Dipartimento di Elettronica e Informazione
Politecnico di Milano

prof. Elisabetta Di Nitto

20133 Milano (Italia)
Piazza Leonardo da Vinci, 32
Tel. (39) 02-2399.3400
Fax (39) 02-2399.3411

Software Engineering II

1 March 2013

Last Name	
First Name	
Id number (Matricola)	

Note

1. The exam is not valid if you don't fill in the above data.
2. Write your answers on these pages. Extra sheets will be ignored. You may use a pencil.
3. The use of any electronic apparatus (computer, cell phone, camera, etc.) is strictly forbidden.
4. You cannot keep a copy of the exam when you leave the room.

Question 1 Alloy (7 points)

A to-do list is composed of tasks. A task may have a precondition, which is a set of tasks that must be completed before the first task can be executed. A task also has an associated capability that must be possessed by the agent that may execute the task. Agents are the entities that may execute tasks; they are characterized by a set of owned capabilities.
A task of the to-do list can be executed if it satisfies a predicate CanExecute, which returns true if: the task precondition is fulfilled and a free agent possessing the required capability is available for the task.

1. Specify this informal description in Alloy: the required signatures, the facts and the predicate CanExecute.

2. Explain the purpose of each signature, attribute and fact.

Solution
(answer to point 2 is in the comment of the Alloy code)

```
// This and the sub-signatures represent the states of a task
abstract sig Status {}
sig Completed extends Status {}
sig OnGoing extends Status {}
sig Planned extends Status {}

// This represent the concept of cabality an agent should have to execute a task
sig Capability {}

// This is the to do list. It only contains a set of tasks
sig ToDoList {
tasks: set Task
}

// This is the task. It has a set of tasks as precondition, a status and a capability
sig Task {
precondition: set Task,
status: one Status,
capability: one Capability
}

// This is the agent. It has a set of capabilities and a set of assigned tasks
sig Agent {
capabilities: set Capability,
assignedTasks: set Task
}

// Tasks are assigned to agent owning the right capabilities
fact capabilityConsistent {
 all t: Task, a: Agent | t in a.assignedTasks implies t.capability in a.capabilities
}

// All tasks belong to one and only one to do list
fact allTaskInAtoDoList {
 all t: Task | one tdl: ToDoList | t in tdl.tasks
}
```

```
// Tasks that are precondition of another belong to the same todo list of this other
fact relatedTasksInTheSameToDo {
 all tdl: ToDoList | all t: Task | t in tdl.tasks implies (
    all t1: Task | t1 in t.precondition implies t1 in tdl.tasks)
}

// A task cannot be in its precondition nor in the precondition of one of its
// ancestors in the precondition relation
fact noCircularDepInPrecondition {
all t: Task | not t in t.^precondition
}

// A completed task cannot have an on going task in its precondition
fact noOnGoingTaskInPrecondOfCompletedTask {
all t: Task | t.status in Completed implies
   all t1: Task | t1 in t.precondition implies t1.status in Completed
}

// The task can be executed if it is not completed or already in execution and
// if all its precondition tasks are completed and if either some agent
// has been already assigned to it or if an assignable agent exists in our world
pred canExecute(t: Task) {
   t.status in Planned and
   all t1: Task | t1 in t.precondition implies t1.status in Completed and
   (some a: Agent | t in a.assignedTasks or
    some a: Agent | t.capability in a.capabilities)
}

pred show { #ToDoList > 1}

run show
run canExecute
```

Questions 2 Requirements Analysis (5 points)

A robot can rotate completely on its axis and can move forward at a speed variable between 0 and 3 km per hour. Moreover, it can monitor the presence of obstacles at 3 mt from its front part and can stop its movement within 2 mt when it is moving at the highest speed. Assume that you need to build the control software for this robot. This control software should ensure that the robot never breaks against an obstacle.
You should place the problem in the context of the Jackson & Zave framework for requirement analysis. More specifically, you should:

1) identify the relevant phenomena and place them in the appropriate domain;
2) determine which ones are shared and, of these, which ones are controlled by the machine;
3) formalize the goal of the control system;
4) define the domain properties;
5) define the requirement that descends from the goal defined at point 3.

Solution
1. The relevant entities are the robot and the obstacles. Associated to these entities we can identify the following phenomena

- SetRobotDirection: this is to define the rotation angle on the robot axis
- MoveRobot: this is to make the robot moving forward in the direction it is facing.
- VarySpeed(speed): this is to increase/decrease the speed of the robot between 0 and the maximum speed.
- ObstacleTrue: this is a signal that reveals the presence of an obstacle at 3 mt from the robot front part
- RobotSpeed: this is either acquired by the control software through some sensor on the robot or calculated directly by the control system depending on the commands it has previously sent to the robot.
- Obstacle: this represents the actual presence of an obstacle in the space
- RobotMoving: this is true if an external observer sees the robot moving

2. The first four phenomena are shared between the word and the machine. RobotSpeed it depends on the actual presence of the speed sensor. If this is the one providing the robot speed, then this is a shared phenomenon as well. If not, RobotSpeed is internal to the machine as it is to be entirely calculated within the control systems. In this case let's assume that the speed sensor exists, so RobotSpeed is shared. Obstacle and RobotMoving are located in the world and not shared with the machine.

The first three phenomena are controlled by the machine (the controller that we need to build). The remaining three are controlled by the environment. RobotSpeed is observed by the machine.

3. The goal of the software to be can be formalized as follows
ObstacleTrue => not RobotMoving

4. The domain properties are the following
Obstacle located at 3 mt from the robot ⇔ ObstacleTrue
No obstacles are dynamically placed at less than 3 mt from the robot during its movement
RobotMoving ⇔ RobotSpeed > 0
VarySpeed(0) => RobotSpeed = 0 (after a certain transitory that for the sake of simplicity we assume to be negligible)

5. Requirement: RobotSpeed > 0 && ObstacleTrue => VarySpeed(0)

Question 3 Software Architectures (5 points)

You are to design an e-commerce service for the distribution of movies in digital format. The user owns the movies (s)he buys, and can keep them stored on his machine(s) afterwards. The service does *not* stream the movies in real-time, but hands them over in digital format to the user.

The service needs to provide the following functionality:

- User registration, login, and logout;
- Search of movies;
- Purchase of movies;
- Distribution of the purchased movies to the user.

In the design of the ecommerce service the company expresses the following non-functional requirement:

- The usage of bandwidth at the company premises should be minimized.
- The cost of running the service should be kept low.

You are to design the high-level architecture for the service. This entails identifying the **key components** and the **architectural styles** you would apply to compose the components (you may also employ a combination of different architectural styles). In deciding what architectural styles to apply, you will also need to keep the non-functional requirements above into account.
Draw one (or more) UML diagrams of your choice to illustrate your solution, and *motivate your reasoning*!

Question 4 Verification (5 points)

Consider the following function, written in a C-like language where function `rand()` returns a pseudo-random (integer) number:

```
int foo() {

    int h,k;
    h = 0;

    for (int i = 0; i < 10; i++) {
         if (h > rand()) {
              k++;
         } else {
              h++;
         }
    }
}
```

You are required to:

- Identify the data flow graph for function `foo()` along with definition and uses of all variables involved;
- Provide an example of what kind of potential problem a typical verification effort using data flow analysis may discover in `foo()`.