Project Allocation with Event-B and ProB

Thai Son Hoang[0000−0003−4095−0732], Abdolbaghi Rezazadeh[0000−0002−0029−469X], and Michael Butler[0000−0003−4642−5373]

School of Electronics and Computer Science (ECS), University of Southampton, U.K.
{t.s.hoang,m.j.butler}@soton.ac.uk, ra3@ecs.soton.ac.uk

Abstract. This short paper presents a formal development in Event-B for allocating student projects. Using Event-B as the modelling language, we precisely specify the requirements of the allocation problem and develop an algorithm satisfying the requirements. We then use ProB to "execute" the Event-B specification to perform the allocation. The formal model and the ability of ProB to execute the formal model help us to ensure the fairness of the allocation process with the possibility of extending the algorithms to consider further requirements.

Keywords: Event-B · ProB · Project Allocation

1 Introduction

Annually, we must allocate our project students (i.e., Year 3 or MSc students) to potential supervisors. This is a challenging and cumbersome task for several reasons: (1) the continual growing of the number of students, (2) each staff member can supervise multiple students, (3) student belong to a specific programme must work on a project relevant to the programme, (4) the allocation has to consider staff's loading constraints, (5) the allocation will try to balance the staff's load as much as possible (e.g., it is undesirable to have a staff with 3 students while another staff has no students).

To facilitate the allocation process, a separate system (called "Choice") was developed for the students to input their project preferences. Each staff will add a list of sample project topics to the *Choice* system (together with the programmes that are relevant for those projects). The students can then select (up to 12) options from the systems as their preferences. These preferences are then used as the input for the allocation process, which is the discussion of this paper.

Previously, an existing software was used to automate this project allocation process. However, in recent years, the software performance does not match the expectation as the new programs are added and the cohort's size increases. This often results in some manual allocation process, which is (unsurprisingly) time-consuming and unreliable. As this software is no longer maintained, we develop a *formal specification that can be used for allocation directly, but also adaptable to future changes in the allocation process.*

The remainder of the paper is structured as follows. Section 2 discusses the background on matching algorithms and the Event-B formal method. Section 3 then presents an allocation algorithm and its formalisation. Section 4 gives some evaluation of the result of the allocation. Section 5 summarises and discusses future work.

2 Background

2.1 Matching Algorithms

Matching problems and algorithms are well-studied topics in computer science. One of the most well-known matching algorithms is the Gale–Shapley algorithm for the stable marriage problem [3]. A generalised version of the stable marriage problem is the college admissions problem [3]. Here, the matching is done between applicants and colleges, where each college can accept multiple applicants up to a certain limit. The algorithm involves several rounds, each round contains a *proposal* phase and an *acceptance* phase.

proposal : Each unallocated applicant applied to the most-preferred college to which they have not yet applied.

acceptance : Each college with quota q will (tentatively) accept the top-q applicants (or all applicants if the number is less than q), amongst the new applicants and the applicants that they have (tentatively) accepted, and reject the rest. Notice that potentially, some already accepted students might be rejected if the college prefers some new applicants.

The process is repeated until every student is on the acceptance list or has been rejected by all colleges to which they applied.

There are similarities and differences between the college admissions problem and our project allocation problem.

- In the college admissions problem, both applicants and colleges have a preference list of each other. In our project allocation problem, the students has the preference list of the staff (by ranking the topics proposed by the staff), but the staff do not rank the students.
- In the college admissions algorithm [3], there may be some applicants who will not be allocated due to their low ranking from the colleges. For our project allocation problem, the goal is to allocate all students to a supervisor.
- In the college admissions algorithm, applicant has to give their preferences. We have to handle a small number of students who have not entered their preferences.
- In both problems, there are limited capacities for colleges or staff. However, in the college admissions algorithm, there are no considerations for load balancing amongst the colleges. We have to balance the staff loading to ensure that the allocation of students is fair. We consider this an extended goal for future work.

2.2 Event-B Modelling Method and Tools

Event-B [1] is a state-based formal modelling method based on first-order logic and set theory. An Event-B model contains *contexts* and *machines*. Contexts specify the static part of the model, including carrier sets (types), constants, and axioms that constrain them. Machines specify the dynamic part of the model and include variables, invariants, and events. An Event-B machine corresponds to a discrete transition system, where the states and transitions are represented by variables and guarded events, accordingly.

Rodin [2] is an Eclipse-based tool that supports the Event-B modelling language. Verification of the consistency of the Event-B models can be done by proving the generated obligations using theorem provers or by model checking. We also utilise here an extension of Rodin called CamilleX [4] to support textual input for Rodin.

ProB [5] is a model checker for Rodin. ProB uses constraint solving to analyse the model. Furthermore, we can also validate the Event-B models by animating the models with ProB. This paper also utilises this feature of ProB for "executing" the project allocation algorithm.

3 Formal Development for Project Allocation

3.1 Requirements for Project Allocation

We assume that we have a set of programmes (ASM 1). Furthermore, each student is associated with exactly one programme that they are studying, however each staff can supervise projects in different programmes.

ASM 1 There is a finite set of programmes

ASM 2 There is a finite set of students

ASM 3 There is a finite set of staff

ASM 4 Each student is associated with a programme

ASM 5 Each staff is associated with a set of programmes

We assume that prior to our allocation, we have information about the student preferences of the staff (we omit the information about the topics here).

ASM 6 Students have a preference ranking (without duplication) of the supervisors

Each staff has a certain capacity for supervision, indicating the maximum number of students that they can supervise.

ASM 7 Each staff has a maximum number of students that they can supervise

Evidently, the project allocation cannot be guaranteed to be successful, e.g., when there are insufficient staff. However, we have the following requirements for allocations.

REQ 8 A successful allocation must ensure that every student is allocated to a supervisor.

REQ 9 A student's programme must match one of the supervisor's indicated programme

REQ 10 If a student has some preferences, then the allocated supervisor must be on their preference list

Notice that we omit the requirement about load-balancing here because it conflicts with the above requirements. Nevertheless, our algorithm will try to allocate as many students as possible.

3.2 An Algorithm for Project Allocation

As discussed in Section 2.1, we need to adapt the existing algorithm for our project allocation problem. The algorithm contains three stages.

Greedy Allocation Stage : In this stage, we use a modified version of the college admissions algorithm to allocate the students to their preferred supervisor. The process also contains two phases, i.e. proposal and acceptance, but there will be no rejection of already allocated students here.

proposal phase : Each unallocated student applied to the most-preferred supervisor to whom they have not yet applied.

acceptance phase : Each supervisor with capacity c will accept the $c - n$ new applicants, where n is the number of the current accepted students for that supervisor (or all new applicants if the number of them is less than $c - n$), and reject the rest. Notice that, once allocated, no students will be removed from their allocation in this stage (this is different from the college admissions algorithm).

The process repeats until either all students are allocated (successful allocation) or all the unallocated students' preferences have been taken into account. In the case of an unsuccessful allocation, we move to the Swapping Allocation Stage.

Swapping Allocation Stage : In this stage, we change the student allocation so that we can maximise the number of allocated students. This process involves an allocated student as and unallocated student us, satisfying:

- The allocated supervisor aS for as is on the preference list of us.
- as has another preferred supervisor pS that they have not yet applied to and pS still has a capacity for supervision.

In this case, as swaps aS for pS as the supervisor and aS will become the supervisor of us. We repeat this swapping until either all students are allocated (successful allocation) or we cannot find an allocated student to perform a swap for an unallocated student. Notice that at this point, any unallocated student with staff preferences will require manual allocation.

No-preference Allocation Stage : This stage tries to allocate the students without preferences to a supervisor. Here, the chosen staff will be a staff for the programme that the student studies and have the most capacity for supervision. The process finishes when all students are allocated (successful allocation) or if there are some unallocated students.

3.3 Formal Development

We give a brief overview of the formalisation of the problem and the algorithm using Event-B here.

The assumptions correspond to various sets and constants in different contexts, with appropriate axioms.

```
set PROGRAMME
axiom @axm1: finite(PROGRAMME) // ASM1
set STAFF
axiom @axm2: finite(STAFF) // ASM3
constant staff_programmes : STAFF ↔ PROGRAMME
axiom @axm3: dom(staff_programmes) = STAFF // ASM5
set STUDENT
axiom @axm5: finite(STUDENT) // ASM2
constant student_programme : STUDENT → PROGRAMME // ASM4
theorem @axm9: finite(student_programme)
constant student_preferences : STUDENT → (STAFF ⇸ ℕ) // ASM6
constant staff_limit : STAFF → ℕ // ASM7
```

Greedy Allocation Stage: This stage is modelled by events such as propose, accept, decline and phase/stage-transition events change_to_acceptance, change_to_propose, and move_to_swapping_stage. For example, event propose is specified as follows.

```
event propose
any student staff where
 @grd0: phase = Proposing
 @grd1: student ∉ dom(allocated) // student is unalocated
 @grd2: student ∉ dom(proposes) // student has not yet make a proposal
 // The staff is the top remaining choice for the student
 @grd3: staff ∈ dom(student_remained_choice(student))
 @grd4: ∀other_staff · other_staff ∈ dom(student_remained_choice(student))
    ⇒ student_remained_choice(student)(other_staff) ≤ student_remained_choice(student)(staff)
then
 @act2: proposes(student) := staff
end
```

Other events are omitted here due to space constraints.

Swapping Allocation Stage : This phase is modelled by a single event, namely swap corresponds to the algorithm in Section 3.2, with some stage-transition event move_to_no_preference_allocation.

```
event swap
any student staff allocated_student free_staff
where
 @grd0: process = SwappingAllocation
 @grd1: student ∉ dom(allocated)
 @grd2: staff ∈ dom(student_preferences(student)) ∧ staff ↦ student ∈ dom(staff_preferences)
 @grd3: allocated_student ∈ dom(allocated) ∧ allocated(allocated_student) = staff
 @grd4: free_staff ∈ dom(student_remained_choice(allocated_student))
 @grd5: staff_capacity(free_staff) ≠ 0
then
 @act1: allocated := allocated ⩤ {student ↦ staff, allocated_student ↦ free_staff}
 @act2: staff_capacity(free_staff) := staff_capacity(free_staff) − 1
end
```

No Preference Allocation Stage : This phase is modelled by a single event, namely no_preference_allocation corresponds to the algorithm in Section 3.2. We omit the details of the event here.

Finally, at any stage, if all students are allocated, then we consider the process finished successfully (REQ 8). Furthermore, the following invariants ensure the consistency of the immediate allocation, i.e., (REQ 9) and (REQ 10).

@inv1: $\forall$ student · student $\in$ dom(allocated) $\Rightarrow$ student_programme(student) $\in$ staff_programmes[{allocated(student)}]
@inv2: $\forall$ student · student $\in$ dom(allocated) $\wedge$ dom(student_preferences) $\neq \varnothing \Rightarrow$ allocated(student) $\in$ dom(student_preferences(student))

4 Evaluation

After developing the formal models specifying the algorithm, we utilise the capability of ProB to "run" the model on the actual data.

- We wrote a Python script to convert students' preferences and staff's programmes in CSV format to extended contexts to provide the actual values for sets and constants, e.g., STUDENT, STAFF, student_programme, etc. The output of the Python script is in CamilleX textual format for Event-B model.
- The algorithm will terminate (either successfully or unsuccessfully) when the formal model is deadlock. As a result, we use ProB to check for deadlock-freeness and the trace that ProB provided as a counter-example will essentially be the trace for the project allocation. In particular, we use ProB in depth-first search to avoid exploring the different traces of the model. This helps ProB to speed up considerably compared to the default search strategy.
- We then extract that last state of the ProB check to get the allocation data.

We used this approach for the allocation for 244 MSc students and 134 staff, each staff having a capacity of 3 students.

- After the Greedy Allocation Stage, 236 students were allocated.
- After the Swapping Allocation Stage, 5 more students were allocated
- After the No Preference Allocation Stage, 3 more students were allocated (all 3 students did not input their preferences).

5 Conclusion

This paper presents an approach for formally specifying an algorithm for project allocation and executing the formal model on actual data using ProB. The approach successfully allocates a large cohort of MSc students to one of their preferences. Compared to the previous year, when the allocation was done manually, we managed to release the project allocation two weeks earlier.

In the future, we want to use this approach as the core for the new project allocation software and extend the feature further (e.g., taking into account the topics). Taking into account the load-balancing constraint for the staff will require the relaxation of some constraints. At the moment, this process is done manually. Furthermore, we want to prove the properties of the algorithm, in particular, to discover the various invariants for proving the consistency of the formal model. Finally, on the tooling side, a more integrated approach with the Python pre/post-processing, Rodin static analyser, and ProB model checker is required.

References

1. Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International journal on software tools for technology transfer*, 12(6):447–466, 2010.
3. David Gale and Lloyd S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, January 1962.
4. Thai Son Hoang, Colin F. Snook, Dana Dghaym, Asieh Salehi Fathabadi, and Michael J. Butler. Building an extensible textual framework for the rodin platform. In Paolo Masci, Cinzia Bernardeschi, Pierluigi Graziani, Mario Koddenbrock, and Maurizio Palmieri, editors, *Software Engineering and Formal Methods. SEFM 2022 Collocated Workshops - AI4EA, F-IDE, CoSim-CPS, CIFMA, Berlin, Germany, September 26-30, 2022, Revised Selected Papers*, volume 13765 of *Lecture Notes in Computer Science*, pages 132–147. Springer, 2022.
5. Michael Leuschel and Michael Butler. Prob: an automated analysis toolset for the b method. *International Journal on Software Tools for Technology Transfer*, 10(2):185–203, 2008.