Example System: BurgerOrderer

Mikael Svahnberg

April 24, 2020

Contents

1 Use Case: Order Food

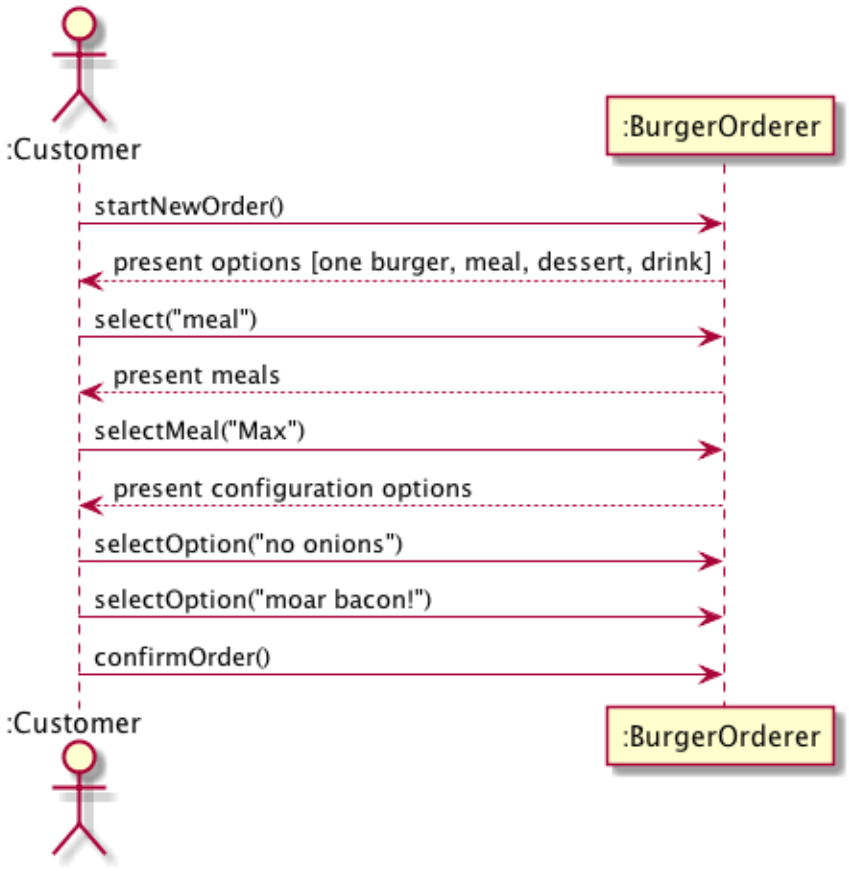
Use case: Order Food Actors: Customer Description: A customer arrives at the BurgerOrderer, selects a meal, configures their hamburger, and orders it. Related Use cases: Pay for Order

Main Course of Events

Actor	System
1. Customer arrives at BurgerOrderer and starts a new order	
	2. System presents options [just one burger, full meal, dessert, dr
3. Customer selects "full meal"	
	4. System presents available meals
5. Customer selects a meal	
	6. System adds the selected meal to o presents configuration options
7. Customer selects "no onions"	
	8. System adds "no onions" to the or
9. Customer selects "more bacon!"	
	10. System adds "more bacon!" to th
11. Customer confirms order	
	12. System initiates use case pay for
	13. System places burger order to the and prints a receipt.

2 System Sequence Diagram

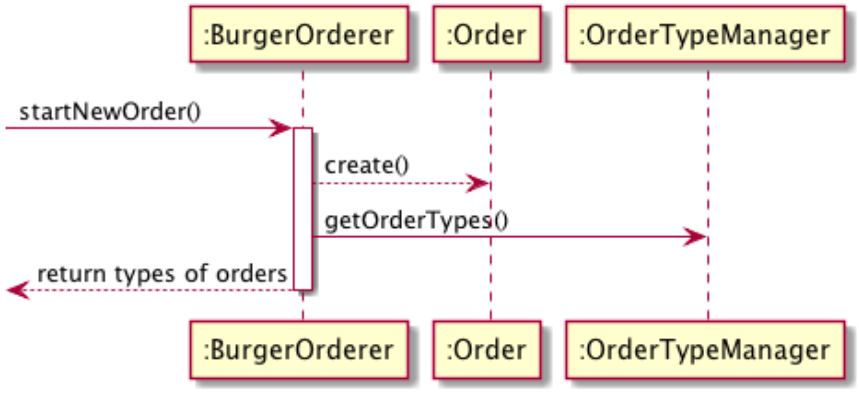
```
actor ":Customer" as customer
participant ":BurgerOrderer" as sys

customer -> sys : startNewOrder()
sys --> customer : present options [one burger, meal, dessert, drink]
customer -> sys : select("meal")
sys --> customer : present meals
customer -> sys : selectMeal("Max")
sys --> customer : present configuration options
customer -> sys : selectOption("no onions")
customer -> sys : selectOption("moar bacon!")
customer -> sys : confirmOrder()
```



3 Interaction Diagrams (Sequence Diagrams)

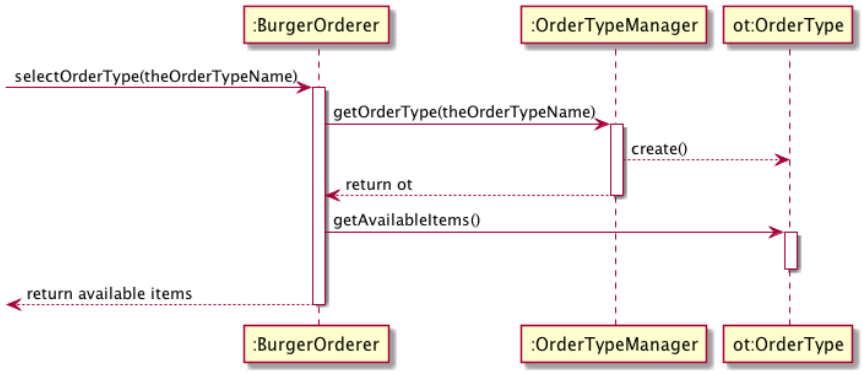
3.1 startNewOrder()

```
participant ":BurgerOrderer" as sys

[-> sys : startNewOrder()
activate sys
' ---------- How the system intents to solve this system call is described below -----
sys --> ":Order" : create()
sys -> ":OrderTypeManager" : getOrderTypes()
[<-- sys : return types of orders
' ---------- end ----------
deactivate sys
```



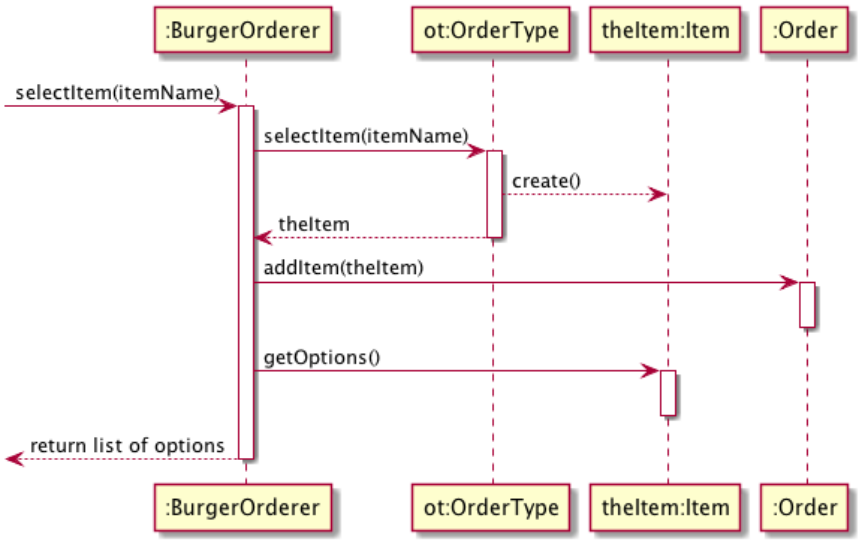
3.2 selectOrderType()

```
participant ":BurgerOrderer" as sys

[-> sys : selectOrderType(theOrderTypeName)
activate sys
' ---------- How the system intents to solve this system call is described below -----
sys -> ":OrderTypeManager" : getOrderType(theOrderTypeName)
activate ":OrderTypeManager"
":OrderTypeManager" --> "ot:OrderType" : create()
":OrderTypeManager" --> sys : return ot
deactivate ":OrderTypeManager"

sys -> "ot:OrderType" : getAvailableItems()
activate "ot:OrderType"
deactivate "ot:OrderType"

[<-- sys : return available items
' ---------- end ----------
deactivate sys
```



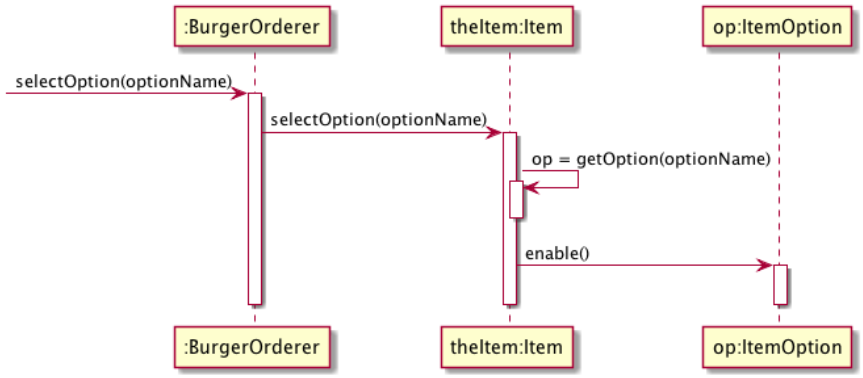
3.3 selectItem("MaxMeal")

```
participant ":BurgerOrderer" as sys

[-> sys : selectItem(itemName)
activate sys
' ---------- How the system intents to solve this system call is described below ------
sys -> "ot:OrderType" : selectItem(itemName)
activate "ot:OrderType"
"ot:OrderType" --> "theItem:Item" : create()
"ot:OrderType" --> sys : theItem
deactivate "ot:OrderType"
sys -> ":Order" : addItem(theItem)
activate ":Order"
deactivate ":Order"

sys -> "theItem:Item" : getOptions()
activate "theItem:Item"
deactivate "theItem:Item"
[<-- sys : return list of options
' ---------- end ----------
deactivate sys
```



3.4 selectOption("no onions")

```
participant ":BurgerOrderer" as sys

[-> sys : selectOption(optionName)
activate sys
' ---------- How the system intents to solve this system call is described below ------
sys -> "theItem:Item" : selectOption(optionName)
activate "theItem:Item"
"theItem:Item" -> "theItem:Item" : op = getOption(optionName)
activate "theItem:Item"
deactivate "theItem:Item"

"theItem:Item" -> "op:ItemOption" : enable()
activate "op:ItemOption"
deactivate "op:ItemOption"

deactivate "theItem:Item"
' ---------- end ----------
deactivate sys
```

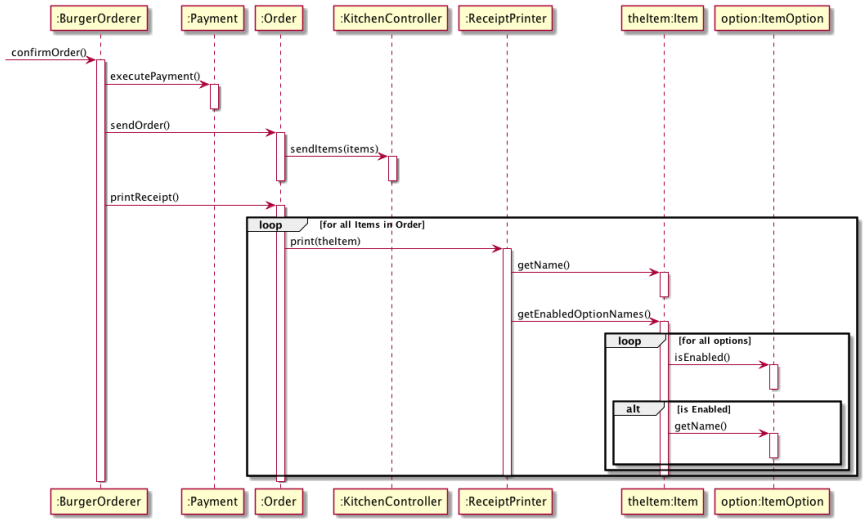


3.5 selectOption("more bacon!")

3.6 confirmOrder()

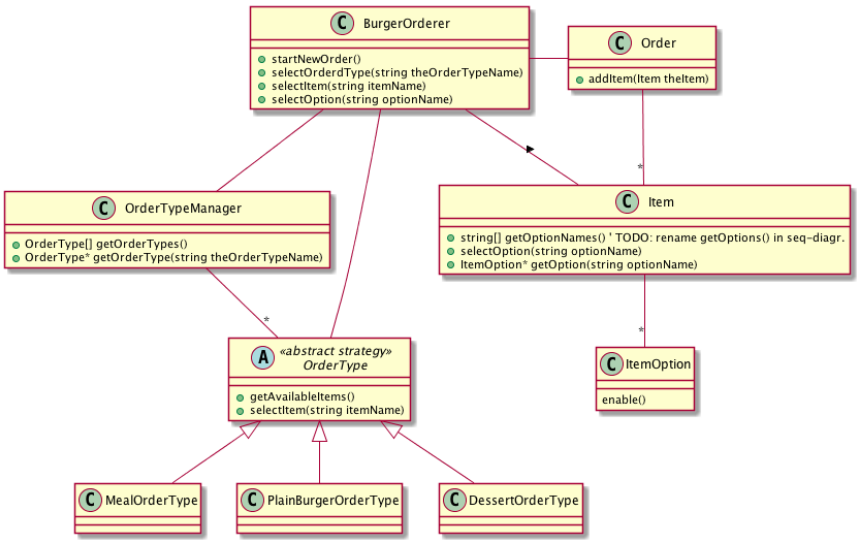
```
participant ":BurgerOrderer" as sys

[-> sys : confirmOrder()
activate sys
' ---------- How the system intents to solve this system call is described below ------
sys -> ":Payment" : executePayment()
activate ":Payment"
deactivate ":Payment"

sys -> ":Order" : sendOrder()
activate ":Order"
":Order" -> ":KitchenController" : sendItems(items)
activate ":KitchenController"
deactivate ":KitchenController"
deactivate ":Order"

sys -> ":Order" : printReceipt()
activate ":Order"

loop for all Items in Order
":Order" -> ":ReceiptPrinter" : print(theItem)
activate ":ReceiptPrinter"
":ReceiptPrinter" -> "theItem:Item" : getName()
activate "theItem:Item"
deactivate "theItem:Item"
```

```
":ReceiptPrinter" -> "theItem:Item" : getEnabledOptionNames()
activate "theItem:Item"
loop for all options
"theItem:Item" -> "option:ItemOption" : isEnabled()
activate "option:ItemOption"
deactivate "option:ItemOption"

alt is Enabled
"theItem:Item" -> "option:ItemOption" : getName()
activate "option:ItemOption"
deactivate "option:ItemOption"
end alt

end loop
deactivate "theItem:Item"

deactivate ":ReceiptPrinter"
end loop
deactivate ":Order"

' ---------- end ----------
deactivate sys
```



4 Class Diagram

```
class BurgerOrderer {
+startNewOrder()
+selectOrderdType(string theOrderTypeName)
+selectItem(string itemName)
+selectOption(string optionName)
}

class Order {
 +addItem(Item theItem)
}

OrderTypeManager : +OrderType[] getOrderTypes()
OrderTypeManager : +OrderType* getOrderType(string theOrderTypeName)

abstract class OrderType <<abstract strategy>>
OrderType : +getAvailableItems()
OrderType : +selectItem(string itemName)
OrderType <|-- MealOrderType
OrderType <|-- PlainBurgerOrderType
OrderType <|-- DessertOrderType

Item : +string[] getOptionNames() ' TODO: rename getOptions() in seq-diagr.
Item : +selectOption(string optionName)
Item : +ItemOption* getOption(string optionName)

ItemOption : enable()

BurgerOrderer - Order
BurgerOrderer -- OrderTypeManager
BurgerOrderer --- OrderType
BurgerOrderer -- Item : >

OrderTypeManager -- "*" OrderType

Item -- "*" ItemOption

Order -- "*" Item
```



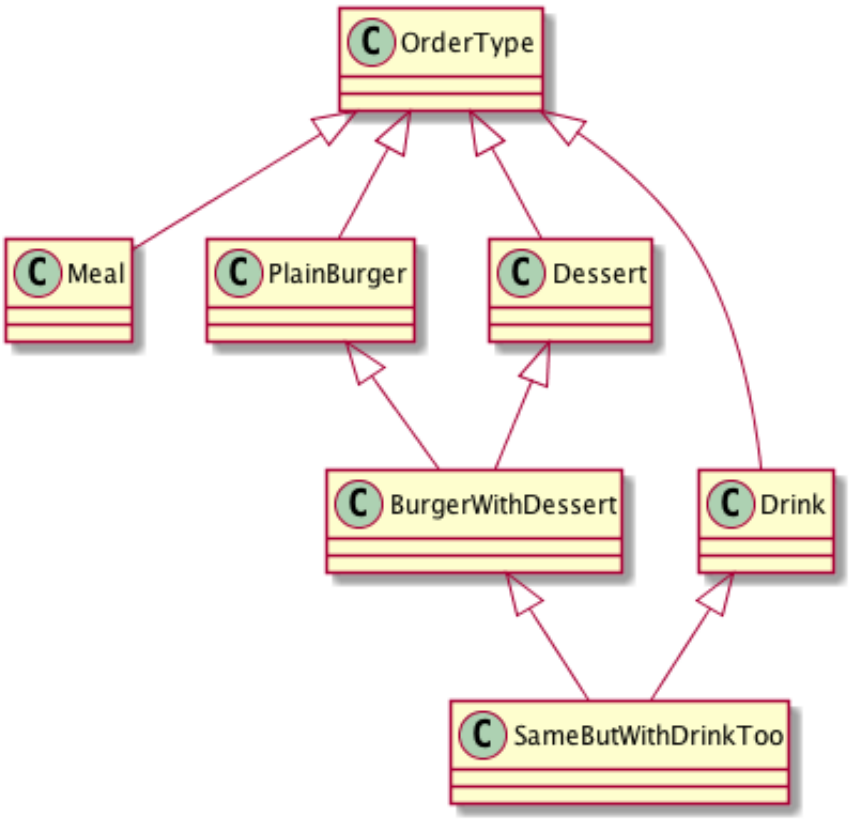
5 Entity Component System

Ett förslag på hur man kan göra "OrderType" annorlunda, för att slippa en del problem om man vill sätta ihop nya ordertyper.

5.1 Bakgrund: lös det med arv

Hur man skulle behövt lösa kompositionen om man inte använt ECS utan vanliga arvshierarkier

```
OrderType <|-- Meal
OrderType <|-- PlainBurger
OrderType <|-- Dessert
OrderType <|-- Drink

PlainBurger <|-- BurgerWithDessert
Dessert <|-- BurgerWithDessert

BurgerWithDessert <|-- SameButWithDrinkToo
Drink <|-- SameButWithDrinkToo
```

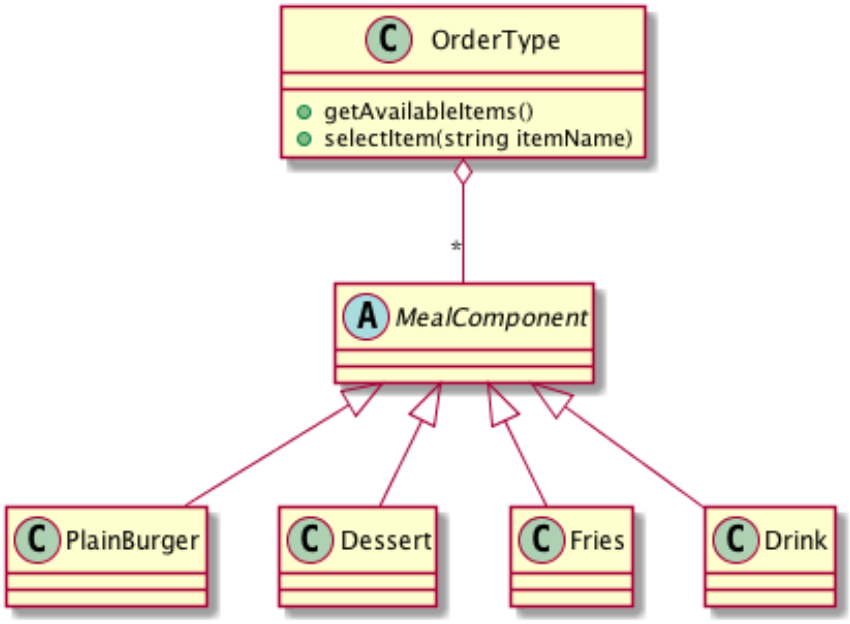


. . . Som synes så blir det nya (multipla) arv varje gång man vill skapa en ny ordertyp. Jobbigt, svårt med multipla arv, man måste koda om systemet varje gång man vill skapa en ny ordertyp.

5.2 Alternaiv: Entity Component System

```
class OrderType {
 +getAvailableItems()
 +selectItem(string itemName)
}

abstract class MealComponent
MealComponent <|-- PlainBurger
MealComponent <|-- Dessert
MealComponent <|-- Fries
```

```
MealComponent <|-- Drink

OrderType o-- "*" MealComponent
```




```
// Note: Here be dragons! Egregious pseudocode below...
class OrderType {
private:
 MealComponent** myComponents;
public:
 void create(string** componentNames) {
  for(c in componentNames) {
    myComponents.push(createNewMealComponent(c));
  }
}

 MealComponent* createNewMealComponent(string cName) {
   switch (cName) {
 case "Burger" : return new PlainBurger(); break;
 case "Dessert" : return new Dessert(); break;
}
}
```

```
}
```

Slutsats: Lätt att kombinera ihop nya ordertyper – vid runtime om så krävs.