Software Quality Engineering, 2018-2019
Homework #1: UML software modeling
Deadline: Monday, November 19th, 2018

- SMART FLOOR -

TECHNOLOGY OVERVIEW

A smart floor is composed by a set of smart squared tiles, each embedding a pressure sensor equipped with low-level APIs for performing basic primitive operations. In particular, the weight applied onto a tile can be calculated from a measured pressure value.
A smart floor is logically partitioned into areas; a floor area may involve one or more smart tiles and, for sake of simplicity, all the areas of the same floor involves the same number of tiles.
By exploiting intra- and inter-area communication and orchestration, the smart floor is able to perform smart inferences, e.g. walking (speed) detection and shapes approximation, based on measured weights and their variations.
Note that, the higher the resolution of the floor is, the finer-grained the inferences could be.

POTENTIAL APPLICATION DOMAINS

The "smart floor technology" has a large number of potential applications, both in private and public environments, such as private homes, malls, museums, etc.
In the following, a non-exhaustive list of possible applications is provided:

Application domain	**Description**
(Security) Thieves/intruders detection and alarming	The goal of the system is to detect intruders (e.g. thieves) and proactively raise proper alarms. In a basic use case of the system, when a thief tries to steal a tv on a piece of furniture or a shelf, the system detects a (unexpected) walk to the corresponding area followed by a weight loss and a subsequent walk far from that area, so it can raise an alarm.
(Safety) Domestic accidents detection and alarming	The goal of the system is to detect falls to the ground (i.e. the smart floor), e.g. of occupants and/or other entities in the room, to interpret such fall(s) and proactively raise alarms, if needed. In a basic use case of the system, when an occupant (e.g. an old person with moving problems) falls to the ground, the system detects a variation in the weight distribution of such person and is able to evaluate such fall and possibly raise an alarm.
(Retrieval) Finding lost objects	The goal of the system is to help users to search for objects that are currently lost. In a simplified basic use case of the system, when the user thinks he have lost an object, he/she can query the system in order to get information (e.g. in the form of maps and/or augmented reality) about the status of the floor, such as current weights, possible weight deviations with respect to standard situations, weight redistributions that might have occurred recently.

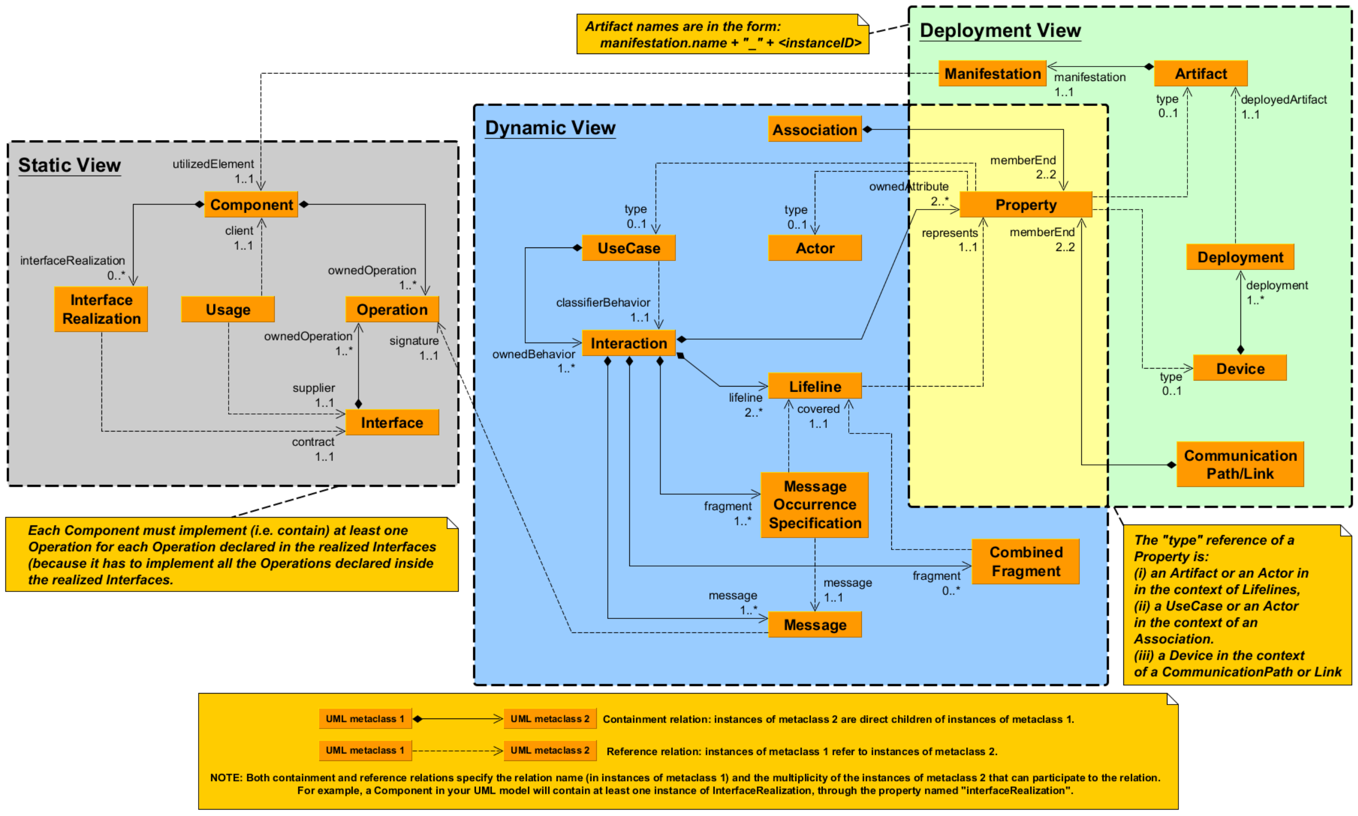
GUIDELINES FOR HOMEWORK DEVELOPMENT

In order to complete your homework you have to perform the following tasks:

- *Choosing one of the potential smart floor application domains above or proposing a different one*. It is suggested to figure out and model the following three macro-layers of the system, which may be distributed among one or more sub-systems, i.e.:
    1. Front-end: it aggregates all the system boundary components, such as sensors and/or apps of smart floor users.
    2. Control: it aggregates the components that deal with requests from both front- and back-end, that process them and perform inferences from which proactive actions (e.g. raising and managing alarms) may be devised and put in place.
    3. Back-end: it aggregates all the interfaces of back-end components (see hint here below), as well as the knowledge repositories (i.e. databases) and corresponding interfaces to access them.

Hint : We strongly suggest to dive your system into an application domain which involves one or more explicit back-end roles/entities, such as municipalities. For example, a municipality might be interested in increasing the citizens' security, hence it might be figured out a security scenario, like the one described above, where a municipality (at the back-end) receives intrusion alarms from their citizens' homes and is able to properly plan and execute intervention(s). This example can be easily modified to address public environments such as shops.

- *Building a UML profile for the considered application domain*, containing at least 5 stereotypes, corresponding tagged values, and (where needed) relations with other stereotypes.
- *Modeling the static, dynamic and deployment views of system*, according to the following figure, which defines the UML subset to be considered. Particular attention is required to ensure model consistency (i.e. arrows that, in the figure below, connect the different views of the system).



Artifact names are in the form: manifestation.name + "_" + <instanceID>

Each Component must implement (i.e. contain) at least one Operation for each Operation declared in the realized Interfaces (because it has to implement all the Operations declared inside the realized Interfaces.

The "type" reference of a Property is:
(i) an Artifact or an Actor in in the context of Lifelines,
(ii) a UseCase or an Actor in the context of an Association.
(iii) a Device in the context of a CommunicationPath or Link

Containment relation: instances of metaclass 2 are direct children of instances of metaclass 1.

Reference relation: instances of metaclass 1 refer to instances of metaclass 2.

NOTE: Both containment and reference relations specify the relation name (in instances of metaclass 1) and the multiplicity of the instances of metaclass 2 that can participate to the relation. For example, a Component in your UML model will contain at least one instance of InterfaceRealization, through the property named "interfaceRealization".

- *Applying the defined profile to your system model*, while taking care of what follows:
  - stereotypes defined in the profile have to appear aside standard UML elements,
  - tagged values of the stereotypes have to be set, as well as
  - relationships among stereotyped model elements, conformingly to the defined profile.
- *Drawing several representations of the system views*, in the form of the following UML diagrams:
  - Use Case Diagram (at least 2 types of actors and 2 use cases)
  - Component Diagram (between 5 and 15 components)
  - Sequence Diagrams (at least 1 diagram for each use case, i.e. the corresponding *classifierBehavior* in the figure above)
  - Deployment Diagram (at least 3 devices)

DELIVERABLES

1. One or two Eclipse project(s), developed with Obeo UMLDesigner or Papyrus, containing respectively the UML profile and the UML model, as two separate .uml files, and the created representations (i.e. diagrams).
2. A textual report (up to 20 pages).