Programų sistemų projektavimo pratybų užduotis

Projektuojamos situacijos aprašymas (pirma, antra dalys)

Dalykinė sritis:

Clothing factory

Pagrindinės esybės:

ClothingFactory, Clothing, Materials

Varijuojamas funkcionalumas:

Production strategija / būdas. Varijuojamas medžiagų tipas ir kokybė, gamybos tikslumas (implementacijoje kuo kruopščiau - tuo ilgiau gaminamas clothing objektas), produkto įpakavimas.

Reikalavimas A:

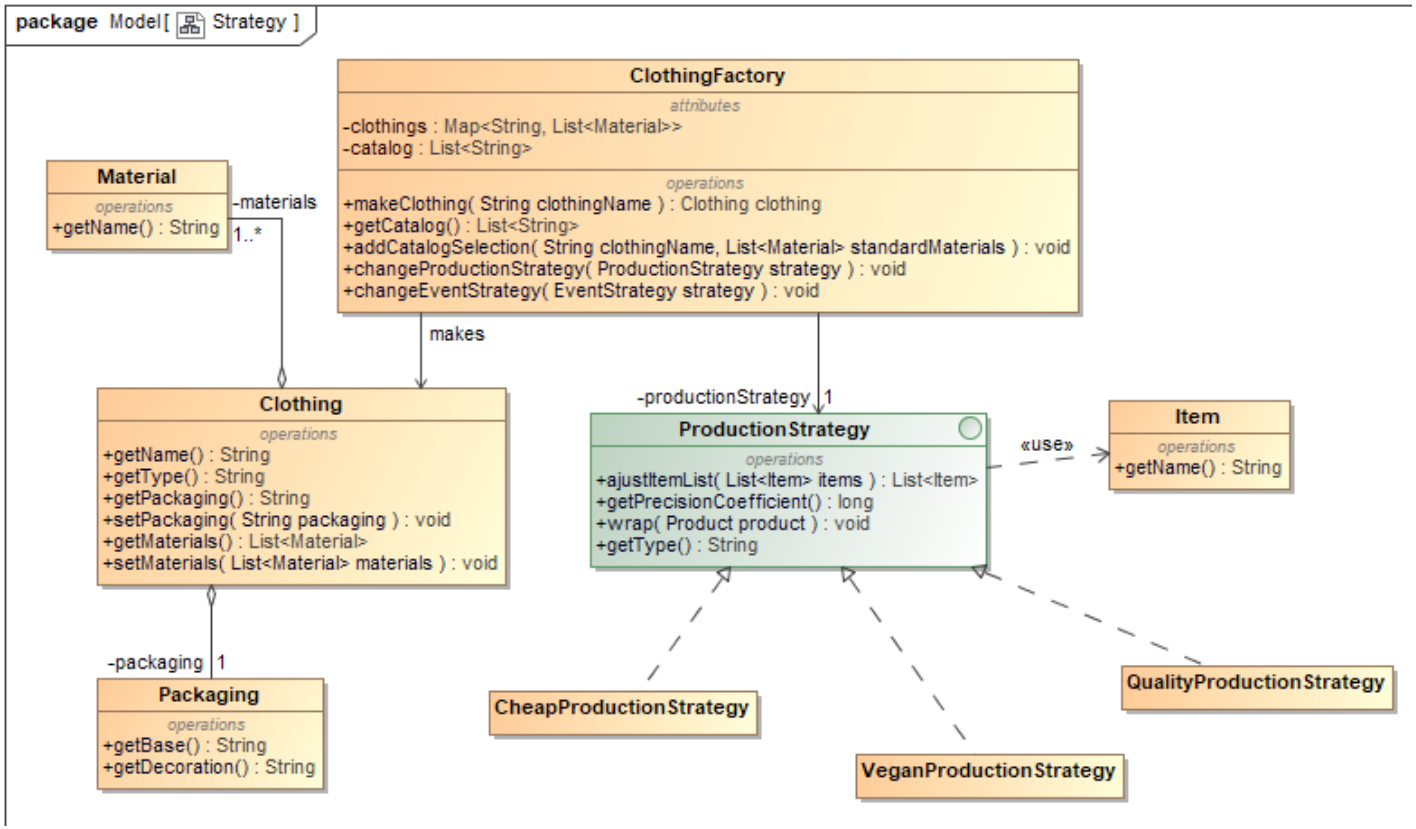
Restaurant gamina Dish iš Ingredients pagal pasirinktą startegiją / gamybos būdą, labai kruopščiai ir ilgai (arba ne), maistą (išsinešimui) skirtingai įpakuoja.

Reikalavimas B:

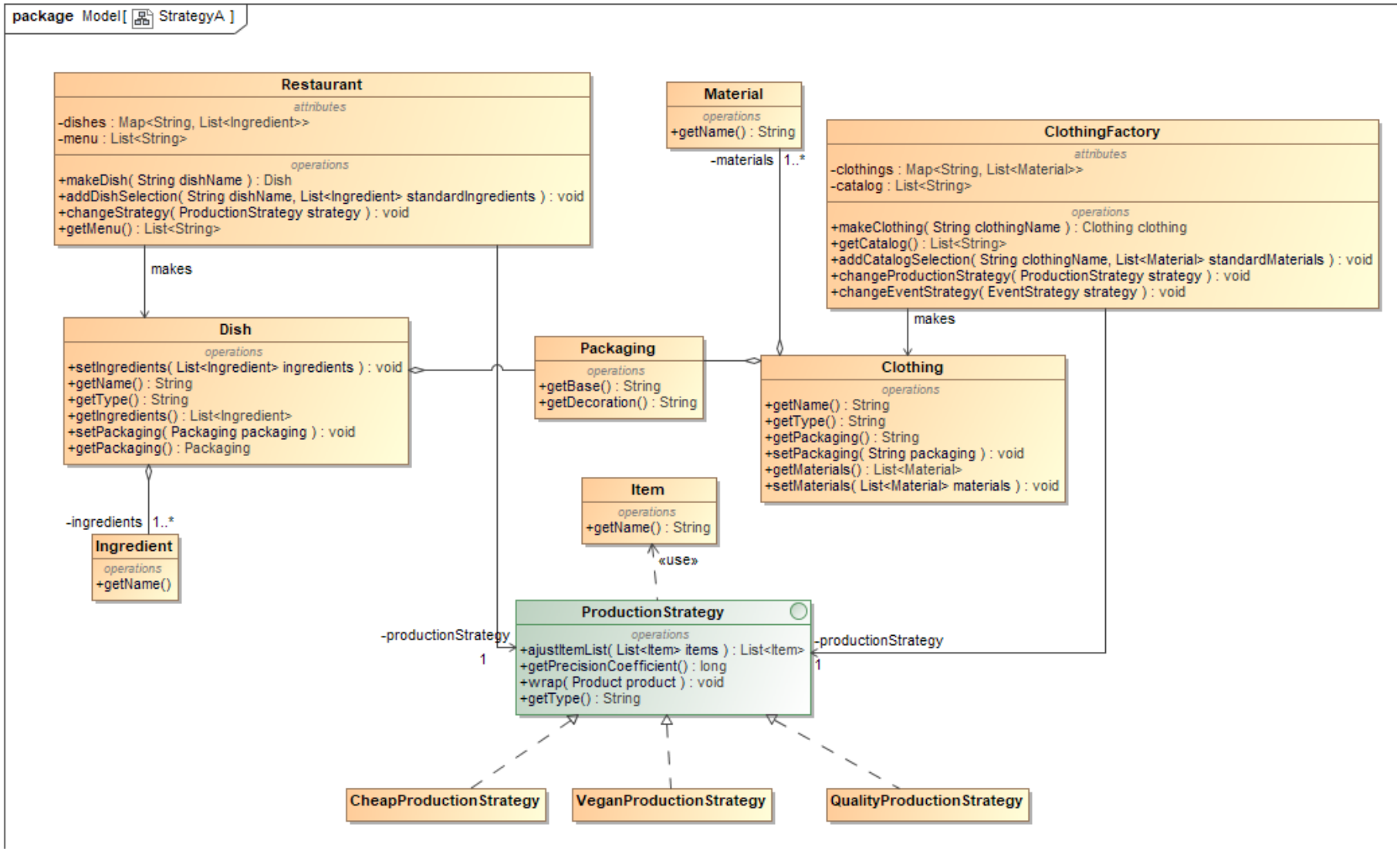
TeamBuildingEvent. Naudojamos strategijos pagal event tipą, pagal biudžetą ir dalyvių skaičių. Parenkama skirtinga vieta, dress code...

Pirma dalis

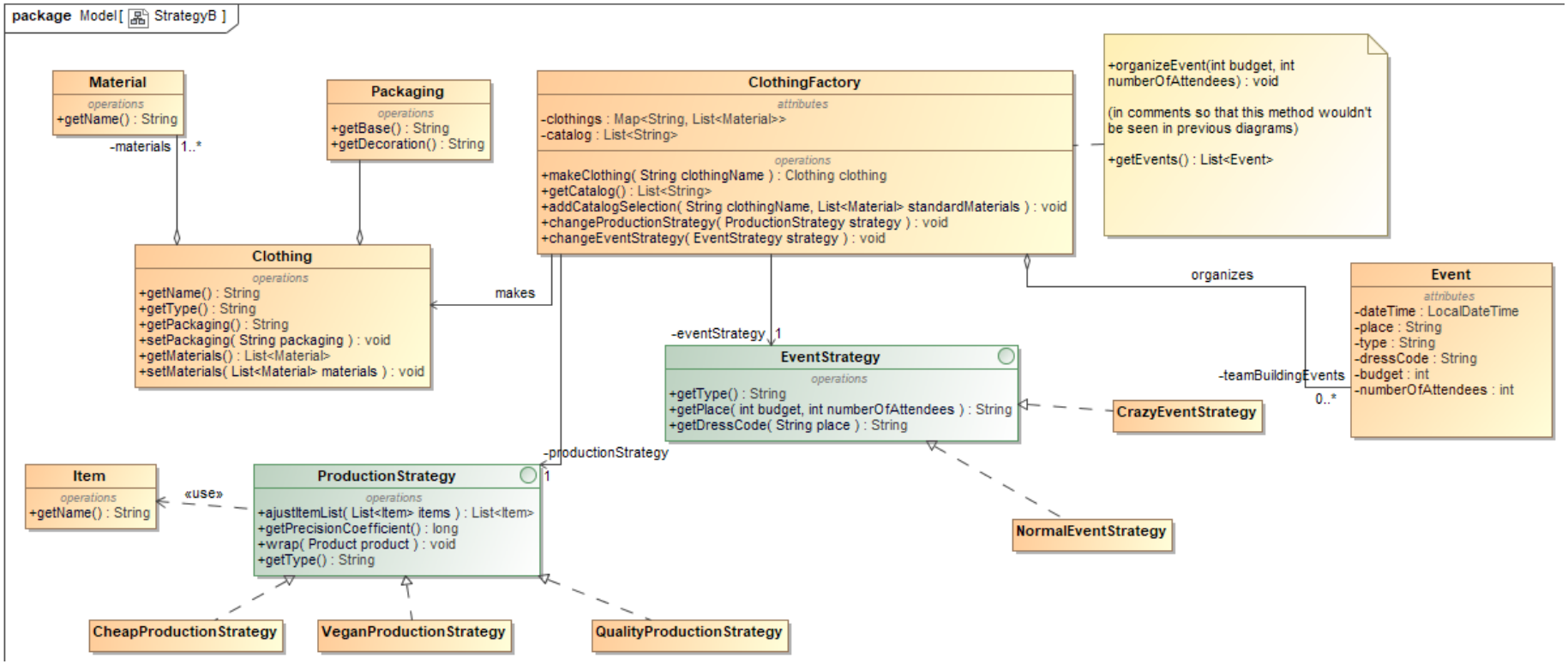
STRATEGY



1 pav. Pagrindinė situacija

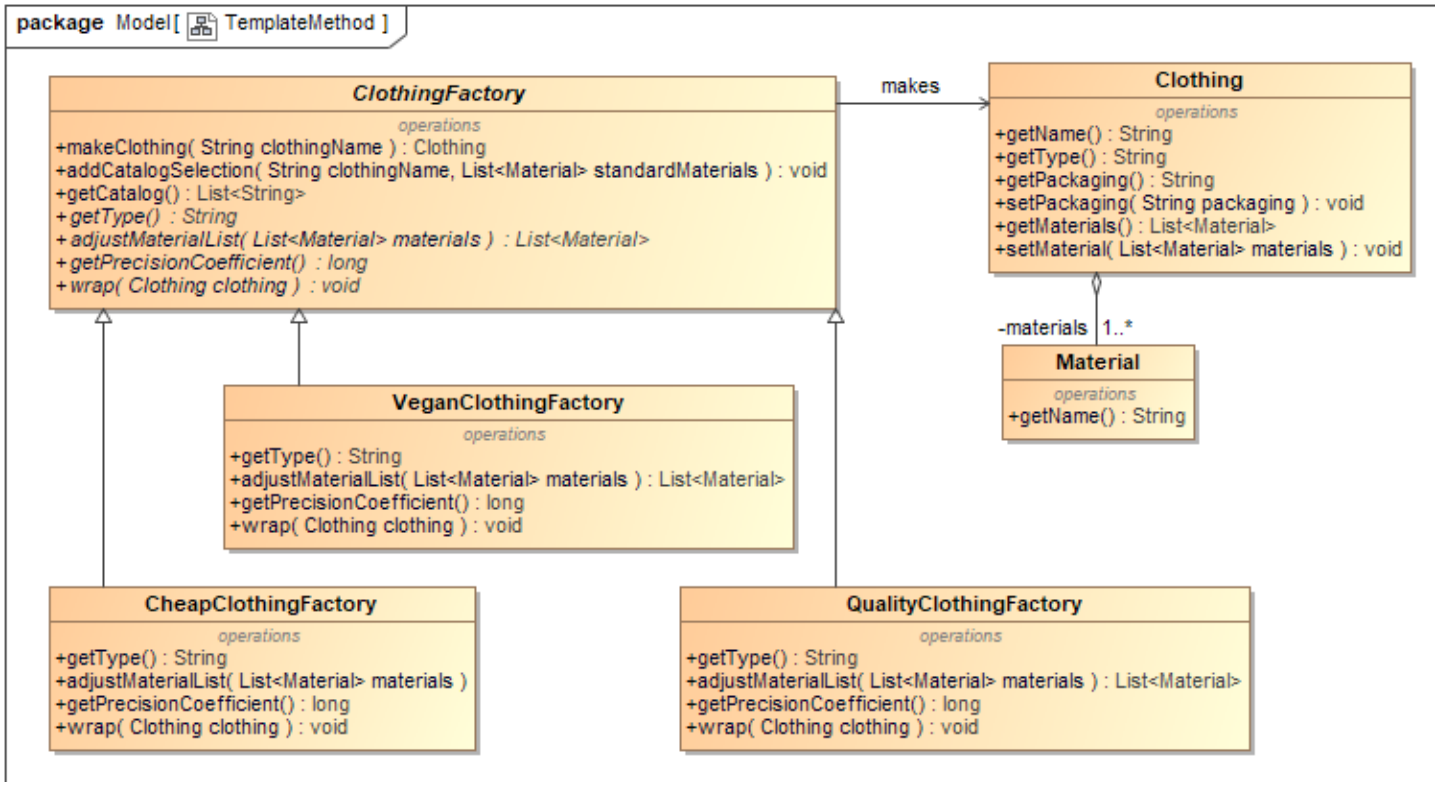


2 pav. Situacija A – nauja esybė perpanaudoja egzistuojantį funcionalumą

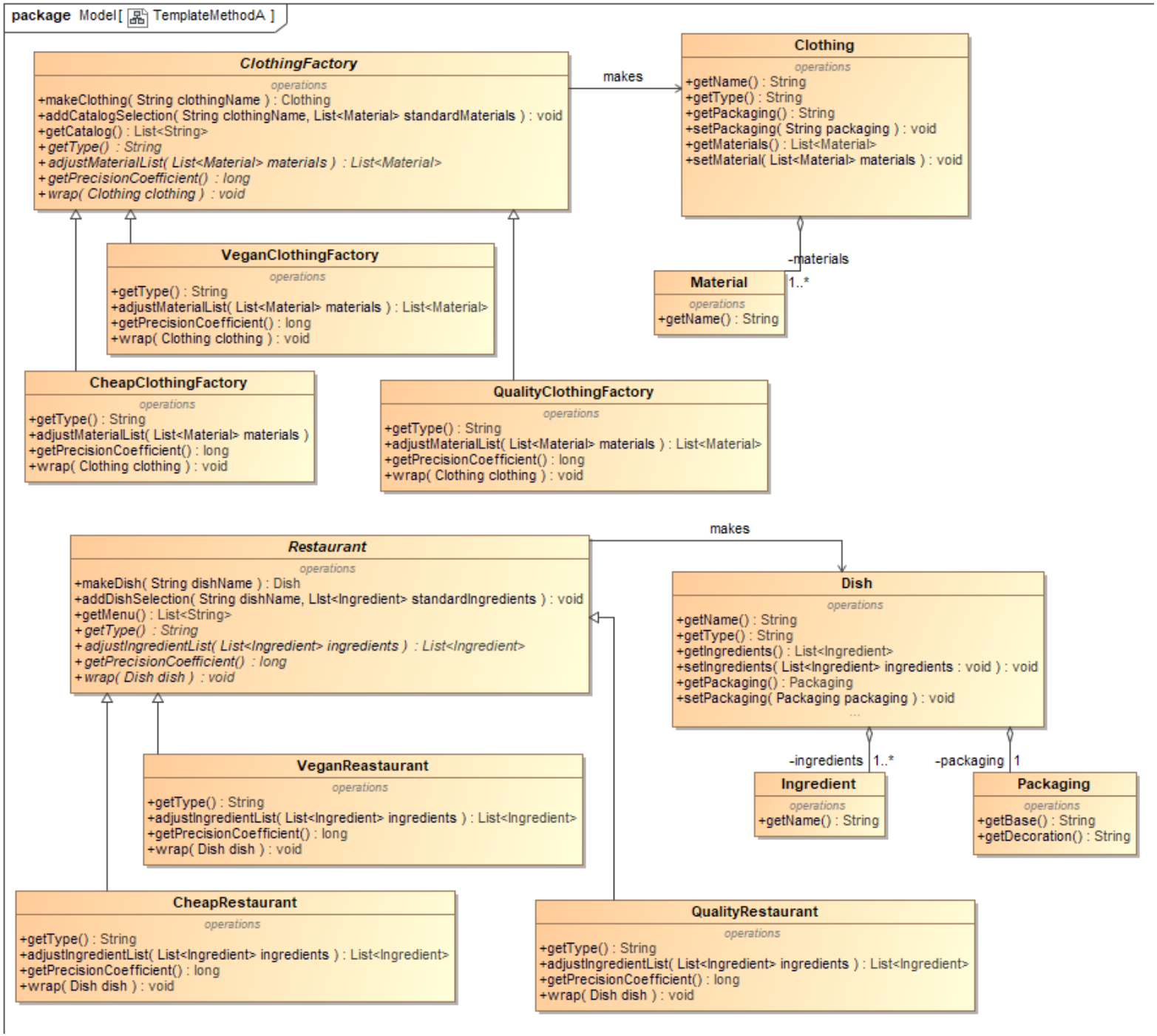


3 pav. Situacija B - esybė naudoja du varijuojamus funkcionalumus

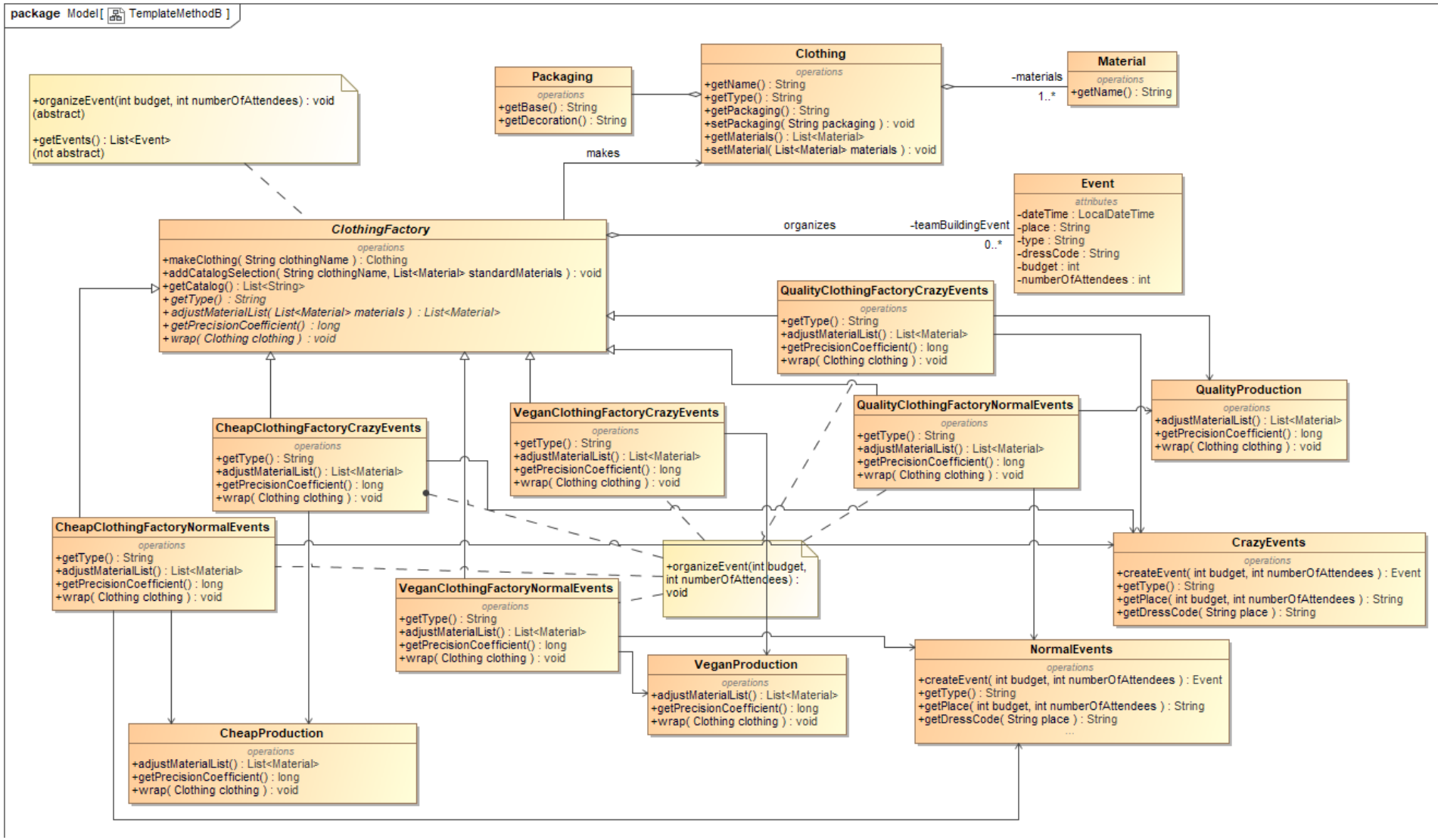
TEMPLATE METHOD



4 pav. Pagrindinė situacija


package Model[TemplateMethodA]
ClothingFactory
operations
+makeClothing(String clothingName) : Clothing
+addCatalogSelection(String clothingName, List<Material> standardMaterials) : void
+getCatalog() : List<String>
+getType() : String
+adjustMaterialList(List<Material> materials) : List<Material>
+getPrecisionCoefficient() : long
+wrap(Clothing clothing) : void
makes
Clothing
operations
+getName() : String
+getType() : String
+getPackaging() : String
+setPackaging(String packaging) : void
+getMaterials() : List<Material>
+setMaterial(List<Material> materials) : void
-materials
1..*
Material
operations
+getName() : String
VeganClothingFactory
operations
+getType() : String
+adjustMaterialList(List<Material> materials) : List<Material>
+getPrecisionCoefficient() : long
+wrap(Clothing clothing) : void
CheapClothingFactory
operations
+getType() : String
+adjustMaterialList(List<Material> materials)
+getPrecisionCoefficient() : long
+wrap(Clothing clothing) : void
QualityClothingFactory
operations
+getType() : String
+adjustMaterialList(List<Material> materials) : List<Material>
+getPrecisionCoefficient() : long
+wrap(Clothing clothing) : void
makes
Restaurant
operations
+makeDish(String dishName) : Dish
+addDishSelection(String dishName, List<Ingredient> standardIngredients) : void
+getMenu() : List<String>
+getType() : String
+adjustIngredientList(List<Ingredient> ingredients) : List<Ingredient>
+getPrecisionCoefficient() : long
+wrap(Dish dish) : void
Dish
operations
+getName() : String
+getType() : String
+getIngredients() : List<Ingredient>
+setIngredients(List<Ingredient> ingredients : void) : void
+getPackaging() : Packaging
+setPackaging(Packaging packaging) : void
...
-ingredients
1..*
-packaging
1
Ingredient
operations
+getName() : String
Packaging
operations
+getBase() : String
+getDecoration() : String
VeganReastaurant
operations
+getType() : String
+adjustIngredientList(List<Ingredient> ingredients) : List<Ingredient>
+getPrecisionCoefficient() : long
+wrap(Dish dish) : void
CheapRestaurant
operations
+getType() : String
+adjustIngredientList(List<Ingredient> ingredients) : List<Ingredient>
+getPrecisionCoefficient() : long
+wrap(Dish dish) : void
QualityRestaurant
operations
+getType() : String
+adjustIngredientList(List<Ingredient> ingredients) : List<Ingredient>
+getPrecisionCoefficient() : long
+wrap(Dish dish) : void

5 pav. Situacija A – nauja esybė su tuo pačiu varijuojamu funkcionalumu



6 pav. Situacija B - esybė naudoja du varijuojamus funkcionalumus

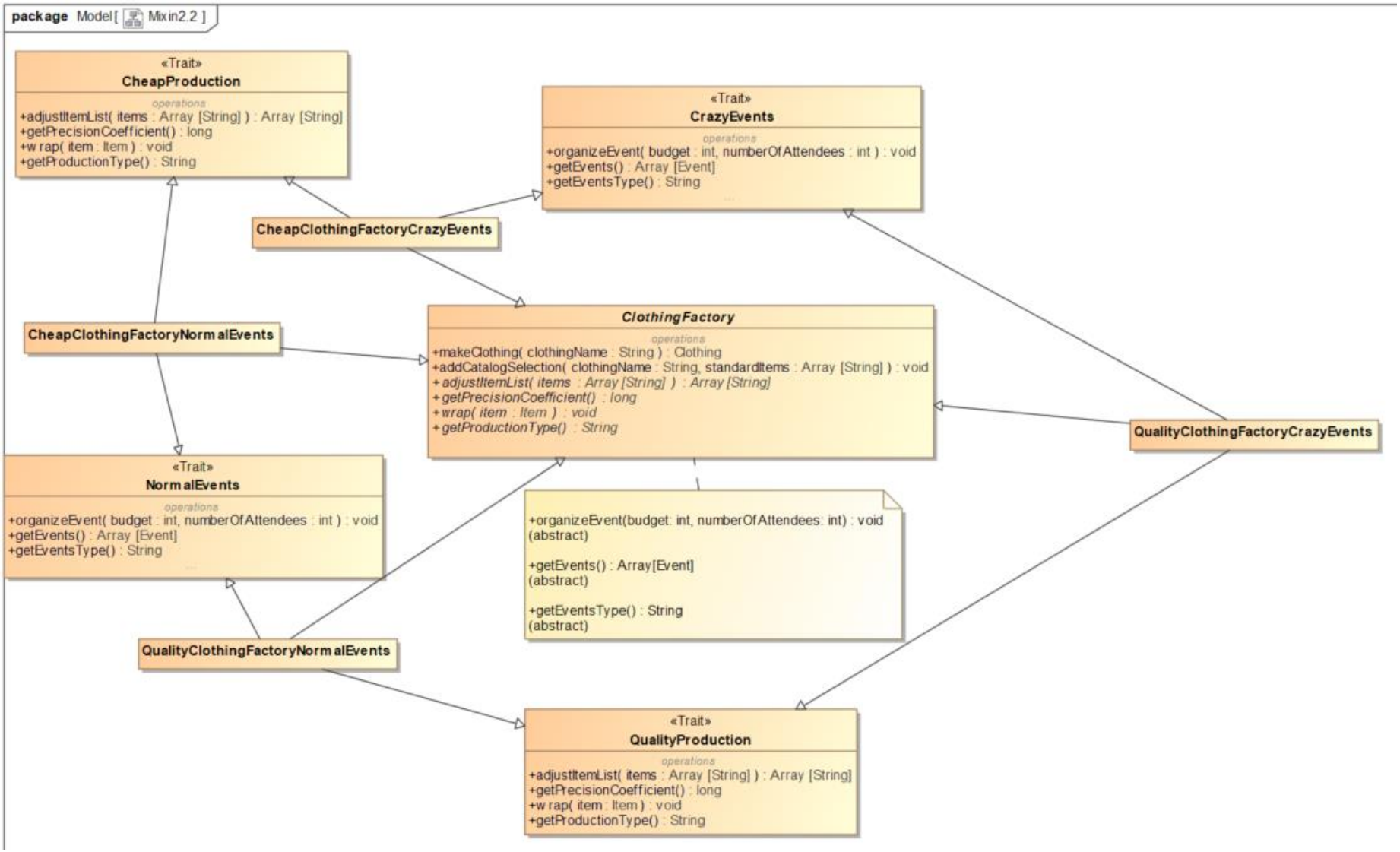
Abiejų projektavimo šablonų palyginimas

Pagrindinė situacija atrodo vienodai tinkama suprojektuoti tie su Strategy, tiek su TemplateMethod.

Situacijai A (varijuojamo funkcionalumo perpanaudojimas su kita esybe) akivaizdžiai labiau tinka Strategy šablonas, nes galima pasinaudoti tuo pačiu interfeisu ir implementuojančiomis funkcionalumo kalsėmis. TemplateMethod šablono atveju atsiranda daugiau klasių.

Situacija B (esybė turi kelis varijuojamus funkcionalums) irgi labiau tinka Strategy šablonas, vien pažvelgus į TemplateMethod šablono įgyvendinimo betvarkę aišku, kad sugeneruojama gerokai daugiau klasių ir ryšiai vis labiau komplikuojasi.

Antra dalis



7 pav. Esybė naudoja du varijuojamus funkcionalumus

Mixin metodo įvertinimas

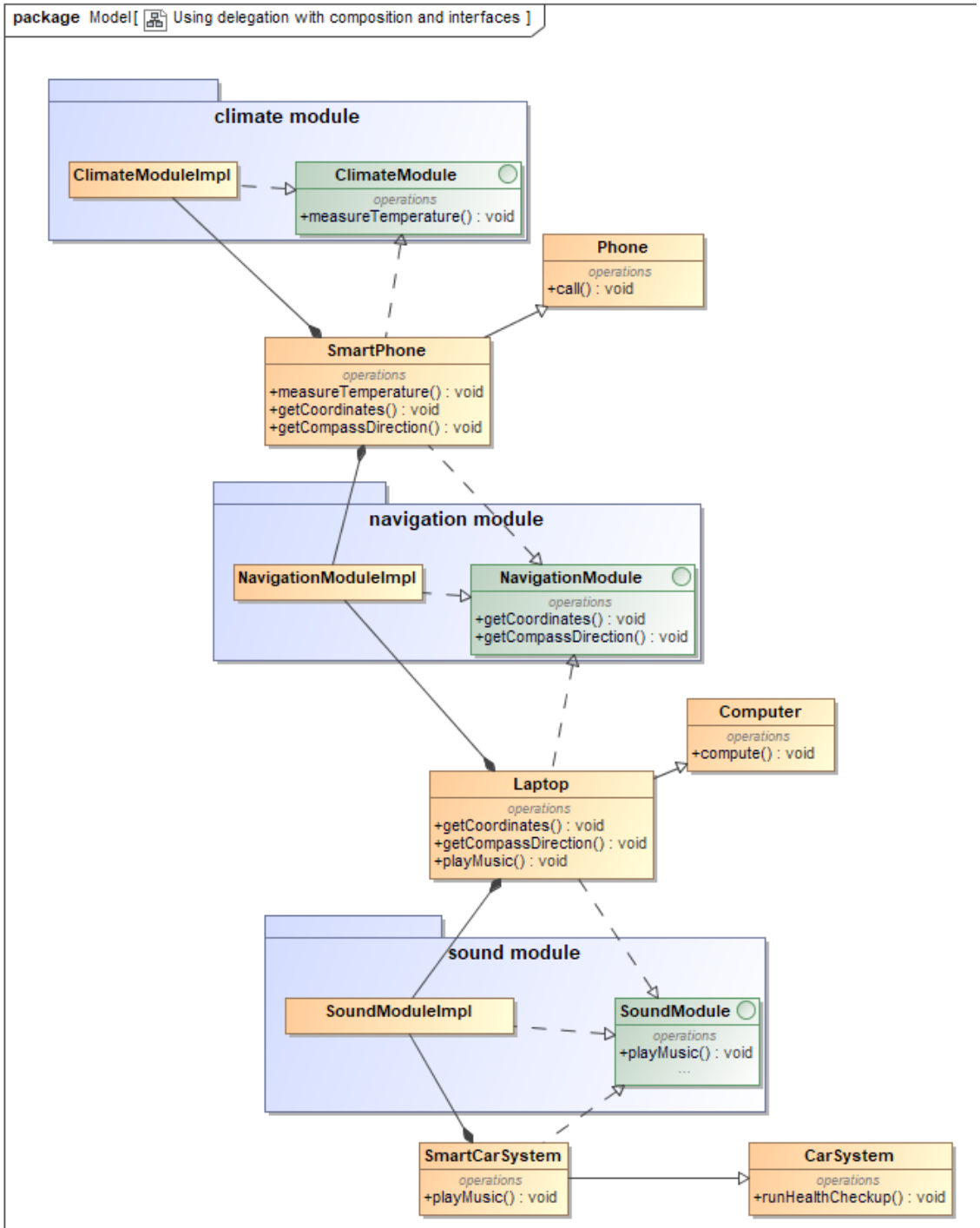
Naudojant daugybinį paveldėjimą diagrama pasidaro švaresnė ir tvarkingesnė, pavyzdžio implementacija struktūriškai gaunasi lengviau valdoma, nes naudojamas paveldėjimas, o ne nevaldoma agregacija. Kodo dubliavimo problemos išsprendžiamos pasikartojantį funkcionalumą perkeliant į tėvines klases ir paveldint iš skirtingų tėvinių klasių, “susirenkant” norimą funkcionalumą. Kuriamose klasėse nebereikia aprašyti tuščių metodų, kurie tik deleguotų darbą pagalbinėms klasėms.

Trečia dalis

Projektuojamos situacijos aprašymas (trečia dalis)

Esybės: Computer, CarSystem, Phone.

Papildomi funkcionalumai: climate (temperature measuring), sound (playing music), navigation (getting coordinates, getting compass direction).


package Model[Using delegation with composition and interfaces]
climate module
ClimateModuleImpl
ClimateModule
operations
+measureTemperature() : void
Phone
operations
+call() : void
SmartPhone
operations
+measureTemperature() : void
+getCoordinates() : void
+getCompassDirection() : void
navigation module
NavigationModuleImpl
NavigationModule
operations
+getCoordinates() : void
+getCompassDirection() : void
Computer
operations
+compute() : void
Laptop
operations
+getCoordinates() : void
+getCompassDirection() : void
+playMusic() : void
sound module
SoundModuleImpl
SoundModule
operations
+playMusic() : void
...
SmartCarSystem
operations
+playMusic() : void
CarSystem
operations
+runHealthCheckup() : void

8 pav. Delegavimas naudojant kompoziciją ir interfeisus

Kas lengviau?

Manau, kad lengviau / geriau / lanksčiau yra sukurti ir įterpti naują funkcionalumo (mixin) klasę, nei papildyti jau esamą. Tokiu atveju lengviau išlaikyti tvarkingą, aiškų kodą, klasės, naudojančios funkcionalumo nebus priverstos turėti ir žinoti apie metodus, kurių joms nereikia, kurie pridėti dėl kitų klasių.