University of Southampton

Alin Formusatii (aaf1u20)
Bagir Bazarov (bb1u20)
Daniel Braghis (ddb1u20)
Logan Gibson (lg2n20)

# COMP1216. Software Modelling and Design (2021-22)

## Group 21: Vaccine Centre Modelling in Event-B

Submission date: 13 March 2022

## Contents

## 1 Introduction

Assignment summary

The objective is to create a formal modelling of a COVID Vaccination Tracking System. The system is designed to encompass the entirety of the journey from the creation of the vaccination centre to the administering the vaccination itself and giving the user a certificate as proof. This is broken down into four sections:
- The user management system.
- Vaccination centres.
- Appointment Booking and Re-booking
- Receiving vaccines

Task Distribution

Alin was tasked with creating the class diagram and assisted on sections 2 and 3.
Bagir assisted on sections 1 and 2 and consultant on code commenting.
Daniel was the main architect on sections 1, 2 and 3. Also tasked with code commenting.
Logan was tasked with reformulating the class diagram, and producing the LaTeX document.

## 2 Task 1. Class diagram

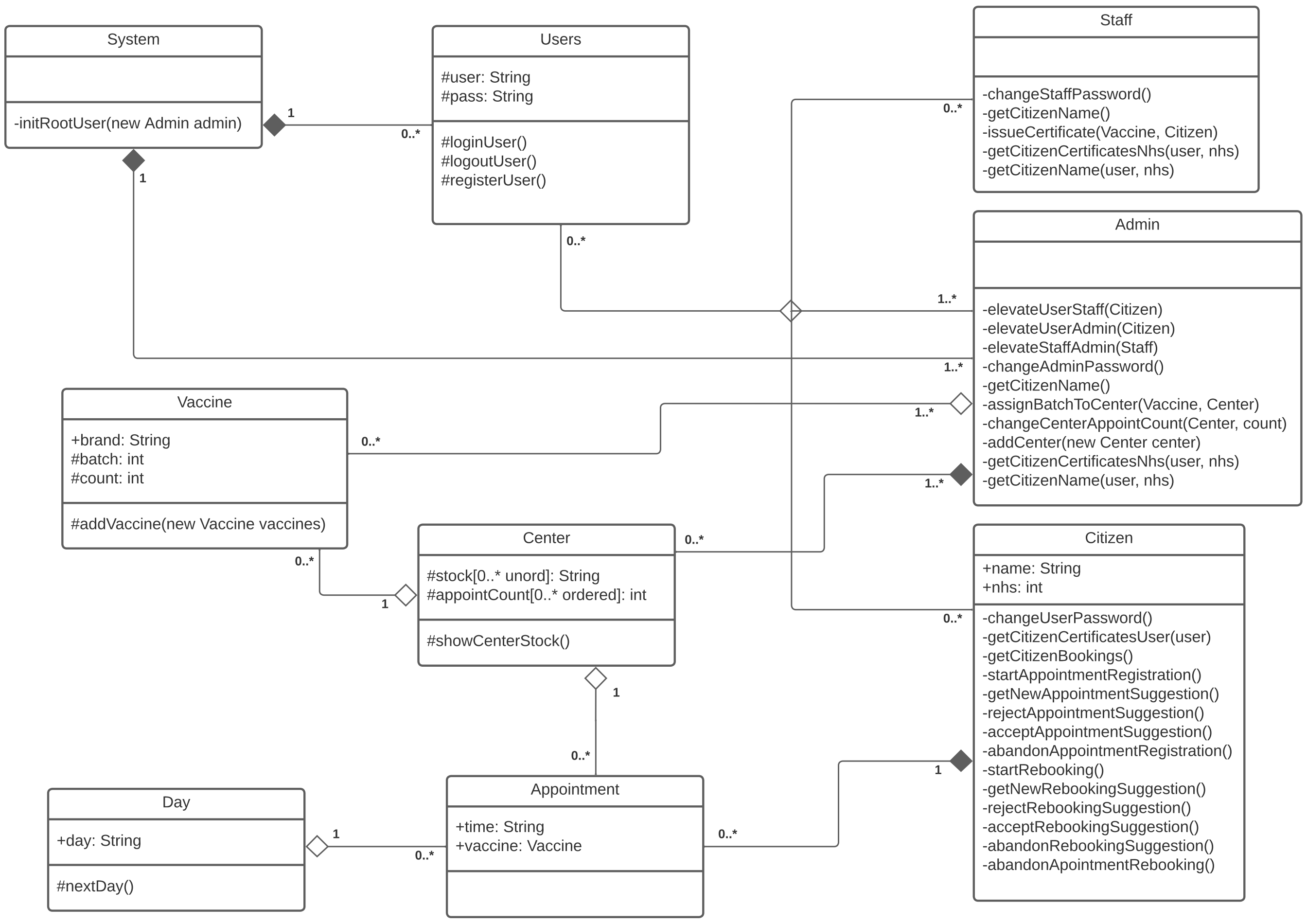
System
-initRootUser(new Admin admin)
Users
#user: String
#pass: String
#loginUser()
#logoutUser()
#registerUser()
Staff
-changeStaffPassword()
-getCitizenName()
-issueCertificate(Vaccine, Citizen)
-getCitizenCertificatesNhs(user, nhs)
-getCitizenName(user, nhs)
Admin
-elevateUserStaff(Citizen)
-elevateUserAdmin(Citizen)
-elevateStaffAdmin(Staff)
-changeAdminPassword()
-getCitizenName()
-assignBatchToCenter(Vaccine, Center)
-changeCenterAppointCount(Center, count)
-addCenter(new Center center)
-getCitizenCertificatesNhs(user, nhs)
-getCitizenName(user, nhs)
Vaccine
+brand: String
#batch: int
#count: int
#addVaccine(new Vaccine vaccines)
Center
#stock[0..* unord]: String
#appointCount[0..* ordered]: int
#showCenterStock()
Citizen
+name: String
+nhs: int
-changeUserPassword()
-getCitizenCertificatesUser(user)
-getCitizenBookings()
-startAppointmentRegistration()
-getNewAppointmentSuggestion()
-rejectAppointmentSuggestion()
-acceptAppointmentSuggestion()
-abandonAppointmentRegistration()
-startRebooking()
-getNewRebookingSuggestion()
-rejectRebookingSuggestion()
-acceptRebookingSuggestion()
-abandonRebookingSuggestion()
-abandonApointmentRebooking()
Day
+day: String
#nextDay()
Appointment
+time: String
+vaccine: Vaccine
1
0..*
1
0..*
0..*
1..*
1..*
0..*
1..*
0..*
1
1..*
0..*
0..*
1
0..*
1
0..*
1
0..*

Figure 1: Class Diagram

## 3 Task 2. Event-B model 1.1 User Management System

The user management system falls under 1.1 of the assignment. The objective was to create a set of registered users that could log in and out of a system using a password. Administrators and staff to be designed as special users where the system can only start with an administrator who is a root user.

**REQ 1:** *The system manages a set of registered users* - Here users(citizens), admins (administrators) and staff are all subsets of the USER type.

**REQ 2:** *The system should only allow a user to log in to the system using the correct password* - The system checks if the user belongs to the domain of registered users, checks that the password belongs to the registered user and ensure that the user does not belong to the domain of already logged in users.

**REQ 3:** *A logged-in user can log out of the system* - The user is checked against the domain of logged in users and that the password belongs to the user. If these conditions are met it logs the user out.

**REQ 4:** *There are three types of users: administrators, staff, and citizens* - The system has three user types as requested. As referenced in REQ 1, the users (citizens), admins (administrators) and staff are all subsets of the USER type.

**REQ 5:** *A user can only have one role in the system* - Each user (users, admins, staff) have a clearly defined role with separate capabilities within the system. This is checked by an invariant where the intersection of the users, admins and staff is equal to the empty set and modelled by removing and assigning the users to/from these sets.

**REQ 6:** *The root user is the initial administrator* - It was decided to use a separate event - InitRootUser to initialize the root user. The event is available only when there are no administrators in the system and initialises one.

**REQ 7:** *A logged-in administrator can register a new user for the role of administrator or staff*

The event ElevateUserStaff can register upgrade a user to a staff role. It verifies that the user belongs to the domain of administrators to permit the elevation, checks that the user to be elevated belongs to the domain of users and they are logged in. There is a union between the elevated_user and the staff and thereafter the elevated_user is removed from users as the upgrade to staff is complete.

There are two other events, ElevateStaffAdmin which elevates a staff member to an admin role, and ElevateUserAdmin which elevates a user to an admin role. These function similarly to the ElevateUserStaff event.

**REQ 8:** *A citizen can register an account with the system directly.* - The event RegisterUser permits for the user to register an account directly with the system. There are checks to ensure that the user does not belong to the domain of previously registered users and that the cardinality of the administrators is greater than zero (making sure the system is initialized first).

**REQ 9:** *A logged-in user can change their password* - The users, admins and staff have their own events to change their passwords. For the citizens it is ChangerUserPassword, for the staff the event is ChangeStaffPassword and for the administrators the event is ChangeAdminPassword. It was necessary to have different events for each to update the correct set in the event.

```
context UserManagementContext
sets
 USER
 PASSWORD
end
```

```
machine UserManagement
sees UserManagementContext

/∗ User Management machine that deals with user registration, login, roles and role management.
 ∗ Requirements implemented:
 ∗ REQ 1 The system manages a set of registered users
 ∗ REQ 2 The system should only allow a user to log in to the system using the correct password
 ∗ REQ 3 A logged−in user can log out of the system
 ∗ REQ 4 There are three types of users: administrators, staff, and citizens
 ∗ REQ 5 A user can only have one role in the system
 ∗ REQ 6 The root user is the initial administrator
 ∗ REQ 7 An logged−in administrator can register a new user for the role of administrator or staff
 ∗ REQ 8 A citizen can register an account with the system directly.
 ∗ REQ 9 A logged−in user can change their password
 ∗/

variables
 users
 admins
 staff
 login
 register

invariants
 /∗ Users can have the same passwords. A many to one relationship∗/
 @inv−user−password: users ∈ USER ⇸ PASSWORD
 @inv−admin−password: admins ∈ USER ⇸ PASSWORD
 @inv−staff−password: staff ∈ USER ⇸ PASSWORD

 /∗ Users, admins and staff are all part of the USER type∗/
 @inv−users−user: dom(users) ⊆ USER
 @inv−admins−user: dom(admins) ⊆ USER
 @inv−staff−user: dom(staff) ⊆ USER

 /∗ Users can have only one role ∗/
 @inv−types−no−intersection: users ∩ admins ∩ staff = ∅

 /∗ Registered users are a union of all roles ∗/
 @inv−login−registered: register = users ∪ admins ∪ staff

 /∗ Logged in users is a subset of registered users ∗/
 @inv−login−sub−registered: login ⊆ register

events

 /∗ Initialize all sets ∗/
 event INITIALISATION
 then
  @act−init−users: users := ∅
  @act−init−admins: admins := ∅
  @act−init−staff: staff := ∅
  @act−init−login: login := ∅
  @act−init−register: register := ∅
 end

 /∗ Initialize the root user as admin if there are no admins ∗/
 event InitRootUser
 any
  root
  pass
 where
  @grd1: root ∈ USER
  @grd2: pass ∈ PASSWORD
  @check−no−admins: card(admins) = 0
 then
  @assign−root: admins := {root ↦ pass}
  @assign−register: register := {root ↦ pass}
 end

 /∗ Register a new basic user ∗/
 event RegisterUser
 any
  user
  pass
 where
  @grd1: user ∈ USER
  @grd2: pass ∈ PASSWORD
  @check−unregistered: user ∉ dom(register)
  @check−nonzero−admin: card(admins) > 0
 then
  @add−register: register := register ∪ {user ↦ pass}
  @add−users: users := users ∪ {user ↦ pass}
 end

 /∗ Login a user ∗/
 event LoginUser
 any
  user
  pass
 where
  @check−registered: user ∈ dom(register)
  @check−password: pass = register(user)
  @check−not−loggedin: user ∉ dom(login)
 then
 @add−login: login := login ∪ {user ↦ pass}
end

/∗ Logout a user ∗/
event LogoutUser
any
 user
 pass
where
 @check−loggedin: user ∈ dom(login)
 @check−password: pass = login(user)
then
 @remove−login: login := login \ {user ↦ pass}
end

/∗ Change a user’s password ∗/
event ChangeUserPassword
any
 user
 pass
 new_pass
where
 @check−user: user ∈ dom(users)
 @check−password: pass = login(user)
 @new−password−type: new_pass ∈ PASSWORD
 @check−loggedin: user ∈ dom(login)
then
 /∗∗ Change the password for user everywhere ∗/
 @act1: login := login <+ {user ↦ new_pass}
 @act2: register := register <+ {user ↦ new_pass}
 @act3: users := users <+ {user ↦ new_pass}
end

/∗ Change a staff’s password ∗/
event ChangeStaffPassword
any
 user
 pass
 new_pass
where
 @check−staff: user ∈ dom(staff)
 @check−password: pass = login(user)
 @new−password−type: new_pass ∈ PASSWORD
 @check−loggedin: user ∈ dom(login)
then
 /∗∗ Change the password for user everywhere ∗/
 @login−change: login := login <+ {user ↦ new_pass}
 @register−change: register := register <+ {user ↦ new_pass}
 @staff−change: staff := staff <+ {user ↦ new_pass}
end

/∗ Change an admin’s password ∗/
event ChangeAdminPassword
any
 user
 pass
 new_pass
where
 @check−admin: user ∈ dom(admins)
 @check−password: pass = login(user)
 @new−password−type: new_pass ∈ PASSWORD
 @check−loggedin: user ∈ dom(login)
then
 /** Change the password for user everywhere */
 @login−change: login := login <+ {user ↦ new_pass}
 @register−change: register := register <+ {user ↦ new_pass}
 @admin−change: admins := admins <+ {user ↦ new_pass}
end

/* Elevate a user to staff role */
event ElevateUserStaff
any
 user
 pass
 elevate_user
where
 @check−admin: user ∈ dom(admins)
 @check−password: pass = login(user)
 @check−elevate−from−user: elevate_user ∈ dom(users)
 @check−loggedin: user ∈ dom(login)
then
 @add−to−staff: staff := staff ∪ ({elevate_user} ◁ users)
 @remove−from−users: users := users \ ({elevate_user} ◁ users)
end

/* Elevate a user to admin role */
event ElevateUserAdmin
any
 user
 pass
 elevate_user
where
 @check−admin: user ∈ dom(admins)
 @check−password: pass = login(user)
 @check−elevate−from−user: elevate_user ∈ dom(users)
 @check−loggedin: user ∈ dom(login)
then
 @add−to−admins: admins := admins ∪ ({elevate_user} ◁ users)
 @remove−from−users: users := users \ ({elevate_user} ◁ users)
end

/* Elevate staff member to admin role */
event ElevateStaffAdmin
any
 user
 pass
 elevate_user
where
 @check−admin: user ∈ dom(admins)
 @check−password: pass = login(user)
 @check−elevate−from−staff: elevate_user ∈ dom(staff)
 @check−loggedin: user ∈ dom(login)
 then
  @add−to−admins: admins := admins ∪ ({elevate_user} ◁ staff)
  @remove−from−staff: staff := staff \ ({elevate_user} ◁ staff)
 end

end


```

# 4 Task 2. Event-B model 1.2 Vaccination Centres & 1.4 Receiving Vaccines

Several parameters were given for section 1.2:
- An administrator must be able to create a vaccination centre.
- After the centre's creation any administrator can provide it with updates.
- These updates are vaccine stock and appointments.
- Vaccines must have brands and a belong to a unique batch number.
- Citizens must have a name, unique NHS number and receive a certificate after receiving their vaccination. - A vaccination centre has vaccine stock and a number of available vaccines.

From a standpoint of the system it was easier to fulfull REQ 30 and REQ 31 as part of the vaccination centre, as this is where the citizen both receives the vaccine and the certificate is administered.

For how each requirement was addressed please see below:

**REQ 10:** *A vaccine has a brand and a batch number* - The set vaccines define a batch number to brand relationship. The event addVaccine is created such that it checks if a brand is a valid brand and that the batch number to be added is unique. The batch must not be a member of the domain of pre-existing vaccines, this is how it remains unique. The event also assigns a number of vaccines for the batch.

**REQ 11:** *A citizen has a name, a unique NHS number and a list of certificates for received shots* - nhs_numbers is the set defining the user and nhs number relation. Citizen_names links each nhs number to a name. Later, using their NHS number, the certificates of a citizen can be retrieved..

**REQ 12:** *Each certificate includes the vaccine information and the date of receiving the shot* - The set certificate_dates stores the date of each certificate and certificate_vaccines stores the batch number of the vaccine administered. Using that information it is possible to find all the other information about the vaccine like the centre and brand.

**REQ 13:** *A citizen can retrieve their certificates from the system* - The event GetCitizenCertificateUser lets a user who is already logged in request their certificates. The result is stored in the citizen_certs set.

**REQ 14:** *A vaccination centre has a vaccine stock specifying the availability of the vaccines* - center_stock stores the batch number to centre relation. Using the batch number one can then retrieve the vaccine count in that batch from vaccines_count to check the vaccine availability of the center.

**REQ 15:** *An administrator can update the vaccine stock by adding more vaccine batches* - AssignBatchToCenter event enables the administrator to assign a batch to a centre by adding it to the center_stock. The event guards check that the batch is not assigned to another centre already and that the batch number is valid.

**REQ 16:** *A vaccination centre has a daily number of appointments* - The set appointments_nr stores a relation from centres to an integer number. Admins can change that number by updating that relation with a new number.

**REQ 17:** *A vaccination centre maintains a list of booked appointments* - appointment_center stores appointments of type APPOINTMENT linked to a centre. We can query the appointments for a centre by providing a centre as input.

**REQ 18:** *An administrator or a staff can view the citizen details using their NHS number* - The event GetCitizenCertificatesNHS enables an administrator who is already logged into the system to retrieve the user's certificates. Other information could be easily queried by inputting the NHS number into other relations of the system.

**REQ 30 & 31:** *A staff member can update the citizen's certificates after administering the vaccine (REQ 30) & the vaccine stock needs to be updated when the certificate is produced for a citizen (REQ 31)* - The IssueCertificate event is defined in machine 2 and extended with 2 more guards in machine 3. We make sure that a certificate can be issued only by a logged in staff member. The NHS number is associated with a certificate showing that a vaccine was administered and the vaccines_count set is updated by decrementing the vaccine count related to the batch number of the vaccine administered.

```
context VaccinationCentersContext extends UserManagementContext
sets
 BRAND
 BATCH_NR
 NAME
 NHS_NR
 CENTER
 CERTIFICATE
end
```

```
machine VaccinationCenters refines UserManagement
sees VaccinationCentersContext

/* Vaccination Centers
 * Requirements implemented:
 * REQ 10 A vaccine has a brand and a batch number
 * REQ 11 A citizen has a name, a unique NHS number and a list of certificates for received shots
 * REQ 12 Each certificate includes the vaccine information and the date of receiving the shot
 * REQ 13 A citizen can retrieve their certificates from the system
 * REQ 14 A vaccination centre has a vaccine stock specifying the availability of the vaccines
 * REQ 15 An adminstrator can update the vaccine stock by adding more vaccine batches
 * REQ 16 A vaccination centre has a daily number of appointments
 * REQ 17 An administrator or a staff can view the citizen details using their NHS number
 * REQ 30 A staff member can update the citizens certificates after administering the vaccine.
 */

variables
 users admins staff login register
```

```
vaccines
centers
center_stock
center_stock_amount
center_vaccines
day
citizen_names
nhs_numbers
certificates
appointments_nr
citizen_name
vaccines_count
citizen_certs
certificate_vaccines
certificate_dates

invariants
 @typeof−centers: centers ⊆ CENTER
 @typeof−vaccines: vaccines ∈ BATCH_NR ↔ BRAND
 @typeof−vaccines−count: vaccines_count ∈ BATCH_NR ↔ ℤ
 @center_stock−total_func: center_stock ∈ BATCH_NR ↔ CENTER
 @center_vaccines: center_vaccines ⊆ BATCH_NR
 @typeof−time: day ∈ ℤ
 @typeof−center_stock_amount: center_stock_amount ∈ ℤ

 @typeof−user−names: citizen_names ∈ NHS_NR ⇸ NAME
 @typeof−nhs−numbers: nhs_numbers ∈ USER ↔ NHS_NR
 @typeof−certificates: certificates ∈ NHS_NR ↔ CERTIFICATE
 @typeof−appointments−static: appointments_nr ∈ CENTER ↔ ℤ

 @typeof−citizen−name: citizen_name ⊆ NAME
 @typeof−citizen−certs: citizen_certs ⊆ CERTIFICATE
 @typeof−certificate−vaccines: certificate_vaccines ∈ CERTIFICATE ⇸ BATCH_NR
 @typeof−certificate−dates: certificate_dates ∈ CERTIFICATE ⇸ ℤ

events

 /∗ Initialize new variables ∗/
 event INITIALISATION extends INITIALISATION
 then
  @act−init−vaccines: vaccines := ∅
  @act−init−vaccines−count: vaccines_count := ∅
  @act−init−centers: centers := ∅
  @act−init−center_stock: center_stock := ∅
  @act−init−center_info: center_vaccines := ∅
  @act−init−day: day := 0
  @act−init−center_stock_amount: center_stock_amount := 0
  @act−init−names: citizen_names := ∅
  @act−init−nhs: nhs_numbers := ∅
  @act−init−certificates: certificates := ∅
  @act−init−appointments: appointments_nr := ∅
  @act−init−name: citizen_name := ∅
  @act−init−certs: citizen_certs := ∅
  @act−init−cert−vaccines: certificate_vaccines := ∅
  @act−init−cert−dates: certificate_dates := ∅
 end
```

```
/∗ Move time to the next day ∗/
event NextDay
then
 @inc−day: day := day + 1
end

/∗ Issue a vaccination certificate to the citizen. In the next machine it will be limited
 ∗ but now it is posibile to issue whenever. ∗/
event IssueCertificate
any
 user
 nhs
 cert
 batch
where
 @check−staff: user ∈ dom(staff)
 @check−loggedin: user ∈ dom(login)
 @check−nhs: nhs ∈ ran(nhs_numbers)
 @check−nhs−user: nhs_numbers∼[{nhs}] ⊆ dom(users)
 @check−cert: cert ∉ ran(certificates)
 @check−vaccine: batch ∈ dom(vaccines_count)
 @check−count: vaccines_count(batch) > 0
then
 @act1: certificates := certificates ∪ {nhs ↦ cert}
 @act2: certificate_vaccines := certificate_vaccines ∪ {cert ↦ batch}
 @act3: certificate_dates := certificate_dates ∪ {cert ↦ day}
 @act4: vaccines_count := vaccines_count ⩤ {batch ↦ (vaccines_count(batch) − 1)}
end

/∗ Let staff and admins get the name of a user using their NHS number ∗/
event GetCitizenName
any
 user
 nhs
where
 @check−admin−or−staff: user ∈ dom(admins) ∨ user ∈ dom(staff)
 @check−loggedin: user ∈ dom(login)
 @check−nhs: nhs ∈ dom(citizen_names)
then
 @act1: citizen_name := {citizen_names(nhs)}
end

/∗ Let the citizen get their certificates ∗/
event GetCitizenCertificatesUser
any
 user
where
 @check−user: user ∈ dom(users)
 @check−loggedin: user ∈ dom(login)
then
 @act1: citizen_certs := certificates[{nhs_numbers(user)}]
end

/∗ Get citizen certificate with their MHS number ∗/
event GetCitizenCertificatesNHS
```

```
any
 user
 nhs
where
 @check−admin−or−staff: user ∈ dom(admins) ∨ user ∈ dom(staff)
 @check−loggedin: user ∈ dom(login)
 @check−nhs: nhs ∈ ran(nhs_numbers)
then
 @act1: citizen_certs := certificates[{nhs}]
end

/∗ Add a vaccine to the system ∗/
event AddVaccine
any
 brand
 batch
 count
where
 @grd1: brand ∈ BRAND
 @grd2: batch ∉ dom(vaccines)
 @grd3: count ∈ ℤ ∧ count ≥ 1
then
 @act1: vaccines := vaccines ∪ {batch ↦ brand}
 @act2: vaccines_count := vaccines_count ∪ {batch ↦ count}
end

/∗ Add a center to the system ∗/
event AddCenter
any
 user
 center
 appoint_count
where
 @check−admin: user ∈ dom(admins)
 @check−loggedin: user ∈ dom(login)
 @check−center: center ∉ centers
 @check−appoint−nr: appoint_count ∈ ℤ ∧ appoint_count ≥ 0
then
 @add−center: centers := centers ∪ {center}
 @add−center−appoint−count: appointments_nr := appointments_nr ∪ {center ↦ appoint_count}
end

/∗ Change the number of daily appointments for a center ∗/
event ChangeCenterAppointCount
any
 user
 center
 count
where
 @check−admin: user ∈ dom(admins)
 @check−loggedin: user ∈ dom(login)
 @check−center: center ∈ centers
 @check−count: count ∈ ℤ ∧ count ≥ 0
then
 @act1: appointments_nr := appointments_nr ⩤ {center ↦ count}
end
```

```

/∗ Assign a batch to a vaccination center ∗/
event AssignBatchToCenter
any
 user
 batch
 center
where
 @check−admin: user ∈ dom(admins)
 @check−loggedin: user ∈ dom(login)
 @check−batch−vaccine: batch ∈ dom(vaccines)
 @check−batch−center: batch ∉ dom(center_stock)
 @check−center: center ∈ centers
then
 @act1: center_stock := center_stock ∪ {batch ↦ center}
end

/∗ Show vaccination center stock ∗/
event ShowCenterStock
any
 center
where
 @check−center: center ∈ centers
then
 @act1: center_vaccines := center_stock ∼ [{center}]
end

/∗ Initialize the root user as admin if there are no admins ∗/
event InitRootUser extends InitRootUser
any
 nhs
 name
where
 @check−nhs: nhs ∉ ran(nhs_numbers)
 @check−name: name ∈ NAME
then
 @add−citizen−name: citizen_names := citizen_names ∪ {nhs ↦ name}
end

/∗ Login a user ∗/
event LoginUser extends LoginUser
end

/∗ Logout a user ∗/
event LogoutUser extends LogoutUser
end

/∗ Register a new basic user ∗/
event RegisterUser extends RegisterUser
any
 nhs
 name
where
 @check−nhs: nhs ∉ dom(citizen_names)
 @check−name−unique: name ∉ ran(citizen_names)
then
```

```
   @add−nhs: nhs_numbers := nhs_numbers ∪ {user ↦ nhs}
   @add−name: citizen_names := citizen_names ∪ {nhs ↦ name}
  end

  /∗ Change a user’s password ∗/
  event ChangeUserPassword extends ChangeUserPassword
  end

  /∗ Change a staff’s password ∗/
  event ChangeStaffPassword extends ChangeStaffPassword
  end

  /∗ Change an admin’s password ∗/
  event ChangeAdminPassword extends ChangeAdminPassword
  end

  /∗ Elevate a user to staff role Their name and nhs number are removed.∗/
  event ElevateUserStaff extends ElevateUserStaff
  then
   @remove−name: citizen_names := nhs_numbers[{elevate_user}] ⩤ citizen_names
   @remove−nhs: nhs_numbers := nhs_numbers \ ({elevate_user} ◁ nhs_numbers)
  end

  /∗ Elevate a user to admin role. Their name and nhs number are removed. ∗/
  event ElevateUserAdmin extends ElevateUserAdmin
  then
   @remove−name: citizen_names := citizen_names \ (nhs_numbers[{user}] ◁ citizen_names)
   @remove−nhs: nhs_numbers := nhs_numbers \ ({elevate_user} ◁ nhs_numbers)
  end

  /∗ Elevate staff member to admin role ∗/
  event ElevateStaffAdmin extends ElevateStaffAdmin
  end

 end
```

## 5 Task 2. Event-B model 1.3 Appointment booking and rebooking

This section pertains to appointment bookings. Here a citizen can book an appointment under certain conditions (as detailed in provided specifications). The requested criteria for how and under what conditions the citizen can make an appointment were respected and can be found under each requirement below:

**REQ 19:** *A citizen can only book an appointment if they have received less than 3 shots* - In the event StartBooking the cardinality of the citizen's certificates is checked to ensure that it is less than 3.

**REQ 20:** *A citizen can only book an appointment if their last shot was more than 28 days (4 weeks) from the date of login* - Using list comprehensions we go through the citizen's certificates and build a set of all the certificates that are older than 28 days from the day of certificate issue. If this set is the same as the citizen certificates set this means that all the citizen's certificates are older than 28 days and we proceed.

**REQ 21:** *A citizen is offered the earliest available appointment at any centre* - The event GetNewBookingSuggestion will put all the centres available for a day in the appointments_suggested set offers the earliest available appointment.

**REQ 22:** *If the citizen rejects the current offer, the system offers the following earliest appointment, that might be at a different centre* - The event RejectBookingSuggestion will increment the tmp_day that will move us to the next day and will make GetNewBookingSuggestion available again to query the centres available that day.

**REQ 23:** *If the citizen accepts the current offer, the booking is confirmed* - The event AcceptBookingSuggestion will assign a new unique appointment to the sets storing the association to a NHS number, center and the day of the appointment. This whole flow described above is controlled by a helper variable state that enables and disables the events when appropriate.

**REQ 24:** *At any point during the booking process, the use can abandon the system without completing the booking* - The event AbandonAppointmentBooking will set the day we want to book to 15. This will trigger the events described in REQ 25 and we will interrupt the appointment booking flow.

**REQ 25:** *The booking is stopped if there are no more available appointments within the 14 days from the date of login to offer* - All the events have guards that will stop the flow when the day we are checking for reaches 15 and the StartAppointmentRegistration event will become again available to start a new booking flow.

**REQ 26:** *A citizen can view their current booking (if any).* - GetCitizenBookings enables users to view their bookings that are stored in the citizen_books set which allows for them to view their current booking.

**REQ 27:** *Rebooking an appointment can be done up to 1 day before the date of the original* *one* - The guard in StartRebooking makes sure of this by checking that there are appointments that have their day older than the current day using a list comprehension.

**REQ 28** *Rebooking follows the same rules as booking* - The flow is practically the same for the StartRebooking, GetNewRebookingSuggestion, AceeptRebookingSuggestion and RejectRebookingSuggestion with small changes to update the sets related to appointments instead of adding to them and some additional guards.

**REQ 29:** *After re-booking is confirmed, the original appointment becomes available* - This occurs as there are separate sets storing the information of an appointment, updating it is enough to free the old appointment slot.

```
context AppointmentManagementContext extends VaccinationCentersContext
sets
 APPOINTMENT
end
```

```
machine AppointmentManagement
refines VaccinationCenters
sees AppointmentManagementContext

/*AppointmentManagementContext
* Requirements implemented:
* REQ 18 A vaccination centre maintains a list of booked appointments
* REQ 19 A citizen can only book an appointment if they have received less than 3 shots
* REQ 20 A citizen can only book an appointment if their last shot was more than 28 days
* (4 weeks) from the date of the last shot
* REQ 21 A citizen is offered the earliest available appointment at any centre
* REQ 22 If the citizen rejects the current offer, the system offers the following earliest
* appointment, that might be at a different centre
* REQ 23 If the citizen accepts the current offer, the booking is confirmed
* REQ 24 At any point during the booking process, the use can abandon the system
* without completing the booking
* REQ 25 The booking is stopped if there are no more available appointments within the
* 14 days from the date of login to offer
* REQ 26 A citizen can view their current booking (if any).
* REQ 27 Rebooking an appointment can be done up to 1 day before the date of the original one
* REQ 28 Rebooking follows the same rules as booking
* REQ 29 After re−booking is confirmed, the original appointmnet becomes available
*/

variables
 users
 admins
 staff
 login
 register
 vaccines
 centers
 center_stock
 center_vaccines
 day
 day_tmp
 citizen_names
 nhs_numbers
 certificates
 appointments_nr
 citizen_name
 vaccines_count
 citizen_certs
 certificate_vaccines
 certificate_dates
 center_stock_amount

 appointments_potential
 appointments_suggested
 calendar
 state
 user_session
 appointment_nhs
 appointment_center
 appointment_day
 citizen_books

invariants
 @typeof−appointments−potential: appointments_potential ⊆ centers
 @typeof−appointments−suggested: appointments_suggested ⊆ centers
 @typeof−calendar: calendar ∈ centers ↔ ℤ
 @typeof−day−tmp: day_tmp ∈ ℤ
 @typeof−state: state ∈ ℤ

 @typeof−user−session: user_session ⊆ USER
 @typeof−appoint−nhs: appointment_nhs ∈ APPOINTMENT ↔ NHS_NR
 @typeof−appoint−batch: appointment_center ∈ APPOINTMENT ↔ CENTER
 @typeof−appoint−day: appointment_day ∈ APPOINTMENT ↔ ℤ
 @typeof−citizen−books: citizen_books ⊆ APPOINTMENT

events

 /∗ Initialize new variables ∗/
 event INITIALISATION extends INITIALISATION
 then
  @act−init−appointments−offered: appointments_potential := ∅
  @act−init−appointments−suggested: appointments_suggested := ∅
  @act−init−calendar: calendar := ∅
  @act−init−day−tmp: day_tmp := 0
  @act−init−state: state := 0
  @act−init−user−session: user_session := ∅
  @act−init−appoint−nhs: appointment_nhs := ∅
  @act−init−appoint−center: appointment_center := ∅
  @act−init−appoint−day: appointment_day := ∅
  @act−init−citizen−books: citizen_books := ∅
 end

 /∗ Issue certificate is now restricted to make certificates on the day of the appointment ∗/
 event IssueCertificate extends IssueCertificate
 any
 appointment
where
 @check−appointment: appointment ∈ appointment_nhs ∼ [{nhs}]
 @check−day: appointment_day(appointment) = day
end

/∗ Get citizen’s appointments ∗/
event GetCitizenBookings
any
 user
where
 @check−citizen: user ∈ dom(users)
 @check−loggedin: user ∈ dom(login)
then
 @act1: citizen_books := appointment_nhs ∼ [{nhs_numbers(user)}]
end

/∗ Starts the rebooking sequence ∗/
event StartRebooking
any
 user
where
 /∗ Check the citizen is logged in ∗/
 @check−citizen: user ∈ dom(users)
 @check−loggedin: user ∈ dom(login)

 /∗ Check that the citizen has less than 3 vaccines ∗/
 @check−vaccine−count: card(certificates[{nhs_numbers(user)}]) < 3

 /∗ Check the user had their last vaccine more than 28 days ago. The list comprehension returns
  ∗ all the certificates that are 28 days old then it compares it to the set of citizen certificates.
  ∗ If they are the same then all the vaccines are 28+ days old.
  ∗/
 @check−vaccine−last: {cert | cert ∈ certificates[{nhs_numbers(user)}] ∧ (day − certificate_dates(cert) >
    28)}
 = certificates[{nhs_numbers(user)}]

 /∗ Check that there are centers with non zero stock and have available appointments ∗/
 @check−non−zero: {ctr | ctr ∈ {cr | cr ∈ centers ∧ {btch | btch ∈ center_stock ∼ [{cr}] ∧ vaccines_count(
    btch) > 0} ≠ ∅} ∧
  ((card({ctr} ◁ calendar) < appointments_nr(ctr) ∨ ctr ∉ dom(calendar)) ∧ appointments_nr(ctr) ≠ 0)
 } ≠ ∅

 @check−appointment: nhs_numbers(user) ∈ ran(appointment_nhs)
 @check−day: {appoint | appoint ∈ appointment_nhs ∼ [{nhs_numbers(user)}] ∧ appointment_day(
    appoint) > day} ≠ ∅
then
 @get−valid−centers: appointments_potential := {ctr | ctr ∈ centers ∧
  {btch | btch ∈ center_stock ∼ [{ctr}] ∧ vaccines_count(btch) > 0} ≠ ∅
 }
 @init−first−day: day_tmp := day + 1
 @change−state: state := 3
 @save−user: user_session := {user}
end

/∗ Generate rebooking options for a day in the calendar. Make unavailable after 14 days from today. ∗/
```

```
event GetNewRebookingSuggestion
where
 @appointment−started: appointments_potential ≠ ∅
 @check−day−tmp: day_tmp < day + 15
 @check−state: state = 3
then
 @get−valid−centers: appointments_suggested := {ctr | ctr ∈ appointments_potential ∧
  ((card({ctr} ◁ calendar) < appointments_nr(ctr) ∨ ctr ∉ dom(calendar)) ∧ appointments_nr(ctr) ≠ 0)
 }
 @change−state: state := 4
end

/∗ Reject the rebooking options for that day. Move to the next day. Make unavailable after 14 days from
    today.∗/
event RejectRebookingSuggestion
where
 @check−day−tmp: day_tmp < day + 15
 @check−state: state = 4
then
 @chande−day: day_tmp := day_tmp + 1
 @change−state: state := 3
end

/∗ Accept a rebooking option and change the booking information. Make unavailable after 14 days from
    today. ∗/
event AcceptRebookingSuggestion
any
 center
 appointment
 user
where
 @check−center: center ∈ appointments_suggested
 @check−appointment: appointment ∉ dom(appointment_nhs)
 @check−day−tmp: day_tmp < day + 15
 @check−state: state = 4
 @check−user−session: user ∈ user_session
then
 @calendar−add: calendar := calendar ∪ {center ↦ day_tmp}
 @link−center−change: appointment_center := appointment_center  {appointment ↦ center}
 @link−day−change: appointment_day := appointment_day  {appointment ↦ day_tmp}
 @interupt−session: day_tmp := day + 15
 @change−state: state := 0
end

/∗ Stop the rebooking sequence.∗/
event AbandonAppointmentRebooking
where
 @check−state: state = 3 ∨ state = 4
then
 @interupt−session: day_tmp := day + 15
 @change−state: state := 0
end

/∗ Starts the booking sequence ∗/
event StartAppointmentRegistration
any
```

```
 user
where
 /∗ Same as rebooking ∗/
 @check−citizen: user ∈ dom(users)
 @check−loggedin: user ∈ dom(login)
 @check−vaccine−count: card(certificates[{nhs_numbers(user)}]) < 3
 @check−vaccine−last: {cert | cert ∈ certificates[{nhs_numbers(user)}] ∧ (day − certificate_dates(cert) > 27)} = certificates[{nhs_numbers(user)}]
 @check−non−zero: {ctr | ctr ∈ {cr | cr ∈ centers ∧ {btch | btch ∈ center_stock ∼ [{cr}] ∧ vaccines_count(btch) > 0} ≠ ∅} ∧
  ((card({ctr} ◁ calendar) < appointments_nr(ctr) ∨ ctr ∉ dom(calendar)) ∧ appointments_nr(ctr) ≠ 0)
 } ≠ ∅
then
 @get−valid−centers: appointments_potential := {ctr | ctr ∈ centers ∧ {btch | btch ∈ center_stock ∼ [{ctr}] ∧ vaccines_count(btch) > 0} ≠ ∅}
 @init−first−day: day_tmp := day + 1
 @change−state: state := 1
 @save−user: user_session := {user}
end

/∗ Generate booking options for a day in the calendar. Make unavailable after 14 days from today. ∗/
event GetNewAppointmentSuggestion
where
 @appointment−started: appointments_potential ≠ ∅
 @check−day−tmp: day_tmp < day + 15
 @check−state: state = 1
then
 @get−valid−centers: appointments_suggested := {ctr | ctr ∈ appointments_potential ∧
  ((card({ctr} ◁ calendar) < appointments_nr(ctr) ∨ ctr ∉ dom(calendar)) ∧ appointments_nr(ctr) ≠ 0)
 }
 @change−state: state := 2
end

/∗ Reject the booking options for that day. Move to the next day. Make unavailable after 14 days from today.∗/
event RejectAppointmentSuggestion
where
 @check−day−tmp: day_tmp < day + 15
 @check−state: state = 2
then
 @init−first−day: day_tmp := day_tmp + 1
 @change−state: state := 1
end

/∗ Accept a booking option and change the booking information. Make unavailable after 14 days from today. ∗/
event AcceptAppointmentSuggestion
any
 center
 appointment
 user
where
 @check−center: center ∈ appointments_suggested
 @check−appointment: appointment ∉ dom(appointment_nhs)
 @check−day−tmp: day_tmp < day + 15
 @check−state: state = 2
```

```
  @check−user−session: user ∈ user_session
then
  @calendar−add: calendar := calendar ∪ {center ↦ day_tmp}
  @link−citizen: appointment_nhs := appointment_nhs ∪ {appointment ↦ nhs_numbers(user)}
  @link−center: appointment_center := appointment_center ∪ {appointment ↦ center}
  @link−day: appointment_day := appointment_day ∪ {appointment ↦ day_tmp}
  @interupt−session: day_tmp := day + 15
  @change−state: state := 0
end

/∗ Stop the booking sequence.∗/
event AbandonAppointmentRegistration
where
  @check−state: state = 1 ∨ state = 2
then
  @interupt−session: day_tmp := day + 15
  @change−state: state := 0
end

event NextDay extends NextDay
end

event GetCitizenName extends GetCitizenName
end

event GetCitizenCertificatesUser extends GetCitizenCertificatesUser
end

event GetCitizenCertificatesNHS extends GetCitizenCertificatesNHS
end

event AddVaccine extends AddVaccine
end

event AddCenter extends AddCenter
end

event ChangeCenterAppointCount extends ChangeCenterAppointCount
end

event AssignBatchToCenter extends AssignBatchToCenter
end

event ShowCenterStock extends ShowCenterStock
end

event InitRootUser extends InitRootUser
end

event LoginUser extends LoginUser
end

event LogoutUser extends LogoutUser
end

event RegisterUser extends RegisterUser
```

```
  end

  event ChangeUserPassword extends ChangeUserPassword
  end

  event ChangeStaffPassword extends ChangeStaffPassword
  end

  event ChangeAdminPassword extends ChangeAdminPassword
  end

  event ElevateUserStaff extends ElevateUserStaff
  end

  event ElevateUserAdmin extends ElevateUserAdmin
  end

  event ElevateStaffAdmin extends ElevateStaffAdmin
  end

end
```