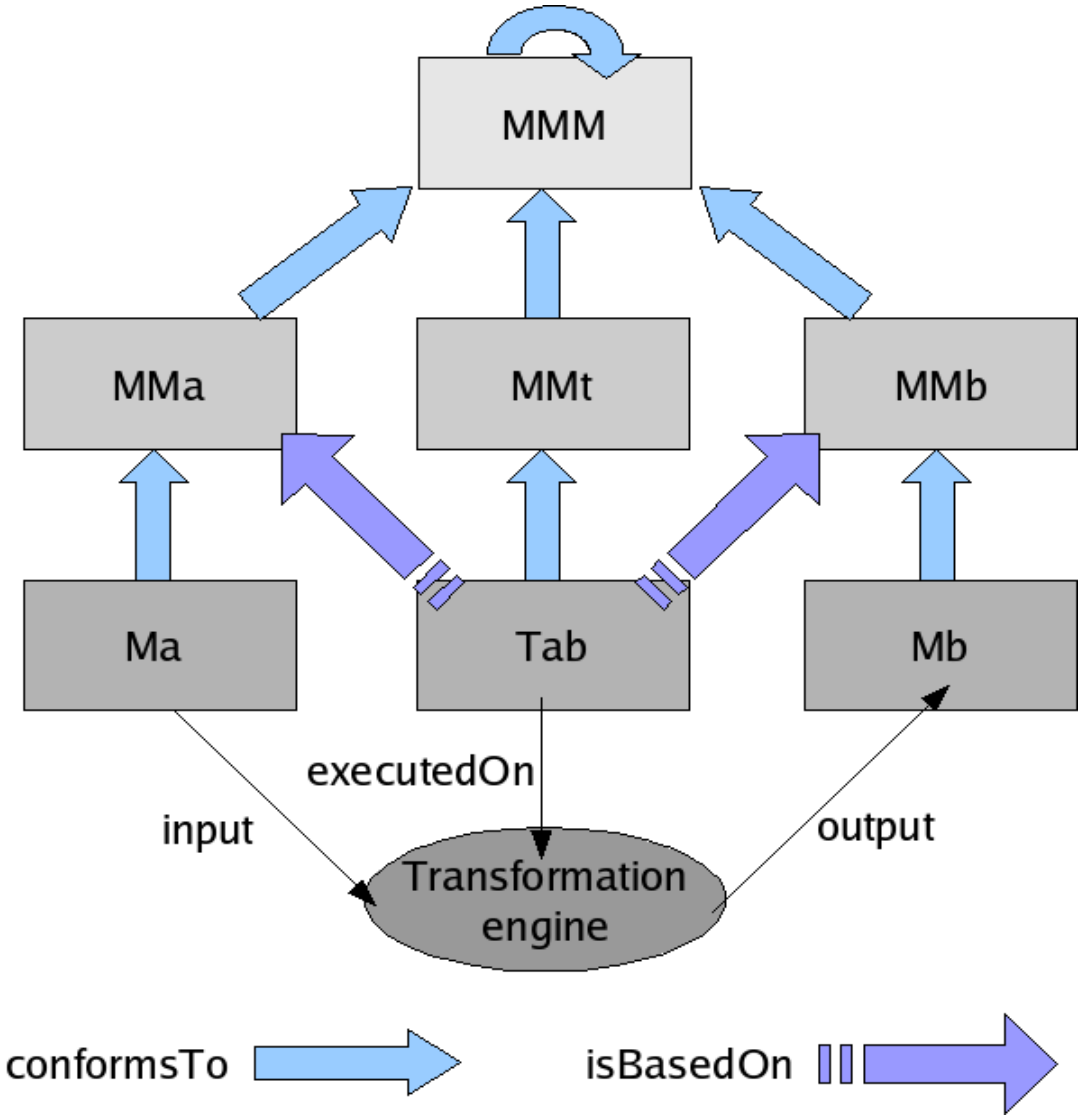
Mallimuunnosten luokittelua esimerkkeineen.

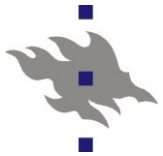
Mallimuunnosten ominaisuuksia

- Muunnosmäärittelyn muoto
  - Onko määrittely suoraan suoritettavassa muodossa vai ei?
- Muunnossääntöjen muoto
  - Miten säännöt määritellään? Esim. kaavaimet tai funktionaaliset kielet
- Sääntöjen soveltamisen kontrolloiminen
  - Vaikutusalueen määrittely, skedulointi
- Sääntöjen organisointi
  - Sääntöjen ja muunnosten modularisointi; uudelleenkäyttö
- Lähde- ja kohdemallien välinen suhde
  - Muutetaanko lähdemallia vai luodaanko uusi malli?
- Inkrementaalisuus
  - Voidaanko kohdemalleja päivittää lähdemallissa tapahtuneiden muunnosten perusteella?
- Muunnosten suunnattavuus
  - Yksi- vai kaksisuuntaisia muunnoksia?
- Muunnosten jäljitettävyys

Mallista malliin -muunnokset
(model to model -transformation, M2M)

- Erilaisia lähestymistapoja
  - Deklaratiivinen, proseduraalinen, verkkoperustainen,...
- Muunnoskieliä:
  - QVT, ATL, Xtend,...





Suoran manipuloinnin ja strukturaalisen lähestymistavan mallimuunnokset

- Suora manipulointi
  - Mallin käsittelyyn on käytettävissä API
  - Mallimuunnos suoritetaan jollain yleiskäyttöisellä ohjelmointikielella
  - Esimerkiksi: käytetään Eclipse Modeling Frameworkin metamallikohtaisia Java-luokkia ja koodataan muunnos Java-kielellä
- Strukturaalinen lähestymistapa
  - Kaksivaiheinen kohdemallin tuottaminen
    - Ensin tuotetaan kohdemallin hierarkinen rakenne
    - Toisessa vaiheessa laitetaan kohdemallin attribuutit ja assosiaatiot paikalleen
  - Perustuu geneeriseen ohjelmointikieleen perustuvaan mallimuunnoskehykseen sekä mallielementtien kopioimiseen lähteestä kohteeseen
    - Käyttäjä toteuttaa elementtikohtaiset kopiointisäännöt; muunnoskehys pitää huolen mm. sääntöjen skeduloinnista ja suorituksesta

Operationaaliset mallimuunnokset

- Operationaalinen tapa
  - Kuin strukturaalinen lähestymistapa, mutta käytössä on paremmin muunnossääntöjen määrittelemiseen käytettävä kieli
    - Esimerkiksi OCL-pohjainen kieli + imperatiivisia mallimuunnosoperaatioita
  - Esimerkiksi: QVT Operational mappings (kuvassa), Kermeta

```
metamodel BOOK {
  class Book {title: String; composes chapters: Chapter [*];}
  class Chapter {title : String; nbPages : Integer;}
}

metamodel PUB {
  class Publication {title : String; nbPages : Integer;}
}

transformation Book2Publication(in bookModel:BOOK,out pubModel:PUB);

main() {
  bookModel->objectsOfType(Book)->map book_to_publication();
}

mapping Class::book_to_publication () : Publication {
  title := self.title;
  nbPages := self.chapters->nbPages->sum();
}
```

Kaavainpohjaiset ja relationaaliset mallimuunnokset

- Kaavainpohjainen lähestymistapa
  - Hyödynnetään mallikaavaimia, joihin sisällytetty metakoodia
  - Metakoodi evaluoi kyseiseen paikkaan oikean sisällön
  - Yleensä mallikaavaimet ilmaistaan kohdemallin konkreettisella syntaksilla
- Relationaalinen lähestymistapa
  - Deklaratiivinen mallimuunnostapa, joka perustuu matemaattisiin relaatioihin
  - Määritellään relaatioita lähde- ja kohdemallien välille
  - Ovat luonnostaan kaksisuuntaisia muunnoksia
  - Esimerkiksi QVT Relations (kuvassa) ja AMW

```
relation ClassToTable   /* map each persistent class to a table */
  {
    domain uml c:Class {
         namespace = p:Package {},
         kind='Persistent',
         name=cn
     }
    domain rdbms t:Table {
          schema = s:Schema {},
           name=cn,
           column = cl:Column {
                name=cn+'_tid',
                type='NUMBER'},
                     primaryKey = k:PrimaryKey {
                 name=cn+'_pk',
                 column=cl}
     }
    when {
      PackageToSchema(p, s);
    }
    where {
      AttributeToColumn(c, t);
    }
  }
```

Verkkomuunnospohjaiset mallimuunnokset

- Verkkomuunnospohjainen lähestymistapa
  - Hyödyntää verkkoteoriaa
    - Verkon kaaret ja solmut ovat tyypitetty ja nimetty sekä voivat sisältää attribuutteja
  - Lähde- ja kohdemallin välinen muunnossääntö määritellään verkkokaavaimien avulla
    - Verkkokaavaimet voidaan antaa myös lähde- ja kohdemallien konkreettisia syntakseja käyttäen
  - Esimerkkejä: VIATRA, AGG (kuvassa), GReAT

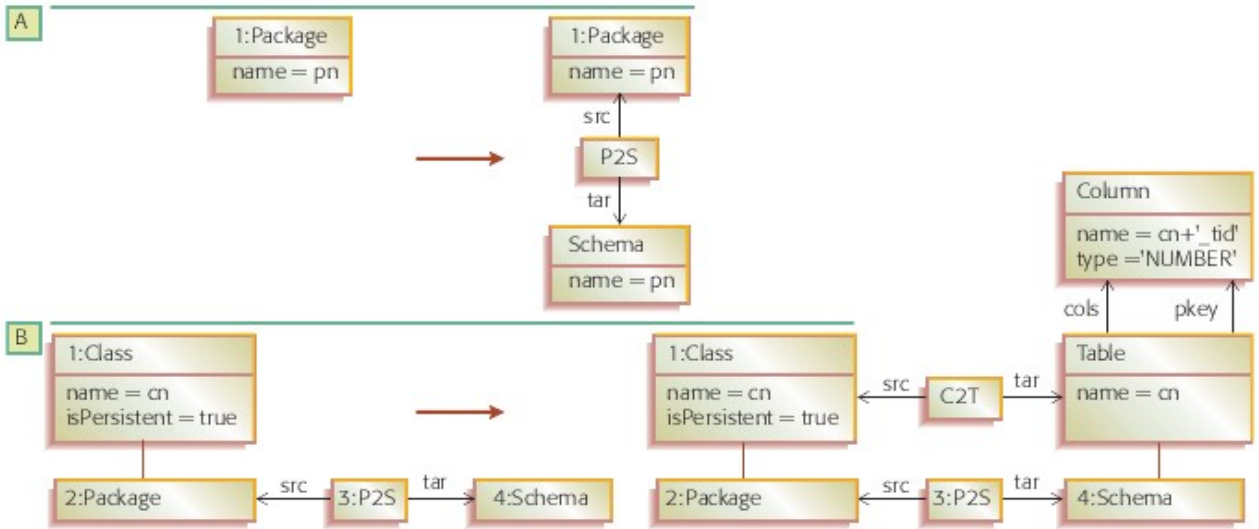


Figure 10
Graph transformation in AGG: (A) package-to-schema rule and (B) class-to-table rule

Hybridimallimuunnokset

- Hybridit lähestymistavat hyödyntävät edellisten lähestymistapojen komponentteja
- Esimerkkejä: ATL

```
-- @path Families=/Families2Persons/Families.ecore
-- @path Persons=/Families2Persons/Persons.ecore
module Families2Persons;
create OUT : Persons from IN : Families;

helper context Families!Member def: familyName : String =
if not self.familyFather.oclIsUndefined() then
self.familyFather.lastName
else
if not self.familyMother.oclIsUndefined() then
self.familyMother.lastName
else
if not self.familySon.oclIsUndefined() then
self.familySon.lastName
else
self.familyDaughter.lastName
endif
endif
endif;

helper context Families!Member def: isFemale() : Boolean =
if not self.familyMother.oclIsUndefined() then
true
else
if not self.familyDaughter.oclIsUndefined() then
true
else
false
endif
endif;

rule Member2Male {
from
s : Families!Member (not s.isFemale())
to
t : Persons!Male (
fullName <- s.firstName + ' ' + s.familyName
)
```

Tauko..

Mallien kudonta *(model weaving)*

- Mallien kudonnassa määritellään deklaratiivisia yhteyksiä erilaisten mallien välille
- Hyödyntämiskohteita:
  - Mallien sulauttaminen (*model merging)*, muunnosten jäljittäminen *(traceability),* aspektimallien käyttö, työkalujen yhteentoimivuus,...
- Lähestymistapoja ja kieliä
  - Kudontalinkit
    - Määritellään linkkityyppejä, jotka sisältävät sovellusaluekohtaista semantiikkaa
    - AMW (Atlas Model Weaver)
  - Aspektiperustainen kudonta
    - Perustuu ”join-point” -määrittelyihin (vrt. AOP)
    - XWeave

Atlas Model Weaver (AMW)

- Perustuu geneeriseen ydinkudontametamalliin
  - Antaa peruskäsitteet ”linkkien” määrittelyyn erilaisten mallien välille
  - Linkit ovat tyypillisesti suuntaamattomia
- Periaatteet
  - Linkit määrittelevät sovellusaluekohtaista semantiikkaa
  - Varsinainen kudontametamalli (*weaving metamodel, WMM)* tuotetaan sovellusalueasiantuntijan toimesta *laajentamalla* ydinkudontometamallia
    - Metamallien laajennus (*extension*) on eräs metamallien välinen suhde, jota ei tällä kurssilla käsitellä
  - Kudontamalli tuotetaan manuaalisesti tai (semi)automaattisesti heuristiikkaa käyttäen
  - Kudontomallia voidaan käyttää jatkokäsittelyyn
    - Esimerkiksi nk. korkeamman kertaluokan muunnosten tuottamiseen (*higher-order transformations)*

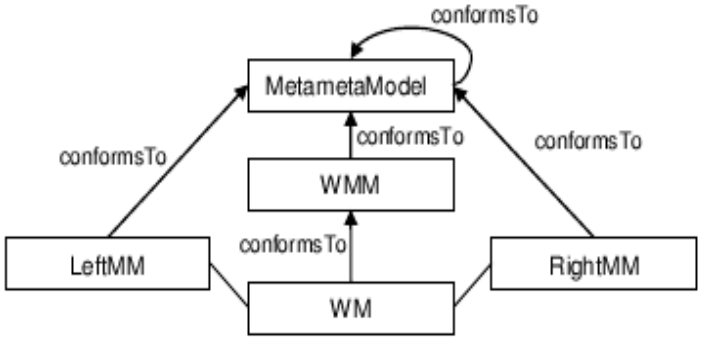


Fig. 4. Weaving conformance relations

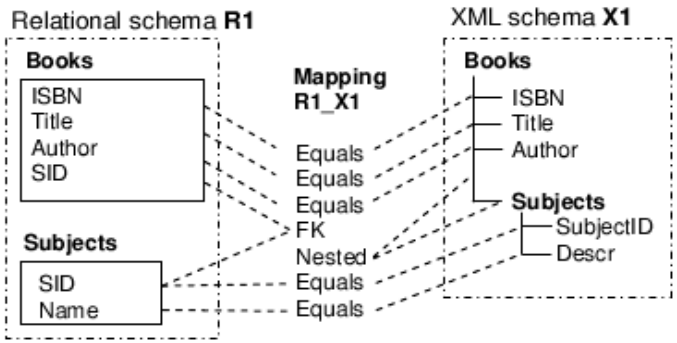


Fig. 3. Links between a relational and an XML schema

XWeave

- Mallien kudonta perustuu aspektimallien ja niihin liittyvien kudontasääntöjen määrittelyyn
  - Käytössä sekä nimi- että join point-perustaiset kudontasäännöt
- Hyödyntämiskohteita aspektipohjaiselle kudonnalle
  - Mallien evoluutio: mallien muutokset eristetään aspektimalleiksi
  - Produt Line Engineering (PLE): yhteisten ja varioivien tekijöiden eristäminen malleiksi ja aspektimalleiksi
  - Arkkitehtuuriset näkökulmat: näkökulmamallien yhdistäminen järjestelmän kokonaismalliksi

Figure 7: A Homogeneous Aspect

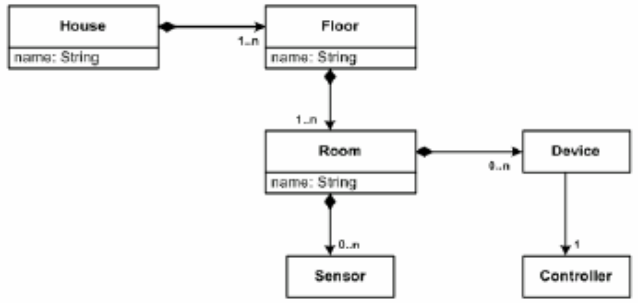


Figure 2: Smart Home Meta Model

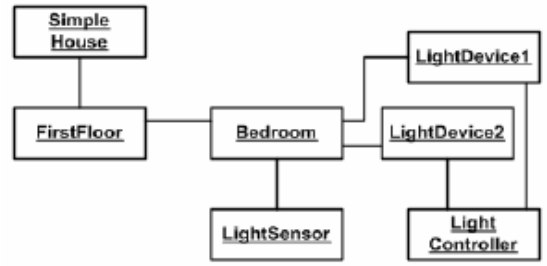


Figure 3: Smart Home Model

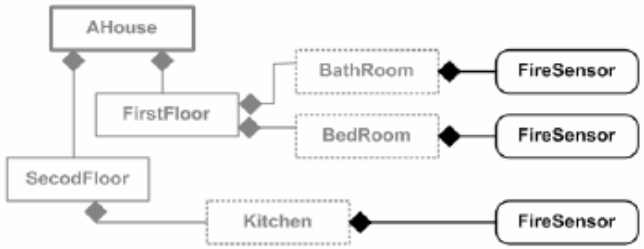
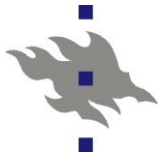


Figure 6: Resulting Model after Homogeneous Weaving



Mallista tekstiin -muunnokset *(model to text -transformation, M2T)*

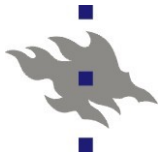
- Syöte: 1-n mallia
- Tuloste: 1-n tekstimuotoista tiedostoa
  - Esimerkiksi Java-lähdekoodia, XML-tiedosto, konfigurointitiedosto,...
- Erilaisia lähestymistapoja
  - Kaavaimet (*template),* ohjelmallinen muunnos, Visitor -suunnittelumalliin perustuvat
- Kieliä
  - JET (Java Emitter Templates), Xpand, MOFScript,...

Kaavainpohjaiset mallista-tekstiin muunnokset

- Suurin osa mallista-tekstiin -muunnostyökaluista on kaavainpohjaisia
- Tässä lähestymistavassa kohdekielen mukaiseen tekstiin upotetaan mallimuunnossääntöjä
- Esimerkkejä: OMG:n MOF Model To Text (Mof2Text), JET, MetaEdit+

```
[module generate(http://www.eclipse.org/uml2/2.1.0/UML)/]

[template public generate(c : Class)]
        [file (c.name.concat('.java'), false)]
        public class [c.name.toUpperFirst()/] {
        [for (p : Property | c.attribute)]
        private [p.type.name/] [p.name/];
        [/for]
        [for (p : Property | c.attribute)]
        public [p.type.name/] get[p.name.toUpperFirst()/]()
{
                return this.[p.name/];
        }
        [/for]
        [for (o : Operation | c.ownedOperation)]
        public [o.type.name/] [o.name/]() {
                // TODO should be implemented
        }
        [/for]
        }
        [/file]
[/template]
```

Esimerkki Acceleolla (Mof2Text) tehdystä M2T -muunnoksesta, joka luo jokaista UML-mallin luokkaa vastaavan Java-luokan.



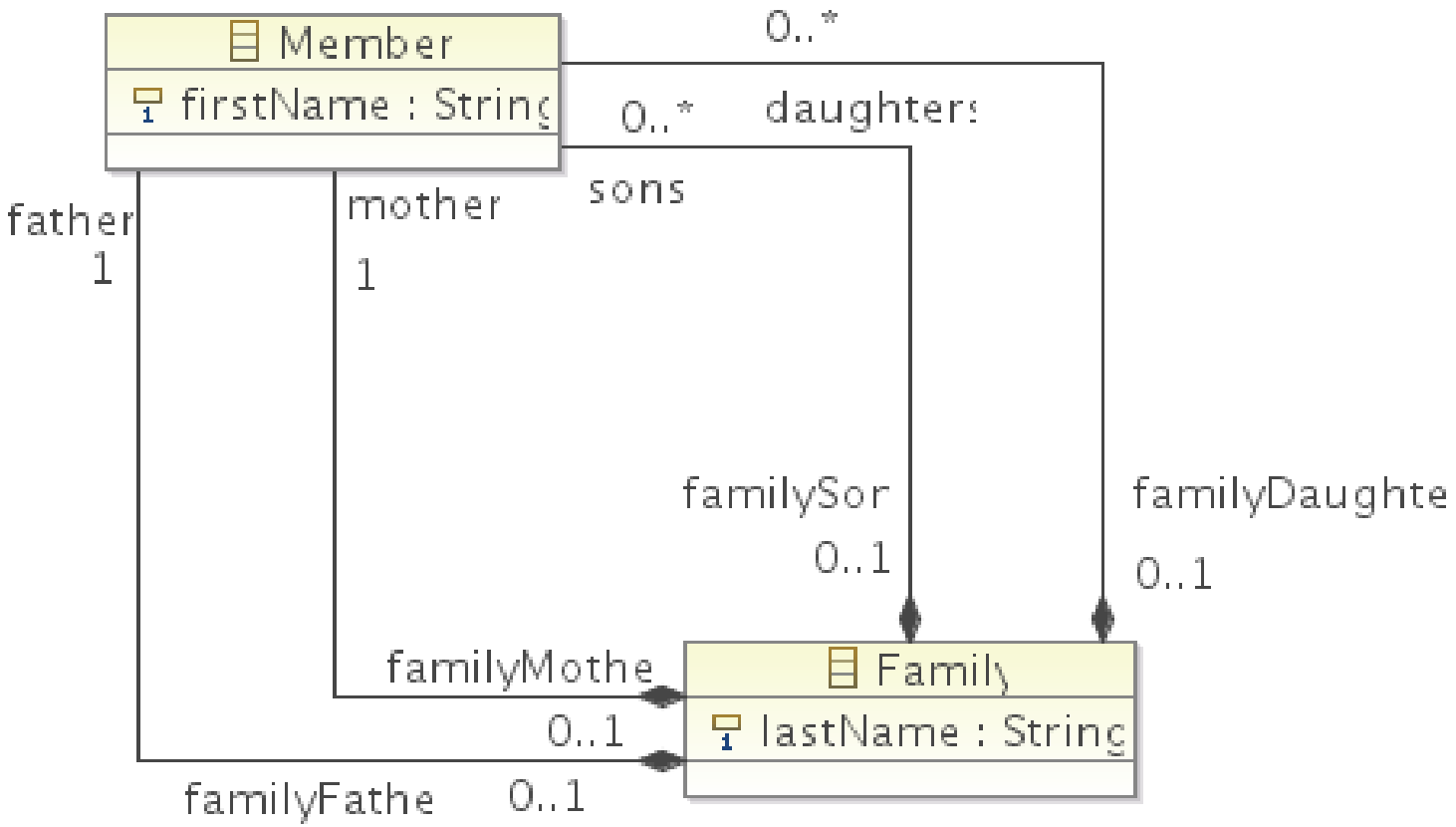
Visitor-pohjaiset mallista-tekstiin muunnokset

- Perustuu *Visitor*-suunnittelumallin käyttöön
- Jokaiselle muunnettavalle elementille toteutetaan sitä vastaava muunnosrutiini
- Esimerkkejä: Jamda

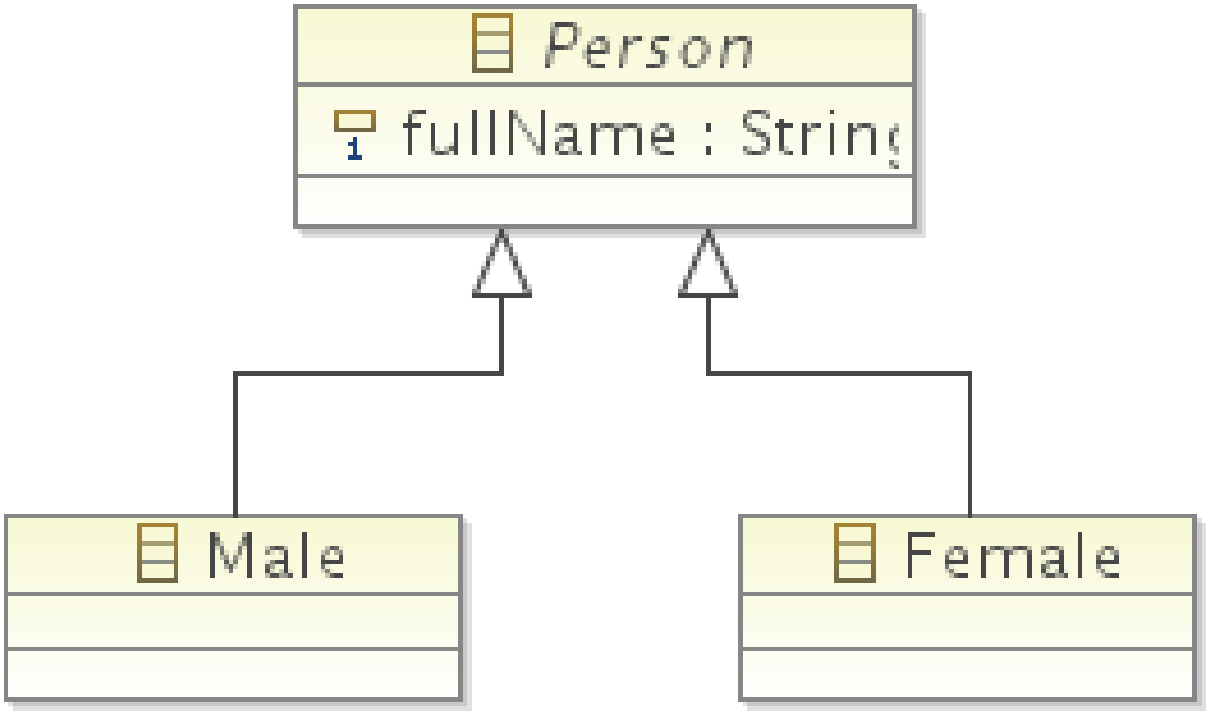
Esimerkki mallista malliin -muunnoksesta

- Yksinkertainen esimerkki mallista malliin muunnoksesta
  - Muunnetaan Families-metamallin mukainen mallin Persons-metamallin mukaiseksi malliksi
  - Käytetään Eclipse Ecorea metamallien määrittelyyn
  - Käytetään ATL -kieltä muunnosmäärittelyyn
- Lähtömalli
- Kohdemalli
- Muunnosmäärittely

Families metamalli



Persons metamalli

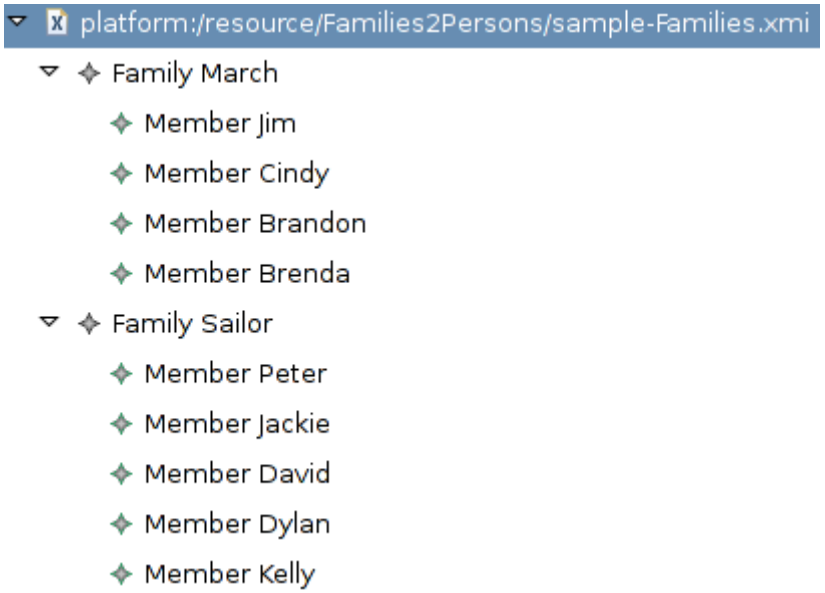


ATL muunnosmäärittely

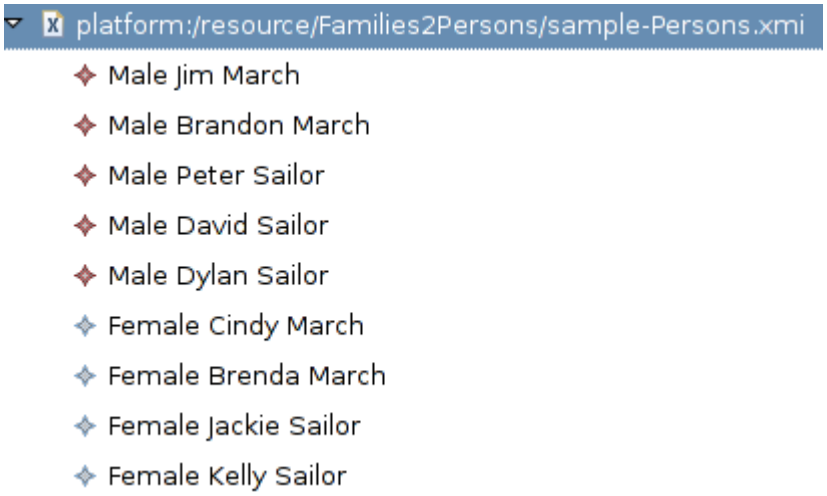
```
-- @path
Families=/Families2Persons/Families.ecore
-- @path
Persons=/Families2Persons/Persons.ecore

module Families2Persons;
create OUT : Persons from IN : Families;

helper context Families!Member def:
familyName : String =
if not self.familyFather.oclIsUndefined()
then
self.familyFather.lastName
else
if not self.familyMother.oclIsUndefined()
then
self.familyMother.lastName
else
if not self.familySon.oclIsUndefined() then
self.familySon.lastName
else
self.familyDaughter.lastName
endif
endif
endif;
```

```
helper context Families!Member def: isFemale() :
Boolean =
if not self.familyMother.oclIsUndefined() then
true
else
if not self.familyDaughter.oclIsUndefined() then
true
else
false
endif
endif;

rule Member2Male {
from
s : Families!Member (not s.isFemale())
to
t : Persons!Male (
fullName <- s.firstName + ' ' + s.familyName
)
}

rule Member2Female {
from
s : Families!Member (s.isFemale())
to
t : Persons!Female (
fullName <- s.firstName + ' ' + s.familyName
)
}
```

Lähtö- ja kohdemallit



Lähtömalli



Kohdemalli muunnoksen jälkeen

Esimerkki mallista tekstiin -muunnoksesta

- Eclipsen Acceleo työkalulla tehty mallista-tekstiin muunnos
- Muuntaa UML-mallin luokat niitä vastaaviksi Java-tiedostoiksi
- Java-luokkien nimeksi tulee UML-luokan nimi
- Yksityisiksi muuttujiksi UML-luokan attribuutit
- Metodeiksi UML-luokan määrittelemät operaatiot

```
[module generate('http://www.eclipse.org/uml2/2.1.0/UML')/]
[template public generate(c : Class)]
	[file (c.name.concat('.java'), false)]
	public class [c.name.toUpperFirst()/] {
	[for (p : Property | c.attribute)]
	private [p.type.name/] [p.name/];
	[/for]
	[for (p : Property | c.attribute)]
	public [p.type.name/] get[p.name.toUpperFirst()/]() {
		return this.[p.name/];
	}
	[/for]
	[for (o : Operation | c.ownedOperation)]
	public [o.type.name/] [o.name/]() {
		// TODO should be implemented
	}
	[/for]
	}
	[/file]
[/template]
```