Main idea of Algorithm

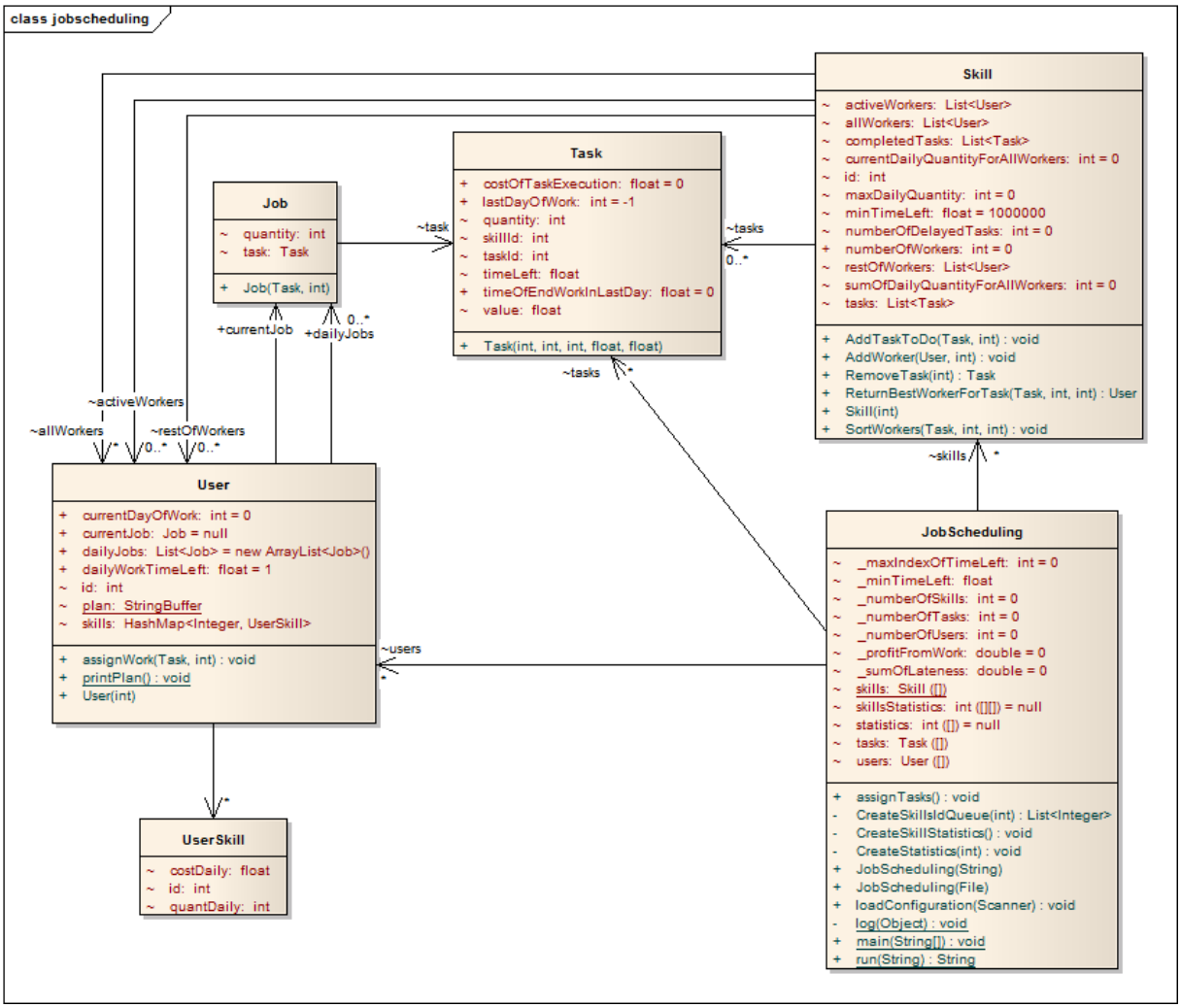
At the beginning of algorithm execution, all tasks were grouped into subgroups (skill object) . Each group contains tasks, which require the same skill. Tasks are sorted by time left. Each skill object contains also list of users, who have appropriate skill.

To get best results, I have applied following steps.

At first delayed tasks are done. All delayed tasks are sorted – the first is the task, which has the smallest quantity. A given task is done by many users in parallel. In this way average lateness is minimized.

Then tasks with the lowest value of time left are selected. Users whose work is the cheapest, are assigned to do the task (users, who have bigger coefficient: quantDaily/costDaily). If user is not able to finish task before deadline (before time left), next user is assigned to do the work. Usually only tasks, which value of time left is lower than 1, need to be done. Sometimes it is necessary to do the tasks that should be finished in the further future. For example: suppose that we have 10 users with the same skill, for the first 5 days only 3 users have to finish tasks on time (only 3 users are working), but in 6th day we have so much work that all users are not able to finish all tasks. This situation should be predicted – for the first 5 days also some part of work scheduled for day 6 should be done. To avoid similar situation at the beginning of each iteration (at the beginning of each day), for each skill object daily quantity of work is calculated.

Class diagram



User class:

- **currentDayOfWork** – this variable should have the same value as dayIndex;
- **dailyWorkTimeLeft** = 1 – (time of work in currentDayOfWork);

Skill class:

- **tasks** – tasks to do (unfinished tasks), that require considered skill;
- **currentDailyQuantityForAllWorkers** – at the beginning of each iteration value of sumOfDailyQuantityForAllWorkers is assigned to currentDailyQuantityForAllWorkers; this value is decreasing during algorithm execution;
- **sumOfDailyQuantityForAllWorkers** – maximal number of quantity that all users (related with considered skill) can do during one day;
- **maxDailyQuantity** – quantity of tasks (related with considered skill), that can be done during current day ; this variable is calculated at the beginning of each iteration (at the beginning of each day) and is decreasing during algorithm execution;
- **minTimeLeft** - minimal task timeLeft of all tasks related with considered skill - this value is set at the beginning of each iteration;
- **numberOfWorkers** – number of users related with considered skill;

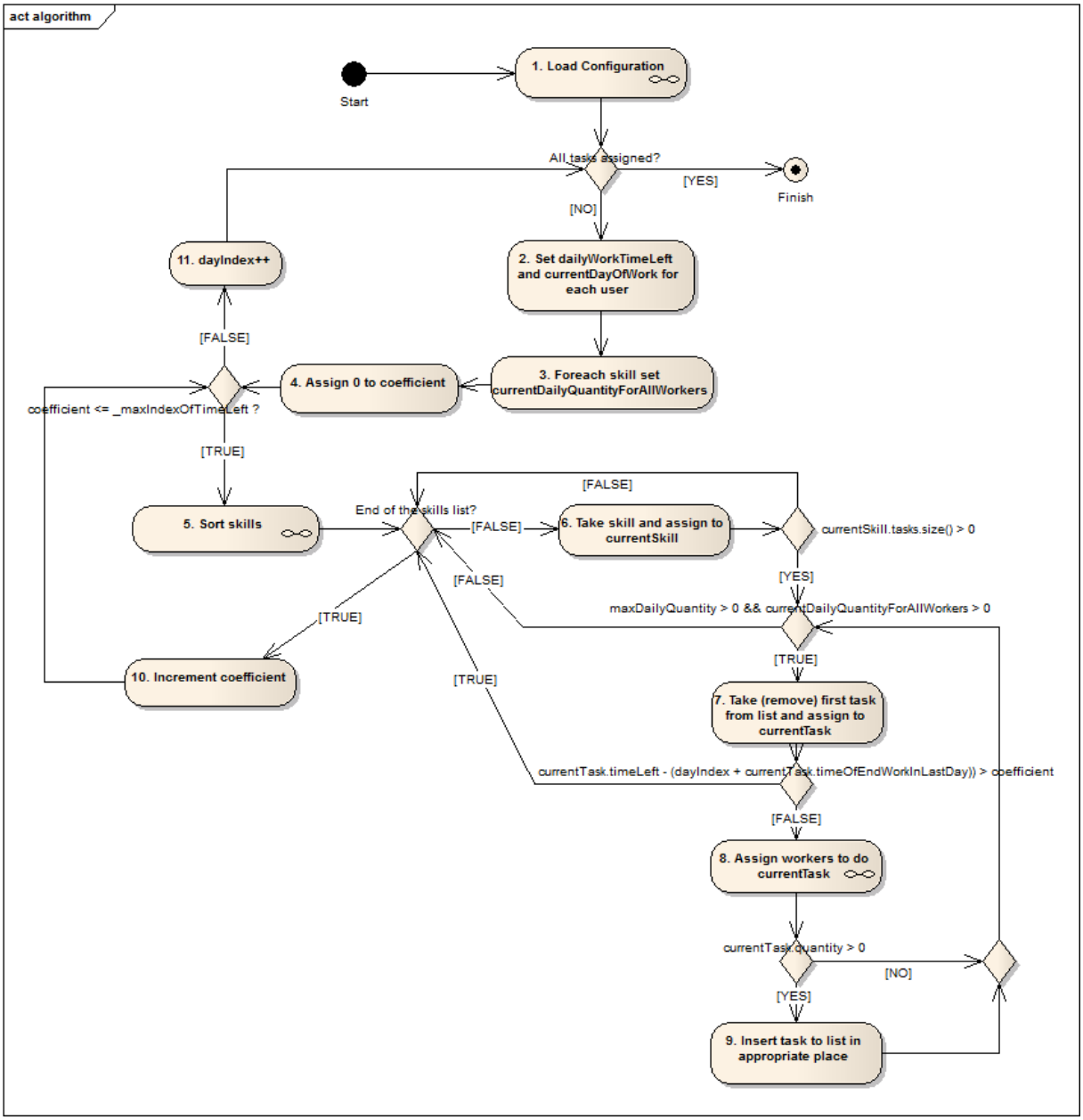
- **allWorkers** – all users related with considered skill; sorted list – the first one is user, whose work is the cheapest (has bigger coefficient: quantDaily/costDaily);
- **activeWorkers** – users from allWorkers, who can be assigned to do tasks (sorted list);
- **restOfWorkers** – allWorkers \ activeWorkers (users from allWorkers but not contained in activeWorkers list);

JobScheduling class:

- **skillsStatistics[s][d]** – sum of quantity of all tasks for skill s on day d (if task t should be finished on day d, quantity of task is added to: skillsStatistics[s][d], skillsStatistics[s][d+1],…, skillsStatistics[s][_maxIndexOfTimeLeft]); skillStatistics are used to calculate maxDailyQuantity**;**
- **statistics[s]** – sum of quantity of all tasks (for skill s), which are delayed;
- **_maxIndexOfTimeLeft** – max. task timeLeft of all tasks ((int)task.timeLeft +1);
- **_minTimeLeft** – min. task timeLeft of all tasks;

Schema of algorithm

This schema shows only the main idea of algorithm. Many important details are included in code.



1. Load Configuration

During configuration loading list of skills is created. Each task is added to appropriate skill object. Each user is added to appropriate skill objects (if user has 3 userSkills, he is added to 3 skill objects) – user is inserted into allWorkers list in appropriate place;

2. Set dailyWorkTimeLeft and currentDayOfWork for each user

dailyWorkTimeLeft =1; currentDayOfWork = dayIndex;

3. For each skill set currentDailyQuantityForAllWorkers

currentDailyQuantityForAllWorkers = sumOfDailyQuantityForAllWorkers;

4. Assign 0 to “coefficient”
5. Sort skills

Before execution of skill sort procedure, skillStatistics and statistics tables are created. In this moment maxDailyQuantity for each skill is calculated.

y, z – skill objects;
y_1 - minTimeLeft for skill y;
y_2 - statistics[y];
z_1 - minTimeLeft for skill z;
z_2 - statistics[z];

$$R = \{(y,z)\ \epsilon\ X \times X; y_1 < z_1 \vee\ (y_1 = z_1 \wedge y_2 < z_2)\}$$

If relation R contains (y,z) it means that y is "better" than z.

6. Take skill and assign to currentSkill
7. Take (remove) first task from the list and assign to currentTask
8. Assign workers to do currentTask

In this moment activeWorkers and restOfWorkers lists are created. If (task.timeLeft - dayIndex) <= 1, all users from allWorkers list are assigned to activeWorkers list, else only "the cheapest" users are considered. Users in activeWorkers and restOfWorkers lists are sorted due to specific criteria. For more details see function in code: public void SortWorkers(Task task, int dayIndex, int quantityToDo)

9. Insert task to list in appropriate place
10. Increment coefficient
11. dayIndex++