Question on Alloy
The operating system kernel of a multiprocessor machine (with M CPUs) organizes the processes in three disjoint sets:

- RUNNING: the processes that are being executed by some CPU.
- READY: the processes that are ready to be executed.
- WAITING: the processes that are waiting for some event from some peripheral device.

Answer to the following points:

A) Model in Alloy this kernel defining proper signatures and facts. Assume that processes are modeled through this signature:

```
sig Process { }
```

B) Model the operation Resurrect that moves a process from WAITING to READY

C) Model also the following constraint:
The kernel should not waste computational resources, that is, if there are READY processes, there should not be any available CPUs (that is, a CPU that is not executing any process).

Solution.
Note: this solution is slightly different from the one provided in the exam this exercise belongs to (and we discussed during the last lab). In the original solution there was the following fact

```
fact {
  all disjoint k1, k2: Kernel | k1.ready+k1.waiting+k1.running & k2.ready+k2.waiting+k2.running =
none
}
```

that, as rightly observed by one of you at the end of the lab, is constraining too much our world. If we leave it in the Alloy specification then the Resurrect predicate will be inconsistent because it will not be possible to find two kernels, k and k', that share the same processes.

```
sig Process { }

sig CPU {
  busy: lone Process
}

sig Kernel {
  cpus: set CPU,
  ready: set Process,
  waiting: set Process,
  running: set Process
}
{
  no (ready & waiting)
  no (ready & running)
  no (waiting & running)
  all c: CPU | c in cpus implies (c.busy = none or c.busy in running)
  all p: Process | p in running implies (one c: CPU | c in cpus and p in c.busy)
}
```

```
fact doNotWasteCPUs {
  all k: Kernel | k.ready!=none implies no c: CPU | c in k.cpus and c.busy = none
}


pred Resurrect(k, k': Kernel, p: Process) {
  //precondition
  p in k.waiting

  //postcondition
  k'.cpus = k.cpus
  k'.ready = k.ready + p
  k'.waiting = k.waiting - p
  k'.running = k.running
}

pred show[] {}

run show
run Resurrect
```

Exercise on JEE

Consider the following scenario.

We need to develop an online betting application. The application allows the users to bet for the result of some sport event. Each user has a "virtual account" used to bet and to accumulate winnings.

The operations associated to deposit and withdraw of money on the virtual account are performed outside the system.

The sport events can be of various kinds, e.g., soccer match, Formula 1 race, …

The application should allow:

- System administrators:
  - To create sport events inserting the corresponding name (e.g., "soccer championship, match Roma-Napoli"), the bet subjects (e.g, "victory of Roma", "victory of Napoli", "draw result") and the multiplication factor associated to each subject. If a user bets on a winning subject, he/she receives on the virtual account an amount of money equal to the bet multiplied by the multiplication factor.
- Registered users:
  - To visualize the sport events, the bet subjects and the associated multiplication factors.
  - To bet selecting a specific event, subject and an amount of money. This amount is detracted from the virtual account.
  - To visualize the list of bets and the associated results as soon as they are available.

The outcome of sport events is received by the system through a web service offered by a sport official authority.

Based on the received outcomes, the application automatically calculates the winnings and assigns them to the virtual accounts of users.

You have been contracted to develop this system using JEE. In particular, you are requested to start developing the following elements:

- A simple entity representing sport events. It includes the event name, an ID and a map of strings and double that represents the set of subjects associated to the event.
- A bean that queries the database and offers the following operations
  - public SportEvent getFromName(String eventName)
  - public SportEvent insertEvent(SportEvent e)
  - public List<SportEvent> getSportEventList()

- A simple servlet that receives an event to be inserted in the database and call the insertEvent offered by the bean described above.

Entity

```
package it.polimi.classexample.entities;

import java.io.Serializable;
import java.lang.String;
import java.util.Map;
import javax.persistence.*;

@Entity
@Table(name = "SportEvent")
public class SportEvent implements Serializable {

   private String eventName;
   @ElementCollection
   private Map<String, Double> subjects;
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private int id;
   private static final long serialVersionUID = 1L;

   public SportEvent() {
      super();
   }

   public String getEventName() {
      return this.eventName;
   }

   public void setEventName(String eventName) {
      this.eventName = eventName;
   }

   public Map<String, Double> getSubjects() {
      return this.subjects;
   }

   public void setSubjects(Map<String, Double> subjects) {
      this.subjects = subjects;
   }

   public int getId() {
      return this.id;
   }

   public void setId(int id) {
      this.id = id;
   }

}
```

Stateless Bean

```
package it.polimi.classexample.beans;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.NoResultException;
import javax.persistence.PersistenceContext;
import javax.servlet.ServletException;

import it.polimi.classexample.entities.SportEvent;

/**
 * Session Bean implementation class SportEventBean
 */
@Stateless
public class SportEventBean {

   @PersistenceContext(unitName = "class-example")
   private EntityManager em;

   public SportEventBean() {

   }

   public SportEvent getFromName(String eventName) {
      try {
         return (SportEvent) this.em.createQuery("SELECT c FROM SportEvent c WHERE
c.eventName=:eventName")
               .setParameter("eventName", eventName).getSingleResult();
      } catch (NoResultException ex) {
         return null;
      }
   }

   public SportEvent insertEvent(SportEvent e) throws IOException {
      if (getFromName(e.getEventName()) == null) {
         em.persist(e);
         return e;
      } else {
         return null;
      }
   }

   @SuppressWarnings("unchecked")
   public List<SportEvent> getSportEventList() {
      try {
         return (List<SportEvent>) this.em.createQuery("SELECT e FROM SportEvent
e").getResultList();
      } catch (NoResultException ex) {
         return new ArrayList<SportEvent>();
```

```
    }
  }

}
```

Servlet

```
package it.polimi.classexample.servlets;

import java.io.IOException;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import it.polimi.classexample.beans.SportEventBean;
import it.polimi.classexample.entities.SportEvent;

/**
 * Servlet implementation class AdminServlet
 */
@WebServlet("/AdminServlet")
public class AdminServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;

  @EJB
   private SportEventBean sportEventService;
   /**
    * @see HttpServlet#HttpServlet()
    */
   public AdminServlet() {
      super();
      // TODO Auto-generated constructor stub
   }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
                ServletException, IOException {
        try {
          SportEvent e = new SportEvent();
          e.setEventName(req.getParameter("sportEventName"));
          if(sportEventService.insertEvent(e) != null){
            req.getRequestDispatcher("/adminHome.jsp").forward(req, resp);
          } else {
            req.getRequestDispatcher("/error.jsp").forward(req, resp);
          }
       } catch (IllegalArgumentException e) {
          req.setAttribute("error", e.getMessage());
          req.getRequestDispatcher("/adminHome.jsp").forward(req, resp);
       }
```

}

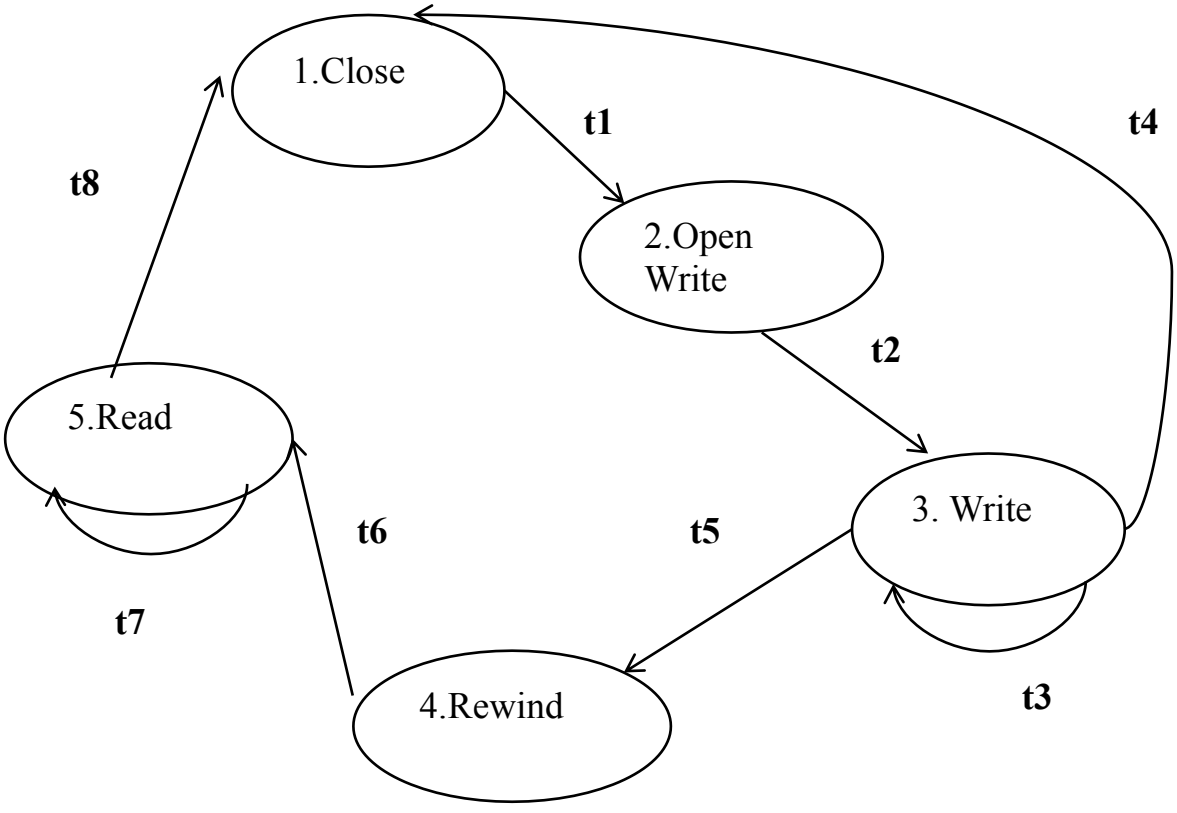
Question 3: Testing (6 points)
Consider the following specification of possible operations on sequential files.

To perform write (WRITE) operations on a given file, it is first necessary to open it (OPEN). After writing, to perform read (READ) operations, it is first necessary to carry out a rewind (REWIND) operation that moves the cursor to the beginning of the file.

Describe a possible approach to use this specification to derive test cases.
Add explicitly any hypothesis you think is important to complete the specification.

Solution

Given the description above, the approach to be used is black box testing.
Having the specification, it is possible deriving a state diagram describing the system behavior and then identifying the test cases.
The state diagram is illustrated below:



Test cases depend on the selected coverage criterion: coverage of states, or coverage of transitions.

In the first case, the test cases are (numbers refer to states):
TC1: 1 2 3 1
TC2: 1 2 3 4 5 1

In the second case:
TCa: t1 t2 t3 t3 t4

TCb: t1 t2 t4
TCc: t1 t2 t5 t6 t7 t8
TCd: t1 t2 t5 t6 t8

Given the above states, we could also test the system by trying to enforce transitions that should not be allowed. For instance, the sequences 1, 2, 4 and 1, 5, 4.

Question on Alloy
We want to model the behavior of a component that migrates data from a source storage to a destination storage. Data is divided in chunks that are migrated independently from each other and possibly in parallel to improve performance (independence and parallelization of chunk migration is not to be handled in this exercise).
Each chunk can be in one of the following states: ready to be migrated, under migration, migrated.
Of course, the whole migration ends when all chunks in a data set are in the migrated state.
Queries are related to specific chunks of data in a one-to-many relation. These queries can be potentially executed on the chunks when these last ones are still to be migrated or when they are migrated, while they have to be postponed when one of the involved chunks is under migration.

Define an Alloy model for the above problem. Focus on the definition of the signatures and facts that describe the constraints described above.

Solution

```
sig Storage {}
sig Data {}

abstract sig ChunkStatus {}
one sig Ready extends ChunkStatus {}
one sig UnderMigration extends ChunkStatus {}
one sig Migrated extends ChunkStatus {}

abstract sig QueryStatus {}
one sig Executable extends QueryStatus {}
one sig Postponed extends QueryStatus {}

sig MigrationStatus {}
one sig Stopped extends MigrationStatus {}

one sig MigrationSystem {
  status: MigrationStatus,
  source: Storage,
  destination: Storage,
  dataSet: set Data
}

sig Chunk {
 status: ChunkStatus,
 data: set Data
}{status = Ready || status = UnderMigration || status = Migrated}

sig Query {
  status: QueryStatus,
  chunks: set Chunk
}
```

```
//Data in chunks belong to the set of data managed by the migration system
fact allDataInChunk {
  all d: Data | d in MigrationSystem.dataSet and d in Chunk.data
}

//Different chunks include different sets of data
fact disjointChunks {
all disjoint c1, c2: Chunk | c1.data & c2.data = none
}

//If the migration is terminated, all corresponding chunks are in the Migrated state
fact migrationEnd {
  all c: Chunk | (c.data in MigrationSystem.dataSet and MigrationSystem.status = Stopped)
implies c.status = Migrated
}

//if the status of a chunk is UnderMigration, the status of a query working on it should be Postponed
fact queryPostponedUnderMigration {
  all q: Query, c: Chunk | c in q.chunks and c.status = UnderMigration
implies q.status = Postponed
}

pred show{}

run show
```

Question on testing

Design a simple project management tool for a company. The application should:

1. Handle information about projects (name, purpose, budget, tasks) and people (name, skills, employee number, role, department, allocated projects, allocated tasks).
2. Enable people to log-in/out.
3. Allow people with role "project manager" to allocate other people to projects and tasks.
4. Allow people to visualize the tasks they are working on and input the level of accomplishment of these tasks.
5. Compute the set of people allocated to a specific project/task.
6. Support people in requesting help across the company based on skills (the user selects a set of skills and the system suggests people with these skills.

Identify possible acceptance test cases for the system focusing on functional aspects. While defining these cases specify the preconditions (if any) that should be true before executing the tests, the inputs provided during the test, and the expected outputs. Finally, explain why you think the selected tests are important for the specific system being considered.

Solution

1. **Project staff allocation:**
   - PRECOND – at least one project is inserted and at least one task is ready to be allocated to at least one developer; also, at least one project manager is allocated to the project;
   - INPUTS – project manager inputs the name of the person or skill needed for the task and should be able to specify the allocation;
   - OUTPUTS – system returns a new team for the project that takes the allocation into account;
2. **Task status visualization and update**
   - PRECOND: at least one developer is working on a task X to which he was allocated; system is currently showing old status or <no-status> for task X;

- INPUTS: task X.ID is used to retrieve the status related to the task and the person responsible for it, if the person coincides with the requestor then modification should be permitted;
- OUTPUTS: a new task status should be presented via the system;

3. **Login-logout**
   - PRECOND: system is live; security check component is live;
   - INPUTS: employee credentials;
   - OUTPUTS: visualization of employee tasks and task status;
4. **Computation of the project team**
   - PRECOND: there is at least one project with at least one task and at least one task allocation per tasks present;
   - INPUTS: none;
   - OUTPUTS: visualization of a graph with team members and tasks allocated to them;
5. **Look for technical support / help**
   - PRECOND: task-allocations have been done, project is started (i.e., task status updates are present and progress can be computed)
   - INPUT: “need-help-on-task” request;
   - OUTPUT: list of possible candidates for lending a helping hand, matched based on skills and current availability;

These tests are critical since they ensure that key functionalities of the core business logic to be exhibited by the system are actually taken care of. User-acceptance tests performed around said functionalities will make sure that the system performs its intended function in the way the user expects it to. Also, in light of the fact that a good test is a test that fails, tests rotating around the functionalities above will reveal key shortcomings in the design and implementation of the system-to-be.