University of Southampton

Samuel John Nugent (sjn1g23)
James Caldow (jc12g23)
Sam Hunt (sh6g23)
Joseph Jewell (jj12n23)
Chak Tim Lam (ctl1e22)

COMP1216. Software Modelling and Design (2023-24)

Group 13: Dentist System

Submission date: 10 May 2024

1 Introduction

Brief description of each member's contribution:
Samuel Nugent - Made and wrote: DentistContext01.bucx, DentistSystem01.bumx (apart from the logIn and logOut events), DentistContext02.bucx, DentistSystem02.bumx.

Sam Hunt - Made and wrote: DentistContext03.bucx, DentistSystem03.bumx (apart from the CheckInPatient, CompletedAppointment and MissedAppointment events and inv18).

Joseph Jewell - Made the CheckInPatient, CompletedAppointmentand MissedAppointment events + inv18

James Caldow - Wrote DentistSystem01.bumx: logIn and logOut events and did the Class Diagram

Chak Tim Lam - Wrote DentistSystem.04.bumx

1.1 Assumptions

The following assumptions about the system have been made:

1. A registered user cannot become unregistered (they cannot be unregistered from the system).

2. When a dentist is registered to the system, they are registered with all the treatments they are qualified to perform. They cannot become qualified to perform more treatments at a later time.

3. When a user is registered they start as being logged out.

4. A patient does not need to be logged-in to check in (as they check in in person).

5. For an appointment to be moved from the checked in state to the completed state it does not need to be checked that it is still the same day that the appointment was booked for as an appointment could take multiple days to complete (if held overnight for example) after checking in on the day it was booked for.

2 Task 1. Class diagram

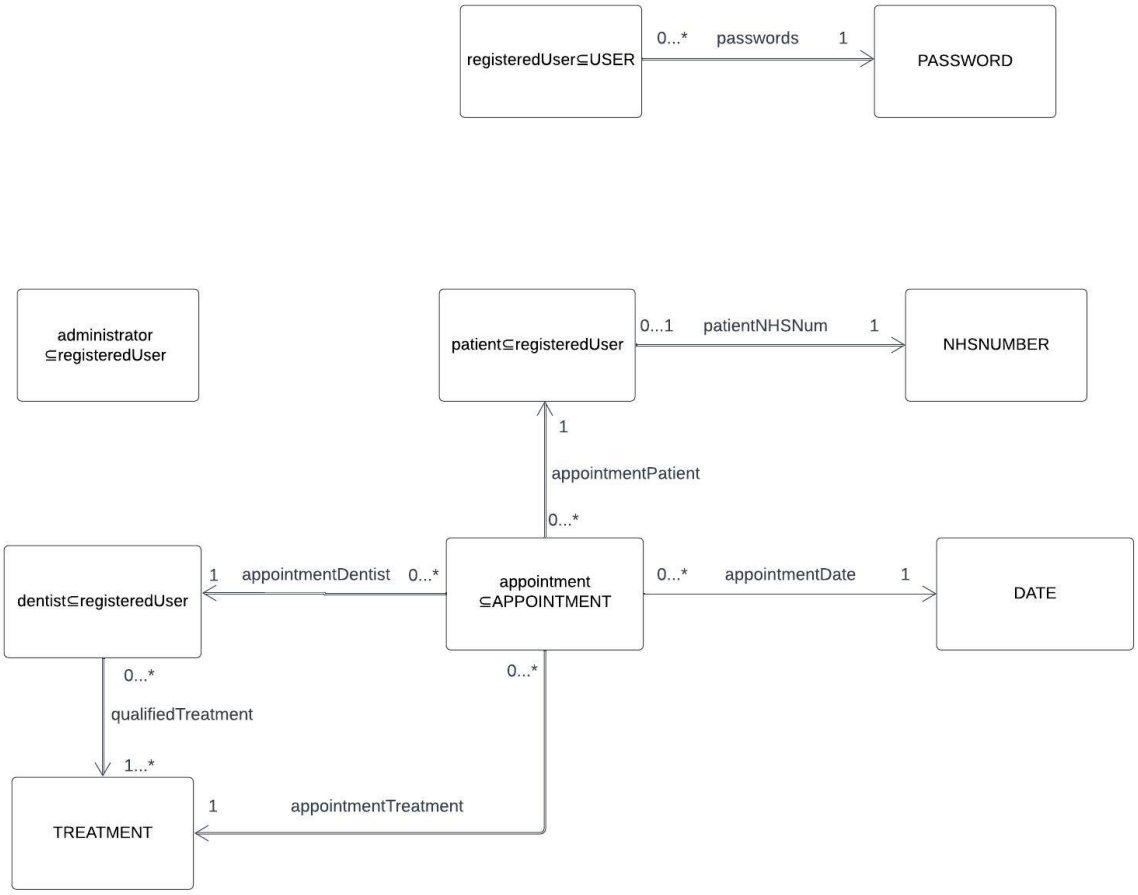

Figure 1: Class Diagram

3 Task 2. Event-B model

```
/*
 * This context models the static part of the model for users
 * by declaring the set of users and the set of passwords
 * and creating a constant for the root user and their password
 * so they can be added to the machine as it initialises
 */
context DentistContext01
sets
 USER // The set of all users (including those not registered on the system)
 PASSWORD // The set of passwords (including those not in use)
constants
ROOTUSER // create a rootUser as a constant
ROOTUSERPASSWORD // create the root user's password as a constant
// these are made as constants so they can be added in when the system starts (initialises)
axioms
@def−rootUser:ROOTUSER ∈ USER // the rootUser is a user
@def−rootUserPassword:ROOTUSERPASSWORD ∈ PASSWORD // the root user's password is a
    password
end
```

```
/*
 * This machine defines the dynamic part of the dentist system for a user.
 * It models the set of registered users, with their passwords and whether
 * the registered user is logged in or out.
 */

machine DentistSystem01
sees DentistContext01
variables
 registeredUser // The users who are registered on the system
 passwords // The password information for the registered users
 loggedIn  // The set of users who are logged in
 loggedOut // The set of users who are logged out
invariants
 @inv1:registeredUser ⊆ USER // registered users are a subset of users, only those that are registered on the
     system
 /*
  * Each registeredUser has exactly 1 password. Each password could be used by multiple users.
  * However, not every possible password is mapped to (only those that are in use).
  */
 @inv2:passwords ∈ registeredUser → PASSWORD
 @inv3:partition(registeredUser, loggedIn, loggedOut) // a user can either be logged in, or logged out, but
     not both at
 // the same time.

events

 /*
  * When the system starts, it starts with a root user as a registeredUser, and starts with passwords as only
     containing the
  * mapping from the root user to their password, and loggedIn starts as the empty set so no users are
```

```
    logged in and
29  * loggedOut starts containing only the root user, so the root user is the only registered user who is logged
    out.
30  */
31  event INITIALISATION
32  begin
33   @act1:registeredUser, passwords, loggedIn, loggedOut := {ROOTUSER}, {ROOTUSER ↦
    ROOTUSERPASSWORD}, ∅, {ROOTUSER}
34  end
35
36  /*
37   * A new (not yet registered) user @user registers with the system with a password @password.
38   * When the user is registered they will then have the ability to log in (i.e. when they register
39   * they are first added to loggedOut)
40   */
41  event registerUser
42  any
43   user // The user to register
44   password // The password they are using to register with
45  where
46   @grd1:user ∈ USER // @user is is a user
47   @grd2:user ∉ registeredUser // The @user is not already registered
48   @grd3:password ∈ PASSWORD // The @password is a password
49  then
50   @act1:registeredUser := registeredUser ∪ {user} // The user @user is now a registered user
51   @act2:passwords(user) := password // The user @user is added with their password @password to the
    password information
52   @act3:loggedOut := loggedOut ∪ {user} // The user @user is set as being logged out
53  end
54
55
56  /*
57   * User @user changes their password from their current password @currentPass to a new password
    @newPassword
58   */
59  event changePassword
60  any
61   user // The user to change their password
62   currentPass // the current password they are using
63   newPassword // the new password they are changing their password to
64  where
65   @grd1:user ∈ loggedIn // User @user is logged in, only need to check this as only registered users can be
    logged in
66   @grd2:passwords(user) = currentPass // The password belonging to user @user matches the given current
    password @currentPass
67   @grd3:passwords(user) ≠ newPassword // The password belonging to user @user does not match the new
    password @newPassword
68   // this is so they can only change their password to a new password
69  then
70   @act1:passwords(user) := newPassword // set the password for the user @user to now be the new
    password @newPassword
71  end
72
73  /*
74   * User @user logs in using the password @password
75   */
```

```
76  event logIn
77  any
78   user // The user to log in
79   password // The password used by the user to log in with
80  where
81   @grd1:user ∈ loggedOut // User @user is currently logged out, only need to check this as only registered
         users can be
82   // logged out
83   @grd2:passwords(user) = password // The password @password matches the user's, @user, password
84  then
85   @act1:loggedOut := loggedOut \ {user} // Remove @user from logged Out
86   @act2:loggedIn := loggedIn ∪ {user} // Add @user to logged in
87  end
88
89  /*
90   * User @user logs out
91   */
92  event logOut
93  any user // The user to log out
94  where
95   @grd1:user ∈ loggedIn // User @user is currently logged in, only need to check this as only registered users
         can be
96   // logged in
97  then
98   @act1:loggedIn := loggedIn \ {user} // Remove @user from logged In
99   @act2:loggedOut := loggedOut ∪ {user} // Add @user to logged Out
100 end
101 end
```

```
1  /*
2   * This context models the static set of treatments and NHS numbers
3   */
4  context DentistContext02
5  extends DentistContext01
6  sets
7   TREATMENT // The set of all possible treatments
8   NHSNUMBER // The set of NHS numbers, both those that are in use and not in use.
9  end
```

```
1  /*
2   * This machine models the roles of users, i.e., when a user is registered
3   * they are registered as a specific role (administrator, dentist, or patient).
4   * Patients can register themselves, whereas only a logged−in administrator
5   * can register a new user as an administrator or dentist
6   */
7  machine DentistSystem02
8  refines DentistSystem01
9  sees DentistContext02
10 variables
11  registeredUser // The set of users which are registered on the system
12  passwords // The password information for the registered users
13  loggedIn // The set of registered users that are logged in
```

```
14  loggedOut // The set of registered users that are logged out
15  administrator // The set of registered administrators
16  dentist // The set of registered dentists
17  patient // The set of registered patients
18  qualifiedTreatment // The information for dentists and the treatments they are qualified to give
19  patientNHSNum // The NHS number information for patients
20
21 invariants
22  /*
23   * A user can either be an administrator, a dentist, or a patient, but they cannot be more than one of those
        things.
24   */
25  @inv4:partition(registeredUser, administrator, dentist, patient)
26
27  /*
28   * Every dentist is qualified to perform at least one treatment (can perform one or more treatment)
29   * Many dentists are qualified to perform the same treatment (treatments can be performed by 0 or more
        dentists)
30   *
31   * Event-B is not able to prove using a total relation, so instead define inv5 with a relation then with
32   * inv6 define that the domain qualifiedTreatment is equal to dentist.
33   */
34  @inv5:qualifiedTreatment ∈ dentist ↔ TREATMENT
35
36  @inv6:dom(qualifiedTreatment) = dentist // each dentist is qualified to perform at least one treatment
37
38  /*
39   * Every patient has only 1 NHS number which must be unique.
40   * Therefore, each NHS number can only belong to 1 patient (or 0 if not in use)
41   */
42  @inv7:patientNHSNum ∈ patient ⤖ NHSNUMBER
43
44 events
45
46  /*
47   * The initialisation of the system
48   */
49  event INITIALISATION
50  extends INITIALISATION
51  begin
52   @init-admin:administrator := {ROOTUSER} // The initial root user is an administrator, so they start as
        the only administrator
53   @init-dentist:dentist := ∅ // Initially, there are no dentists
54   @init-patient:patient := ∅  // Initially, there are no patients
55   @init-qualifiedTreatment:qualifiedTreatment := ∅ // Initially, there are no treatments' dentists can
        perform
56   @init-NHSNum:patientNHSNum := ∅ // Initially, there is no information for patients' NHS numbers
57  end
58
59  /*
60   * The new user @user is registered to the system as an administrator with the password @password
61   * and they are registered by a logged-in administrator @registeringAdmin
62   */
63  event registerAdmin
64  extends registerUser
65  any
```

```
 registeringAdmin // The administrator which is registering the new user for the role of administrator
where
 @regAdmin−exists:registeringAdmin ∈ administrator // The administrator @registeringAdmin is an
     administrator
 @regAdmin−loggedIn:registeringAdmin ∈ loggedIn // The administrator @registeringAdmin is logged−
     in
then
 @addNewAdmin:administrator := administrator ∪ {user} // Add the new user @user to the set of
     administrators
end

/*
 * The new user @user is registered to the system as a dentist with the password @password
 * The new user @user is registered by a logged−in administrator @registeringAdmin.
 */
event registerDentist
extends registerUser
any
 registeringAdmin // The administrator which is registering the new user for the role of dentist
 treatments // The treatments which the new user (who is a dentist) is qualified to perform
where
 @regAdmin−exists:registeringAdmin ∈ administrator // The administrator @registeringAdmin is an
     administrator
 @regAdmin−loggedIn:registeringAdmin ∈ loggedIn // The administrator @registeringAdmin is logged−
     in
 @treatments−subset:treatments ⊆ TREATMENT // @treatments is a subset of all possible treatments
 @treatments−notEmpty:treatments ≠ ∅ // @treatments consists of one or more treatments
then
 @addNewDentist:dentist := dentist ∪ {user} // The user @user is now a registered dentist
 @addQualifiedTreatments:qualifiedTreatment := qualifiedTreatment ∪ ({user} × treatments) // Add the
     information for the treatments @treatments
 // which the user @user is qualified to perform
end

/*
 * Register a new user @user to the system with the password @password as a patient with
 * the NHS number @NHSNum. A patient can register themselves to the system.
 */
event registerPatient
extends registerUser
any
 NHSNum // The NHS number for the new user (which is a patient)
where
 @NHSNum−exist:NHSNum ∈ NHSNUMBER // @NHSNum is an NHS number
 @NHSNum−unique:NHSNum ∉ ran(patientNHSNum) // The NHS number @NHSNum does not already
     belong to another patient
 // therefore, the NHS number is unique.
then
 @addNewPatient:patient := patient ∪ {user} // Add the new user @user to the role of patient
 @addPatientsNHSNum:patientNHSNum(user) := NHSNum // Allocate @NHSNum to the patient @user
end

/*
 * User @user changes their password from their current password @currentPass to a new password
     @newPassword
 */
```

```
114  event changePassword
115  extends changePassword
116  end
117
118  /*
119   * User @user logs in using the password @password
120   */
121  event logIn
122  extends logIn
123  end
124
125  /*
126   * User @user logs out
127   */
128  event logOut
129  extends logOut
130  end
131
132 end
```

```
1  /*
2   * This context models the static set of appointments
3   */
4  context DentistContext03
5  extends DentistContext02
6  sets
7   APPOINTMENT // The set of all possible appointments
8  end
```

```
1  /*
2   * This machine models the appointments stored in the system, as
3   * well as the five states that an appointment can be in. A logged-in
4   * patient can book an appointment for a treatment, cancel an already-booked
5   * appointment, and re-book an appointment for a different time/dentist.
6   *
7   * A patient can also check in for their appointment on the date of the appointment.
8   * A logged-in dentist can then change the status of the appointment from "checked-in"
9   * to "completed"
10  * An administrator can set an appointment to "missed" if a patient has not turned up
11  * for their appointment.
12  *
13  * As DentistSystem03 is a large Event-B model, it has to be included in "bunches", otherwise a "TeX
        capacity exceeded" error will occur.
14  */
15 machine DentistSystem03
16 refines DentistSystem02
17 sees DentistContext03
18 variables
19  registeredUser // The set of users registered to the system
20  passwords // The set of passwords stored in the system
21  loggedIn // The set of registered users that are logged in
22  loggedOut // The set of registered users that are logged out
```

```
23 administrator // The set of registered administrators
24 dentist // The set of registered dentists
25 patient // The set of registered patients
26 qualifiedTreatment // The relationship which maps each dentist to the treatments that they are qualified
       to perform
27 patientNHSNum // The relationship which maps each patient to their unique NHS number
28 day // The current day
29 appointment // The set of all existing appointments
30 appointmentPatient // The relationship which maps each appointment to the patient whose appointment
       it is
31 appointmentTreatment // The relationship which maps each appointment to the treatment to be
       performed
32 appointmentDate // The relationship which maps each appointment to the day it will take place
33 appointmentDentist // The relationship which maps each appointment to the dentist which will perform
       the specified treatment
34 booked // The set of booked appointments
35 checkedIn // The set of checked−in appointments
36 cancelled // The set of cancelled appointments
37 missed // The set of missed appointments
38 completed // The set of completed appointments
39
40 invariants
41 @inv8: day ∈ ℕ // The current day must be a natural number
42 @inv9: appointment ⊆ APPOINTMENT // The set of all appointments is a subset of the carrier set
       APPOINTMENT
43 /*
44  * Every appointment in "appointment" is linked with one single patient
45  * Each patient can have any number of appointments
46  */
47 @inv10: appointmentPatient ∈ appointment → patient
48 /*
49  * Every appointment in "appointment" is mapped to one single treatment
50  * Each treatment can be mapped to any number of appointments
51  */
52 @inv11: appointmentTreatment ∈ appointment → TREATMENT
53 /*
54  * Every appointment in "appointment" is mapped to a single natural number (the date of the
       appointment)
55  * Every natural number can be mapped to multiple appointments, as multiple appointments can happen
       on the same day
56  */
57 @inv12: appointmentDate ∈ appointment → ℕ
58 /*
59  * Every appointment in "appointment" is mapped to a single dentist
60  * Every dentist can be mapped to multiple appointments, as many appointments can be booked for the
       same dentist
61  */
62 @inv13: appointmentDentist ∈ appointment → dentist
63 @inv14: partition(appointment, booked, checkedIn, cancelled, missed, completed) // An appointment can
       only exist in one of its five possible states at a time
64 @inv15: ∀ a · a ∈ booked ⇒ appointmentDate(a) ≥ day // All booked appointments must either be in the
       present or the future, you cannot have an appointment booked for a past date
65 @inv16: ∀ a1, a2 · a1 ∈ booked ∧ a2 ∈ booked ∧ a1 ≠ a2 ∧ appointmentPatient(a1) = appointmentPatient(
       a2) ⇒ appointmentTreatment(a1) ≠ appointmentTreatment(a2) // A patient cannot have 2 booked
       appointments for the same treatment
66 @inv17: ∀ a1, a2 · a1 ∈ booked ∧ a2 ∈ completed ∧ a1 ≠ a2 ∧ appointmentPatient(a1) = appointmentPatient
```

```
(a2) ∧ (appointmentDate(a1) − appointmentDate(a2)) ≤ 14 ⇒ appointmentTreatment(a1) ≠
    appointmentTreatment(a2) // A patient cannot have 2 appointments for the same treatment within 2
    weeks of each other
@inv18: ∀ a · a ∈ missed ⇒ appointmentDate(a) ≤ day //ensures that for every appointment a in the set of
    missed appointments, the appointment date must be less than or equal to the current day.

events
 event INITIALISATION
 extends INITIALISATION
 begin
  @init−day: day := 0 // The first day of the system is day 0
  @init−appointment: appointment := ∅ // Initially, there are no appointments
  /*
   * As there are no appointments initially, there are no mappings between any appointments and any
     patients,
   * treatments, dates or dentists
   */
  @init−appointmentPatient: appointmentPatient := ∅
  @init−appointmentTreatment: appointmentTreatment := ∅
  @init−appointmentDate: appointmentDate := ∅
  @init−appointmentDentist: appointmentDentist := ∅
  /*
   * As there are no initial appointments, the sets representing the different appointment states are also
   * initially empty
   */
  @init−bookedState: booked := ∅
  @init−checkedInState: checkedIn := ∅
  @init−cancelledState: cancelled := ∅
  @init−missedState: missed := ∅
  @init−completedState: completed := ∅
 end

 /*
  * The new user @user is registered to the system as an administrator with the password @password
  * and they are registered by a logged−in administrator @registeringAdmin
  */
 event registerAdmin extends registerAdmin
 end
```

```
 /*
  * The new user @user is registered to the system as a dentist with the password @password
  * The new user @user is registered by a logged−in administrator @registeringAdmin.
  */
 event registerDentist extends registerDentist
 end

 /*
  * Register a new user @user to the system with the password @password as a patient with
  * the NHS number @NHSNum. A patient can register themselves to the system.
  */
 event registerPatient extends registerPatient
 end
```

```
/*
 * User @user changes their password from their current password @currentPass to a new password @newPassword
 */
event changePassword extends changePassword
end

/*
 * User @user logs in using the password @password
 */
event logIn extends logIn
end

/*
 * User @user logs out
 */
event logOut extends logOut
end

/*
 * The current day is increased by one
 */
event nextDay
where
 @grd1: ∀ a · a ∈ booked ⇒ appointmentDate(a) > day // All booked appointments must be for a future date
then
 @act1: day := day + 1 // Increase the day by one
end

/*
 * A registered, logged−in patient @newPatient can book an appointment @newAppointment for the treatment @newTreatment
 * with the dentist @newDentist on a specified date @newDay
 */
event bookAppointment
any
 newAppointment // The new appointment to be booked
 newPatient // The patient who is booking the appointment
 newTreatment // The treatment which is going to be carried out in the appointment
 newDay // The day the appointment is scheduled for
 newDentist // The dentist who will carry out the appointment
where
 @grd1: newAppointment ∈ APPOINTMENT \ appointment // The appointment must be of type APPOINTMENT but cannot already exist on the system
 @grd2: newPatient ∈ patient // The user booking the appointment must be a registered patient
 @grd3: newTreatment ∈ TREATMENT // The treatment scheduled must be of type TREATMENT
 @grd4: newDay ∈ ℕ // The day specified must be a natural number
 @grd5: newDentist ∈ dentist // The dentist who is carrying out the treatment must be a registered dentist
 @grd6: newDay > day // The appointment must be booked for a future date
 @grd7: (newDentist ↦ newTreatment) ∈ qualifiedTreatment // The dentist must be qualified to carry out the treatment
 @grd8: newPatient ∈ loggedIn // The patient booking the appointment must be logged in
 @grd9: ∀ a · a ∈ booked ∧ appointmentPatient(a) = newPatient ⇒ appointmentTreatment(a) ≠ newTreatment // A patient cannot have 2 appointments with the same treatment booked
 @grd10: ∀ a · a ∈ completed ∧ appointmentPatient(a) = newPatient ∧ newDay ≤ (appointmentDate(a) +
```

```
        14) ⇒ appointmentTreatment(a) ≠ newTreatment // A patient cannot have a treatment booked if it is
            within 2 weeks of another completed appointment with the same treatment
165  then
166   @act1: appointment := appointment ∪ {newAppointment} // The new appointment is added to the set of
            appointments
167   @act2: appointmentPatient(newAppointment) := newPatient // The patient is assigned to the
            appointment
168   @act3: appointmentTreatment(newAppointment) := newTreatment // The treatment is assigned to the
            appointment
169   @act4: appointmentDate(newAppointment) := newDay // The date is assigned to the appointment
170   @act5: appointmentDentist(newAppointment) := newDentist // The dentist is assigned to the
            appointment
171   @act6: booked := booked ∪ {newAppointment} // The appointment's state is set to "booked"
172  end
173
174  /*
175   * A registered, logged−in patient @cancellingPatient can cancel an appointment
            @cancelledAppointment
176   */
177  event cancelAppointment
178  any
179   cancelledAppointment // The appointment to be cancelled
180   cancellingPatient // The patient who is cancelling the appointment
181  where
182   @grd1: cancelledAppointment ∈ booked // The appointment to be cancelled must have previously been
            booked
183   @grd2: appointmentDate(cancelledAppointment) − day ≥ 2 // The appointment to be cancelled must be 2
            days or more in the future from the current date
184   @grd3: cancellingPatient ∈ patient // The cancelling patient must be a registered patient
185   @grd4: cancellingPatient ∈ loggedIn // The cancelling patient must be logged in
186  then
187   @act1: booked := booked \ {cancelledAppointment} // Change the appointment's status from "booked"
188   @act2: cancelled := cancelled ∪ {cancelledAppointment} // to "cancelled"
189  end
190
191  /*
192   * A registered, logged−in patient @rebookingPatient can re−book their appointment
            @rebookedAppointment
193   * and change the dentist @newDentist and/or date @newDate
194   */
195  event rebookAppointment
196  any
197   rebookedAppointment // The appointment to be re−booked
198   rebookingPatient // The patient who is re−booking their appointment
199   newDentist // The dentist who will carry out the re−booked appointment
200   newDate // The date of the re−booked appointment
```

```
201  where
202   @grd1: rebookedAppointment ∈ booked // The appointment to be re−booked must already be booked
203   @grd2: appointmentDate(rebookedAppointment) − day ≥ 2 // The appointment to be re−booked must
            be 2 days or more in the future from the current date
204   @grd3: rebookingPatient ∈ patient // The re−booking patient must be a registered patient
205   @grd4: rebookingPatient ∈ loggedIn // The re−booking patient must be logged in
206   @grd5: newDentist ∈ dentist // The dentist who is carrying out the re−booked appointment must be a
```

```
        registered dentist
207     @grd6: (newDentist ↦ appointmentTreatment(rebookedAppointment)) ∈ qualifiedTreatment // The new
        dentist must be qualified to carry out the treatment
208     @grd7: newDate ∈ ℕ // The date specified must be a natural number
209     @grd8: newDate > day // The appointment must be re−booked for a future date
210     @grd9: ∀ a · a ∈ completed ∧ appointmentPatient(a) = rebookingPatient ∧ newDate ≤ (appointmentDate(a
        ) + 14) ⇒ appointmentTreatment(a) ≠ appointmentTreatment(rebookedAppointment) // A patient
        cannot re−book a treatment to new date if the new date is within 2 weeks of a completed appointment
         with the same treatment
211   then
212     @act1: appointmentDate(rebookedAppointment) := newDate // The new date is assigned to the
        appointment
213     @act2: appointmentDentist(rebookedAppointment) := newDentist // The new dentist is assigned to the
        appointment
214   end
215
216
217   /*A patient @patientCheckingIn can check in on the appointment @bookedAppointment date (and
218   the appointment state changes to checked−in).
219    */
220   event CheckInPatient
221    any
222     patientCheckingIn // The patient to check in
223     bookedAppointment // The appointment the patient is attending
224    where
225     @grd1:bookedAppointment ∈ booked // Ensures that the appointment is in the booked state.
226     @grd2:appointmentDate(bookedAppointment) = day /* Ensures that the appointment date of the
        booked appointment
227      * is equal to the current day */
228     @grd3:patientCheckingIn ∈ patient // The patient who is checking in is a patient stored on the system
229     @grd4:appointmentPatient(bookedAppointment) = patientCheckingIn // Validation that the
        appointment @appointment belongs to the patient who is checking in @patientCheckingIn.
230    then
231     @act1:booked := booked \ {bookedAppointment} // The booked appointment is removed from the
        booked set
232     @act2:checkedIn := checkedIn ∪ {bookedAppointment} // Add the booked appointment @appointment
        to the set of checkedIn appointments
233   end
234
235    /*After a checked−in patient @treatedPatient receives the required
236   treatment, a logged−in dentist @registeredDentist updates the appointment @appointment
237   to being completed.
238    */
239   event CompletedAppointment
240   any
241     finishedAppointment //the finished appointment to be completed.
242     registeredDentist //dentist who has completed the treatment.
243     treatedPatient //the patient who received the treatment.
244   where
245     @grd1:treatedPatient ∈ patient
246     @grd2:registeredDentist ∈ dentist
247     @grd3:registeredDentist ∈ loggedIn // Validation that the registeredDentist is a valid dentist who is also
        logged in
248     @grd4:finishedAppointment ∈ checkedIn //appointment @appointment is a member of the checkedIn
        state.
249     @grd5:appointmentPatient(finishedAppointment) = treatedPatient // Validation that the appointment
```

```
        @appointment belongs to the patient whose been treated @treatedPatient.
250     @grd6:appointmentTreatment(finishedAppointment) ∈ qualifiedTreatment[{registeredDentist}]
251     @grd7:appointmentDentist(finishedAppointment) = registeredDentist // Ensures that the dentist
          @registeredDentist was the dentist assigned to the appointment @finishedAppointment.
252   then
253     @act1:checkedIn := checkedIn \ {finishedAppointment} //remove the finished appointment
          @finishedAppointment from checkedIn set.
254     @act2:completed := completed ∪ {finishedAppointment} //add the given appointment to the set of
          completed appointments.
255   end
256
257   /*When the patients @absentPatient do not turn up to their appointment @missedAppointment,
258   an administrator @currentAdmin sets the appointment state to
259   missed.
260   */
261   event MissedAppointment
262   any
263     missedAppointment //the appointment which has been missed.
264     currentAdmin //currentAdmin managing/updating the appointments.
265   where
266     @grd1:missedAppointment ∈ booked // appointment @missedAppointment is booked.
267     @grd2:currentAdmin ∈ administrator // Ensures that the currentAdmin is a member of the
          administrator set.
268     @grd3:currentAdmin ∈ loggedIn // Ensures that the currentAdmin is logged in.
269     @grd4:appointmentDate(missedAppointment) = day //the current day is the same day as the
          appointment date.
270   then
271     @act1:booked := booked \ {missedAppointment} //remove the appointment from booked appointments
272     @act2:missed := missed ∪ {missedAppointment} //add the appointment @missedAppointment to missed
273   end
274 end
```

```
1  machine DentistSystem04
2  refines DentistSystem03
3  sees DentistContext03
4  variables
5    registeredUser // The set of users registered to the system
6    passwords // The set of passwords stored in the system
7    loggedIn // The set of registered users that are logged in
8    loggedOut // The set of registered users that are logged out
9    administrator // The set of registered administrators
10   dentist // The set of registered dentists
11   patient // The set of registered patients
12   qualifiedTreatment // The relationship which maps each dentist to the treatments that they are qualified
       to perform
13   patientNHSNum // The relationship which maps each patient to their unique NHS number
14   day // The current day
15   appointment // The set of all existing appointments
16   appointmentPatient // The relationship which maps each appointment to the patient whose appointment
       it is
17   appointmentTreatment // The relationship which maps each appointment to the treatment to be
       performed
18   appointmentDate // The relationship which maps each appointment to the day it will take place
19   appointmentDentist // The relationship which maps each appointment to the dentist which will perform
```

```
        the specified treatment
20  booked // The set of booked appointments
21  checkedIn // The set of checked-in appointments
22  cancelled // The set of cancelled appointments
23  missed // The set of missed appointments
24  completed // The set of completed appointments
25 events
26
27  event INITIALISATION
28  extends INITIALISATION
29  end
30 /*
31  * The new user @user is registered to the system as an administrator with the password @password
32  * and they are registered by a logged-in administrator @registeringAdmin
33  */
34  event registerAdmin extends registerAdmin
35  end
36
37  /*
38  * The new user @user is registered to the system as a dentist with the password @password
39  * The new user @user is registered by a logged-in administrator @registeringAdmin.
40  */
41  event registerDentist extends registerDentist
42  end
43
44  /*
45  * Register a new user @user to the system with the password @password as a patient with
46  * the NHS number @NHSNum. A patient can register themselves to the system.
47  */
48  event registerPatient extends registerPatient
49  end
50
51  /*
52  * User @user changes their password from their current password @currentPass to a new password
        @newPassword
53  */
54  event changePassword extends changePassword
55  end
56
57  /*
58  * User @user logs in using the password @password
59  */
60  event logIn extends logIn
61  end
62
63  /*
64  * User @user logs out
65  */
66  event logOut extends logOut
67  end
68
69  /*
70  * The current day is increased by one
71  */
72  event nextDay extends nextDay
73  end
```

```
74
75  /*
76   * A registered, logged-in patient @newPatient can book an appointment @newAppointment for the
         treatment @newTreatment
77   * with the dentist @newDentist on a specified date @newDay
78   */
79  event bookAppointment extends bookAppointment
80  end
81
82  /*
83   * A registered, logged-in patient @cancellingPatient can cancel an appointment
         @cancelledAppointment
84   */
85  event cancelAppointment extends cancelAppointment
86  end
87
88  /*
89   * A registered, logged-in patient @rebookingPatient can re-book their appointment
         @rebookedAppointment
90   * and change the dentist @newDentist and/or date @newDate
91   */
92  event rebookAppointment extends rebookAppointment
93  end
94
95  /*A patient @patientCheckingIn can check in on the appointment @bookedAppointment date (and
96  the appointment state changes to checked-in).
97   */
98  event CheckInPatient extends CheckInPatient
99  end
100
101  /*After a checked-in patient @treatedPatient receives the required
102 treatment, a logged-in dentist @registeredDentist updates the appointment @appointment
103 to being completed.
104  */
105 event CompletedAppointment extends CompletedAppointment
106 end
107
108 /*When the patients @absentPatient do not turn up to their appointment @missedAppointment,
109 an administrator @currentAdmin sets the appointment state to
110 missed.
111 */
112 event MissedAppointment extends MissedAppointment
113 end
114
115 /*
116  * logged in dentist or administrator checking the appointment records of patient
117  */
118 event adminOrDentistCheckAppointment
119 any
120  user //the user that is checking the appointment records
121  patientNHSNumber //the NHS number of the patient whose record is getting checked
122  result //the appointment result of a patient
123 where
124  @grd1:user ∈ loggedIn //check if the user is logged in
125  @grd2:user ∈ administrator ∪ dentist //check if the user is administrator or dentist
126  @grd3:patientNHSNumber ∈ ran(patientNHSNum) // check that NHSnumber is connected with a patient
```

```
127   @grd4:result = {a | a ∈ appointment ∧ appointmentPatient(a) = patientNHSNum∼(patientNHSNumber)}
128   // gets the all the appointments which belong to the patient which has the given NHS Number (
          patientNHSNumber)
129  end
130  /*
131   * logged in patient checks appointments booked on a specific day
132   */
133  event patientCheckAppointment
134  any
135   user //the user that is checking the appointment records
136   specificDay // the specific day that the patient chooses
137   result // the booked appointments for this patient for a specific day
138  where
139   @grd1:user ∈ loggedIn ∩ patient //check if the user is logged in and is a patient
140   @grd2:specificDay ∈ ℕ //the day must be a natural number
141   @grd3:specificDay ≥ day // the day cannot be in the past
142   @grd4:result = {a | a ∈ booked ∧ appointmentPatient(a) = user ∧ appointmentDate(a) = specificDay} //
          gets the appointments such that
143   // the appointment is booked, belongs to this patient, and is on the specific day
144  end
145  /*
146   * logged in patient views treatment record
147   */
148  event patientViewTreatment
149  any
150   user //the user that is checking the appointment records
151   result // the treatments which this patient has received
152  where
153   @grd1:user ∈ loggedIn ∩ patient //check if the user is logged in and is a patient
154   @grd2:result = {appt ↦ treatment ↦ date | appt ∈ appointmentPatient∼[{user}] ∩ completed ∧ treatment
          = appointmentTreatment(appt) ∧ date = appointmentDate(appt)}
155   // returns the set of appointments which the patient has completed, along with the associated treatment
          and date which came with each appointment.
156  end
157 end
```