Labs 3 and 4 (UC Demonstrators)

Regulations as in previous lab.

Lab 3 – Problem Description

The lab 3 goal is a valid detailed software design for a prototype version of a **UC-demonstrating software** for the app (partners A of a quartet) and the robot (partners B), based on your analysis and derived use cases (UCs) in Labs 1/2. The software shall cover a ***UC cluster*** (typically related by include/extend, see below) that will end up in a ***domain model of interrelated concepts*** (prototypic classes, see lecture 4b and Ivan Marsic's book chapter) as the 1st step for lab 3 preparation. As a 2nd step for lab 3 try to achieve detailed design to include methods into your prototypic classes and a validated class diagram.

For lab 3 preparation, within your quartets, partners A of the two teams (and partners B correspondingly) will check the overlaps in their UC-diagrams, event flows in detailed UC descriptions and activity diagrams to start their common domain analysis phase. Once you find discrepancies, gaps, fragments or mistakes, try to fix them and incorporate your final cleaned items into your common report. There are different UC clusters in both the app and the firmware. So, different quartets will discover different UC clusters within their lab 2 work. Then, derive a domain model using analysis and decomposition via responsibility-driven design of your UC cluster as shown in lecture 4b.

Recommendations: Check UC clustering in the tracability matrix (2) of lecture 4b slide set (UCs 1, 2 and 7 vs. UCs 3-6 and 8) and see how the UCs of the UC cluster are interrelated in the UC-diagram (and activity diagram) of lecture 2b slide set. Analyze which UC cluster your detailed UC descriptions and corresponding activity diagram of lab 2 belongs to and possibly refine your UC diagram accordingly. Then follow the steps made in lecture 4b slide set, i.e. identifying concept responsibilities, associations and properties. Pure paperwork would be possible, but I'd strongly recommend use of BlueJ as shown in the lecture since then concept responsibilities, associations and properties find their way into class comments /** … */ and fields. For tables of the report, it is easy to extract required information from your prototypic code. Also incorporate your code as a subdirectory into your gitlab report directory!

For lab 3 work, model the dynamic behavior with (a) sequence diagram(s) in order to refine the domain model and derive a valid class diagram. So:

- Prerequisites and derivation of a **domain model** → lab 3 preparation,
- derivation of a **validated class diagram** → lab 3 work.

Lab 4 – Problem Description

The final goal is to design, implement and test, for the app (partners A) and the robot (partners B), a **UC-demonstrator program** similar to the BlueJ application prototypes distributed with the design patterns lectures. Choose UC and patterns such that the demonstrator obeys the following conditions:

- It demonstrates functionality of one of the UC clusters (with included/extended UCs)
- It usefully applies three design patterns, i.e. the design patterns must be a suitable tool according to the list of varying aspects, see design-patterns lecture slide sets.

For lab 4 preparation, identify for your UC-demonstrator **three suitable patterns,** one out of the patterns listed under items 1, 2 and 3 in the following list

1. Observer pattern (present in most UC clusters)
2. strategy pattern, decorator pattern or command pattern,
3. factory pattern or singleton

Suggestions for use of 8 of the classical GoF Design Patterns

From discussions in the second design patterns lecture the following list resulted:

1. Iterator: Iterating trees if in your design at all.
2. Strategy: Think of variable algorithms in your UC designs!
   - A good example in the **robot** is the UC-dependent path planning strategy with different algorithms for
     I. direct route to a destination, e.g. docking station, possibly A* path search algorithm[1]
     II. area covering cleaning route planning possibly using wall-following algorithm[2]
     III. exploration route generation in case map is unavailable, using a SLAM algorihmt[3]
3. Decorator: Where are additional properties assigned?
   - A good example in the **app** is that user can decorate the map with
     I. user-defined regions for UC RegionCleaning
     II. room selection (if auto-segmented) for UC RoomCleaning
     III. floor property for wet vs. dry cleaning in all cleaning UCs
4. Observer: Who may observe a state-changing subject? App and Robot examples:
   - **Robot:** Observable subject is the robot's changing state e.g. location, battery, time etc., Observers: path planning component and the app.
   - **App:** Observable subject is schedule data that can be changed by the user (control component of MVC-pattern), Observers will be display component and robot scheduler via communication boundary
5. Composite: Do you need a tree in your UC design at all?
6. Singleton: Which are classes with max. one instance?
   - Robot platform: database, BMS, Navigation & Drive (by API)
   - App: Display
7. Factory: Think of instantiation of subclasses in your design!
   - Cleaning mode instantiation (App), notification creation (Robot)
8. Command: Where separate command parameters from invoking the command?
   - Cleaning invocation after mode-specific map decoration.

[1] https://en.wikipedia.org/wiki/A*_search_algorithm
[2] https://en.wikipedia.org/wiki/Maze-solving_algorithm
[3] SLAM: Simultaneous Localization And Mapping, see MS teams Q&A sections