Example: Robot System

Mikael Svahnberg*

March 12, 2019

Contents

1 Introduktion

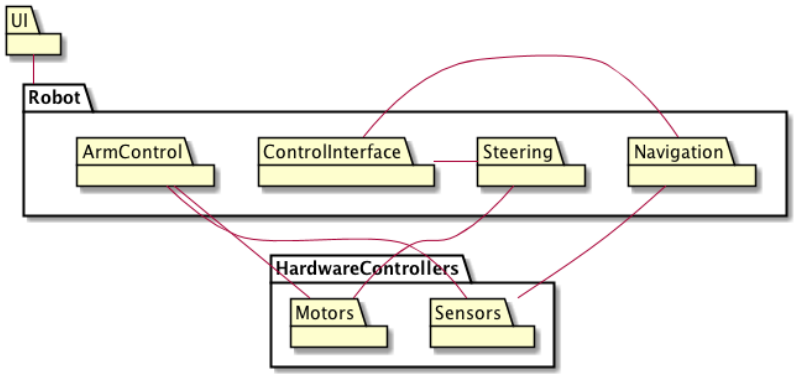
Det här är ett exempel som vi började arbeta med på lektionen den *<2019-02-15 Fri>*, med tanke att fortsätta nästa lektion. Exemplet är inte komplett och förmodligen svårt att hänga med på om man inte var med på föreläsningen; om du är osäker så fråga dina studentkollegor.

2 Package Diagram

```
package UI

package Robot {
package ControlInterface
package Navigation
package Steering
package ArmControl
```

*Mikael.Svahnberg@bth.se

```
ControlInterface - Navigation
ControlInterface - Steering

}

package HardwareControllers {
package Motors
package Sensors
}

UI - Robot
Steering - Motors
Navigation - Sensors
ArmControl - Motors
ArmControl - Sensors
```



3 Use Case: Navigate to Point

Use Case: Navigate to Point **Actors**: User, (System) **Description**: User selects a coordinate and asks for possible routes to this point. System displays possible routes. User selects one route. **Requirements:** FR1, FR10, QA2.

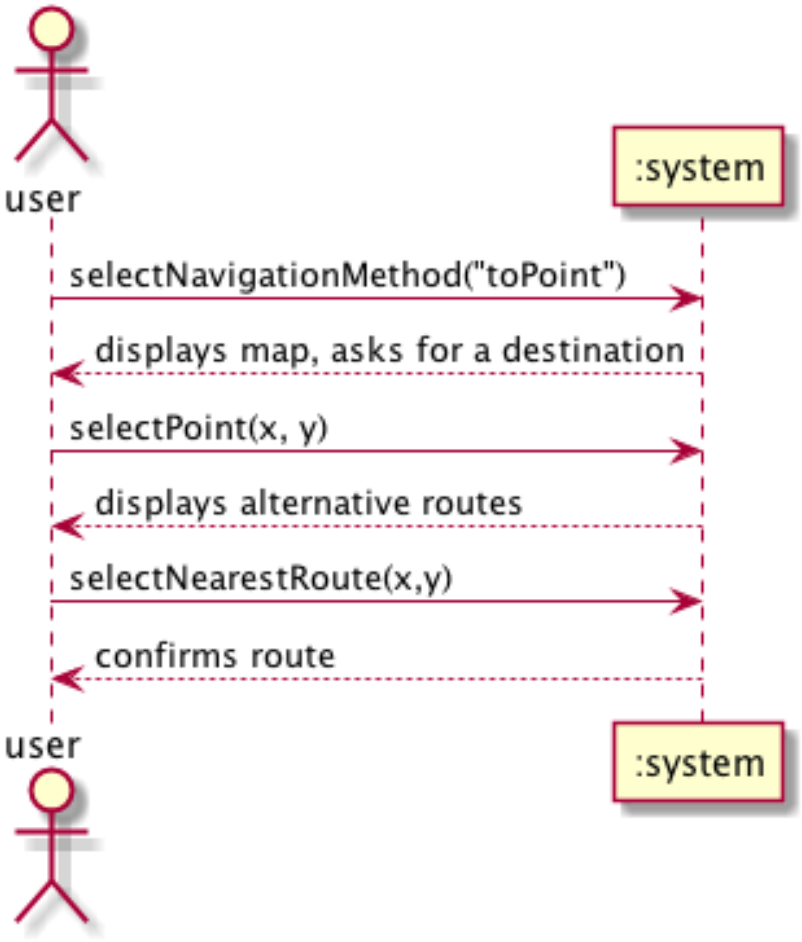
Main Course of Events

Actor	System
User selects "navigate to point"	
	System displays map and asks user to select a point
User selects a point	
	System calculates routes and displays
User selects one route	

4 Systemsekvensdiagram: Navigate to Point

```
actor user
```

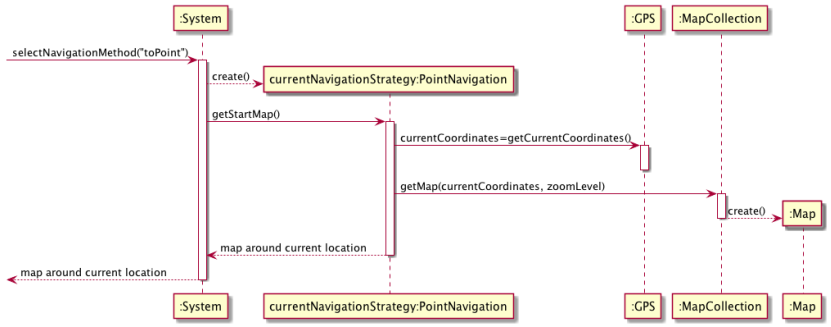
```
participant ":system" as sys

user -> sys : selectNavigationMethod("toPoint")
sys --> user : displays map, asks for a destination
user -> sys : selectPoint(x, y)
sys --> user : displays alternative routes
user -> sys : selectNearestRoute(x,y)
sys --> user : confirms route
```



Kommentar: Systemet borde returnera saker här också. För varje systemhändelse skall det returneras något som kan mappas mot vad som "lovades" i use case:t. På samma sätt som man tar varje enskild systemhändelse som grund till ett sekvensdiagram, så skall man ta med sig returvärderna härifrån. Se sekvensdiagrammen nedan; De börjar med systemhändelsen och skall sluta med samma returvärde som man fick tillbaka enligt systemsekvensdiagrammet. Uppdaterat diagrammet enligt detta *<2019-02-18 Mon>*.

5 Sekvensdiagram - selectNavigationMethod

```
[-> ":System" : selectNavigationMethod("toPoint")
activate ":System"

create "currentNavigationStrategy:PointNavigation"
":System" --> "currentNavigationStrategy:PointNavigation" : create()

":System" -> "currentNavigationStrategy:PointNavigation" : getStartMap()
activate "currentNavigationStrategy:PointNavigation"
"currentNavigationStrategy:PointNavigation" -> ":GPS" : currentCoordinates=getCurrentCoord
activate ":GPS"
deactivate ":GPS"


"currentNavigationStrategy:PointNavigation" -> ":MapCollection" : getMap(currentCoordinate
activate ":MapCollection"
create ":Map"
":MapCollection" --> ":Map" : create()
deactivate ":MapCollection"

"currentNavigationStrategy:PointNavigation" --> ":System" : map around current location
deactivate "currentNavigationStrategy:PointNavigation"
":System" -->[ : map around current location
deactivate ":System"
```

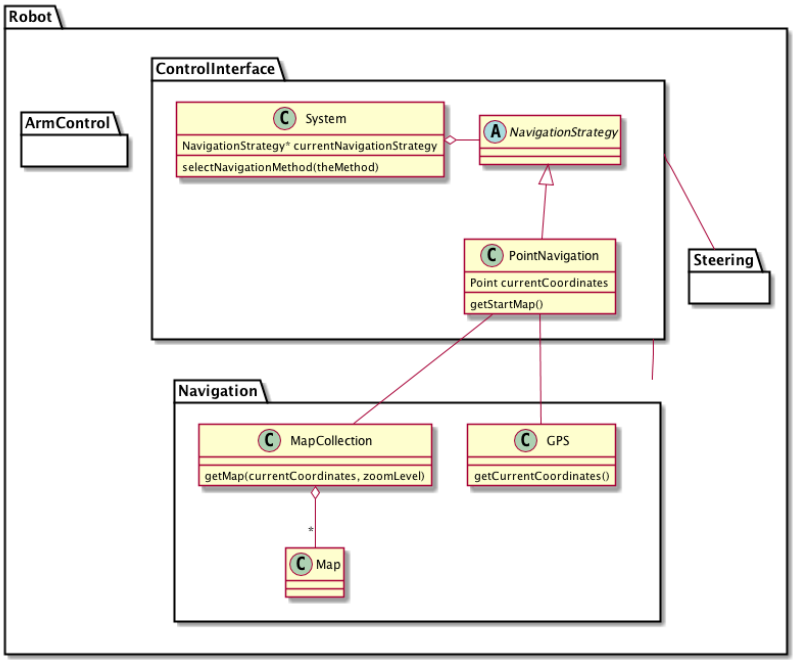


Kommentar: Som mycket riktigt påpekades efter föreläsningen så kan ju inte en konstruktor returnera en massa värden eller kartor och annat. Man behöver alltså dela upp anropen från `:System` till `currentNavigationStrategy:PointNavigation` till en `create()` (som inte returnerar något) och ett anrop till en metod, t.ex. `getStartMap()` (som returnerar kartan). Uppdaterat diagrammet enligt detta *<2019-02-18 Mon>*.

6 Klassdiagram

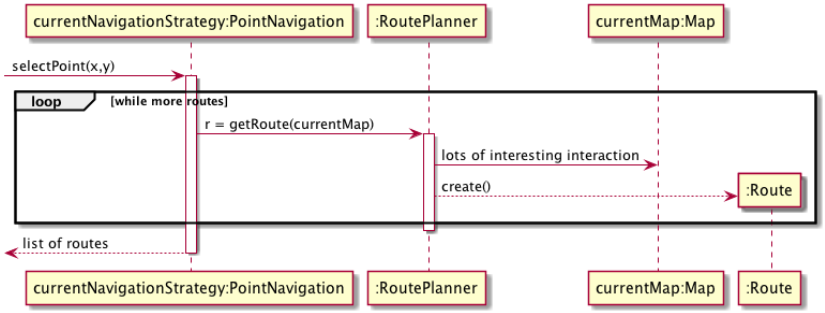
```
package Robot {

package ControlInterface {
```

```
class System {
selectNavigationMethod(theMethod)
NavigationStrategy* currentNavigationStrategy
}

class PointNavigation {
getStartMap()
Point currentCoordinates
}

abstract class NavigationStrategy
NavigationStrategy <|-- PointNavigation

System o- NavigationStrategy
}


package Navigation {
GPS : getCurrentCoordinates()
MapCollection : getMap(currentCoordinates, zoomLevel)

class Map

PointNavigation - GPS
PointNavigation - MapCollection
MapCollection o-- "*" Map

}

package Steering {
}

package ArmControl {
}

ControlInterface - Navigation
ControlInterface - Steering
}
```



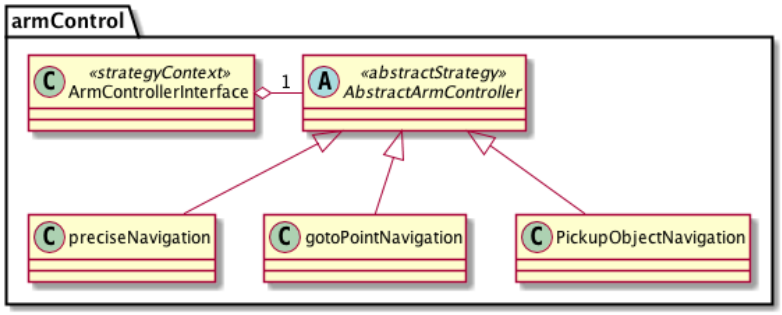
7 Sekvensdiagram - selectPoint

```
[-> "currentNavigationStrategy:PointNavigation" : selectPoint(x,y)
activate "currentNavigationStrategy:PointNavigation"

participant ":RoutePlanner"

loop while more routes
"currentNavigationStrategy:PointNavigation" -> ":RoutePlanner" : r = getRoute(currentMap)
activate ":RoutePlanner"

participant "currentMap:Map"
":RoutePlanner" -> "currentMap:Map" : lots of interesting interaction

create ":Route"
":RoutePlanner" --> ":Route" : create()
end loop

deactivate ":RoutePlanner"

"currentNavigationStrategy:PointNavigation" -->[ : list of routes
deactivate "currentNavigationStrategy:PointNavigation"
```



8 Styrning av Robotarmen

```
package armControl {

class ArmControllerInterface <<strategyContext>>
abstract class AbstractArmController <<abstractStrategy>>
ArmControllerInterface o- "1" AbstractArmController
AbstractArmController <|-- preciseNavigation
AbstractArmController <|-- gotoPointNavigation
AbstractArmController <|-- PickupObjectNavigation
}
```



9 Abstract Factory && Observer Pattern

```
package ObserverPackage {
abstract class Observer
abstract class Observable

Observer "*" - Observable

Observer : notify(Observable thePlaceWhereThingsJustHappened)
Observable : addObserver(Observer theObjectThatWanstToKnowWhatHappens)
}

package FactoryPackage {
buttonFactory : Button* getButton()
```

```
buttonFactory : setStrategy()
note right
void setStrategy(theStrategy) {
 myCurrentStrategy = theStrategy
}
end note

buttonFactory : enum myCurrentStrategy
buttonFactory - Button

Observable <|- Button
abstract class Button
Button <|-- RoundButton
Button <|-- HiddenButton
Button <|-- SquareButton
}

package RestOfTheSystem {

Observer <|- myFancyClassThatNeedsAButton

myFancyClassThatNeedsAButton : someMethod()
note left
void someMethod() {
//...
Button* aButton = new buttonFactory().getButton();
aButton->addObserver(this);
//...
}
end note
}
```