Relatório Final

Métodos Formais em Engenharia de Software
2018/2019

Mestrado Integrado em Engenharia Informática e Computação

Agenda Viral

Bárbara Sofia Silva up201505628@fe.up.pt
Julieta Frade up201506530@fe.up.pt

7 de Janeiro 2019

Conteúdo

1 Descrição do Sistema

Este projeto tem como finalidade modelar uma agenda de eventos culturais, à semelhança da plataforma **Agenda Viral**, mas ainda com a possibilidade de comprar bilhetes. Esta tem como objetivo informar o utilizador do que se passa à sua volta, ajudando-o a descobrir os seus eventos preferidos: concertos, exposições, festas, workshops, conferências, etc.

De forma a prestar uma ajuda mais precisa e rápida, é possível aplicar filtros à pesquisa, respetivamente por distrito, cidade, categoria e data. Cada utilizador pode também sugerir eventos que ainda não estejam presentes na aplicação através de um formulário e comprar bilhetes para os eventos já disponíveis na plataforma.

Por último, cabe ao administrador adicionar eventos à agenda, assim como aceitar ou rejeitar os eventos sugeridos por um utilizador habitual. Pode também visualizar algumas estatísticas da sua aplicação.

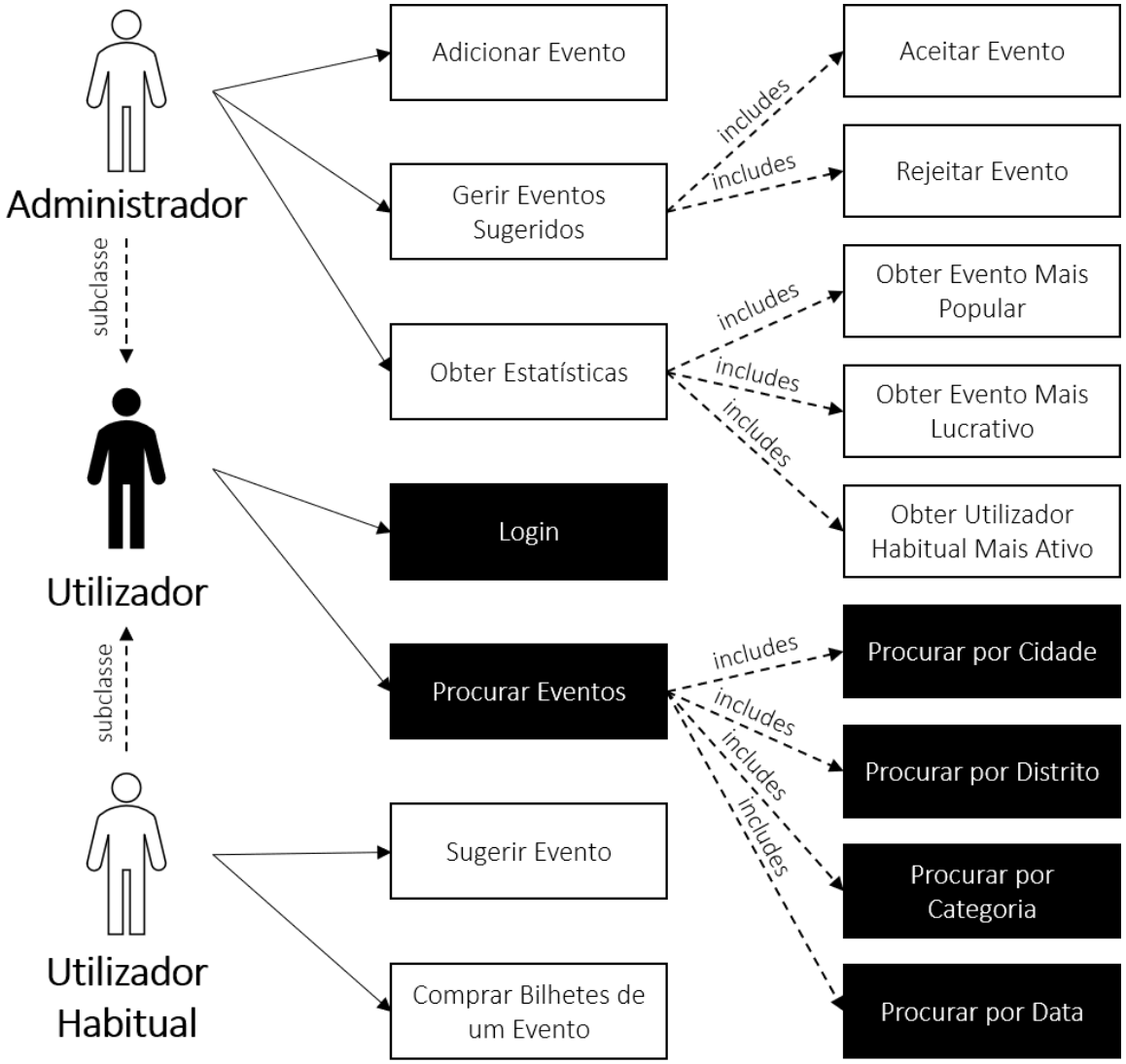
1.1 Lista de Requisitos

Para executar o sistema descrito, são necessários os seguintes requisitos:

ID	**Prioridade**	**Descrição**
R01	Obrigatória	Um utilizador pode iniciar sessão
R02	Obrigatória	Um administrador pode adicionar novos eventos
R03	Obrigatória	Um administrador pode aceitar eventos propostos
R04	Obrigatória	Um administrador pode rejeitar eventos propostos
R05	Obrigatória	Um utilizador habitual pode propor eventos
R06	Obrigatória	Um utilizador habitual pode comprar bilhetes para um evento
R07	Obrigatória	Um utilizador pode procurar eventos por distrito
R08	Obrigatória	Um utilizador pode procurar eventos por cidade
R09	Obrigatória	Um utilizador pode procurar eventos por categoria
R10	Opcional	Um utilizador pode procurar eventos por data
R11	Opcional	Um utilizador pode procurar eventos por múltiplos filtros
R12	Obrigatória	Um administrador pode obter o evento mais popular
R13	Opcional	Um administrador pode obter o evento mais lucrativo
R14	Opcional	Um administrador pode obter o utilizador habitual mais ativo

2 Modelo Visual UML

2.1 Modelo de Caso de Uso



Cenário	Iniciar sessão
Descrição	Um utilizador iniciar sessão no sistema
Pré-Condições	1. O email existir no sistema 2. A password ter mais que 8 caracteres
Pós-Condições	1. O email corresponder à password 2. A *token* de sessão estar preenchido OU 1. O email não corresponder à password
Passos	1. No menu inicial, *Login*, preencher os campos necessários
Exceções	1. O utilizador terminar o programa

Cenário	Adicionar um evento
Descrição	Um administrador adicionar um evento
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. A categoria existir no sistema 4. A cidade existir no sistema 5. Não existir nenhum evento com o mesmo título, categoria, data de início, data de fim e cidade no sistema
Pós-Condições	1. O evento existir na lista de eventos
Passos	1. No menu principal escolher a opção *Add Event* 2. Preencher os campos necessários
Exceções	1. O utilizador terminar o programa

Cenário	Aceitar um evento proposto
Descrição	Um administrador aceitar evento proposto
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. O evento existir na lista de eventos propostos 4. O evento não existir na lista de eventos
Pós-Condições	1. O evento existir na lista de eventos 2. O evento não existir na lista de eventos propostos
Passos	1. No menu principal escolher a opção *Proposed Events* 2. Inserir o identificador do evento 3. Aceitar o evento
Exceções	1. O utilizador terminar o programa

Cenário	Rejeitar um evento proposto
Descrição	Um administrador rejeitar evento proposto
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. O evento existir na lista de eventos propostos
Pós-Condições	1. O eventos não existir na lista de eventos propostos
Passos	1. No menu principal escolher a opção *Proposed Events* 2. Inserir o identificador do evento 3. Rejeitar o evento
Exceções	1. O utilizador terminar o programa

Cenário	Propor um evento
Descrição	Um utilizador habitual propor um evento
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser do tipo Regular 3. A categoria existir no sistema 4. A cidade existir no sistema 5. Não existir nenhum evento com o mesmo título, categoria, data de início, data de fim e cidade no sistema
Pós-Condições	1. O evento existir na lista de eventos propostos
Passos	1. No menu principal escolher a opção *Propose Event* 2. Preencher os campos necessário
Exceções	1. O utilizador terminar o programa

Cenário	Comprar bilhetes
Descrição	Um utilizador habitual comprar bilhetes para evento
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser do tipo *Regular* 3. O evento ter bilhetes disponíveis suficientes 4. O evento estar no estado *Available* 5. O utilizador ter um balanço igual ou superior ao preço do bilhetes
Pós-Condições	1. O número de bilhetes vendidos para o evento incrementar as unidades de bilhetes comprados 2. O número de bilhetes comprados pelo utilizador incrementar as unidades de bilhetes comprados 3. O balanço do utilizador decrementar a soma do preço dos bilhetes comprados
Passos	1. Procurar um eventos 2. Selecionar o evento pelo seu id 3. Escolher a opção *Buy Tickets* 4. Introduzir o número de bilhetes pretendidos 5. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Procurar por distrito
Descrição	Utilizador procurar eventos por distrito
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. O distrito escolhido existir no sistema
Pós-Condições	1. Os eventos retornados serem no distrito escolhido
Passos	1. No menu principal escolher a opção *Find by District"* 2. Escolher o distrito 3. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Procurar por cidade
Descrição	Utilizador procurar eventos por cidade
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. A cidade escolhida existir no sistema
Pós-Condições	1. Os eventos retornados serem na cidade escolhido
Passos	1. No menu principal escolher a opção *Find by City* 2. Escolher a cidade 3. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Procurar por categoria
Descrição	Utilizador procurar eventos por categoria
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. A categoria escolhida existir no sistema
Pós-Condições	1. Os eventos retornados serem da categoria escolhida
Passos	1. No menu principal escolher a opção *Find by Category* 2. Escolher a categoria 3. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Procurar por data
Descrição	Utilizador procurar eventos por data
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. A data escolhida ser válida
Pós-Condições	1. Os eventos retornados ocorrem no dia escolhido
Passos	1. No menu principal escolher a opção *Find by Date* 2. Escolher a data 3. Submeter
Exceções	1. O utilizador terminar o programa

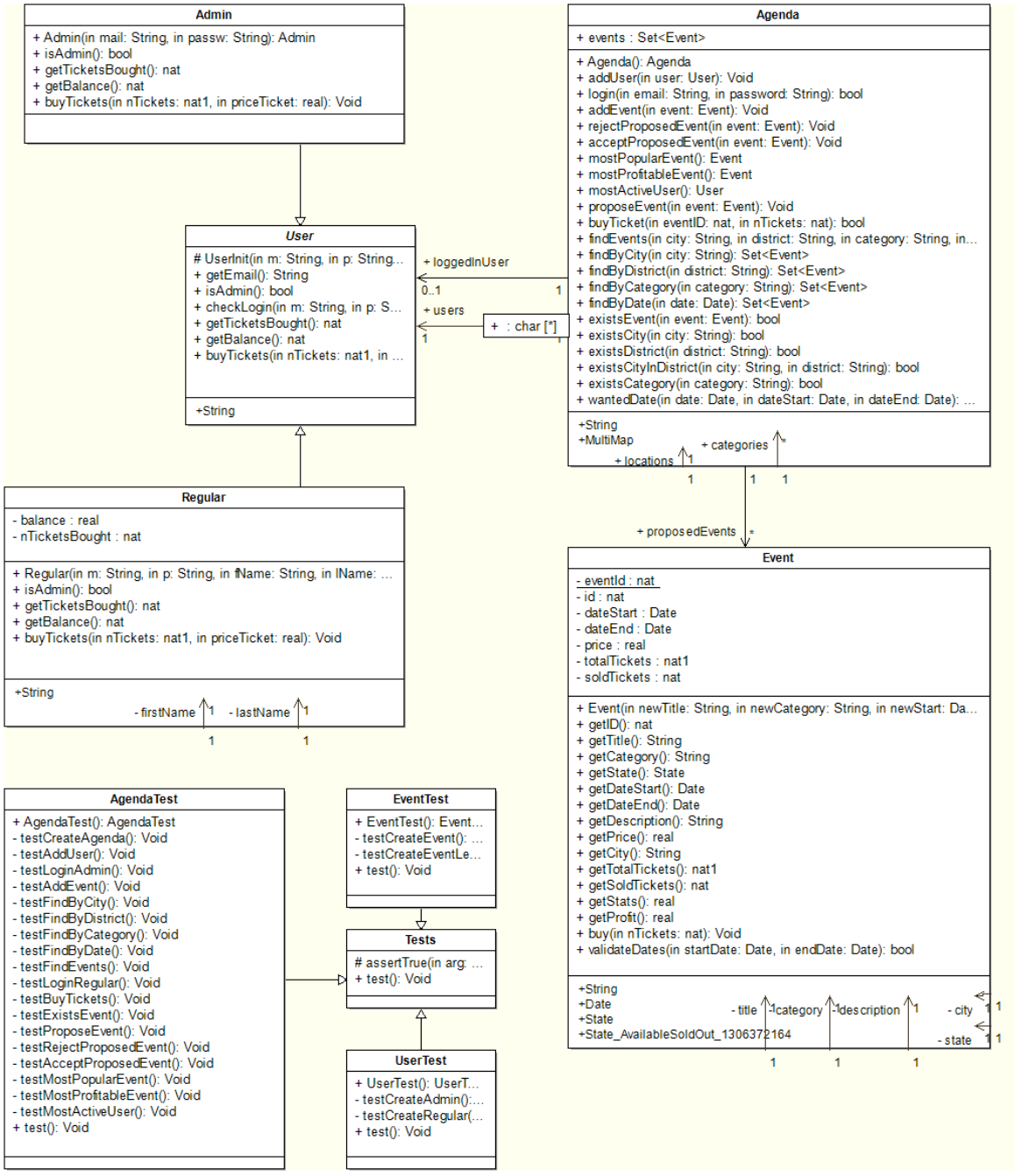
Cenário	Procurar por vários filtros
Descrição	Utilizador procurar eventos por vários filtros
Pré-Condições	1. O utilizador existir no sistema 2. Existir pelo menos um evento no sistema 3. Escolher pelo menos um filtro 4. Não pode escolher os filtros cidade e distrito na mesma procura 5. O distrito escolhido existir no sistema 6. A cidade escolhida existir no sistema 7. A categoria escolhida existir no sistema 8. A data escolhida ser válida
Pós-Condições	1. Os eventos retornados obedecerem aos filtros escolhidos
Passos	1. No menu principal escolher a opção *Find by Multiple Filters* 2. Preencher os filtros desejados 3. Submeter
Exceções	1. O utilizador terminar o programa

Cenário	Obter evento mais popular
Descrição	Um administrador obter o evento mais popular, ou seja, o que tem a maior percentagem de bilhetes vendidos face ao número total de lugares
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. Existir pelo menos um evento na lista de eventos
Pós-Condições	Nenhuma
Passos	1. No menu principal escolher a opção *Most Popular Event*
Exceções	1. O utilizador terminar o programa

Cenário	Obter evento mais lucrativo
Descrição	Um administrador obter o evento mais lucrativo
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. Existir pelo menos um evento na lista de eventos
Pós-Condições	Nenhuma
Passos	1. No menu principal escolher a opção *Most Profitable Event*
Exceções	1. O utilizador terminar o programa

Cenário	Obter utilizador habitual mais ativo
Descrição	Um administrador obter o utilizador habitual mais ativo, ou seja, aquele que comprou mais bilhetes no total
Pré-Condições	1. O utilizador estar com sessão iniciada 2. O utilizador ser administrador 3. Existir pelo menos um utilizador na lista de eventos
Pós-Condições	Nenhuma
Passos	1. No menu principal escolher a opção *Most Active User*
Exceções	1. O utilizador terminar o programa

2.2 Modelo de Classe



3 Modelo Formal VDM++

3.1 Classe Agenda

```
class Agenda

  types

    public String = seq of char;
    public MultiMap = map String to set of String;

  instance variables

    -- Agenda's list of categories
    public categories: set of String := {};

    -- Agenda's list of locations (multimap district->cities)
    public locations: MultiMap := {|->};

    -- Agenda's map of users
    public users: map seq of char to User := {|->};

    -- Agenda's list of events
    public events: set of Event := {};

    -- Logged in user. If nil means there is no user logged in.
    public loggedInUser: [User] := nil;
    -- TODO: depois por a private

    -- Agenda's list of proposed events waiting for admin's approval
    public proposedEvents: set of Event := {};

  operations

    -- Constructor
    public Agenda: () ==> Agenda
    Agenda() == (
      categories :=  {"Concertos", "Exposicoes", "Gastronomia", "Moda"
    , "Desporto", "Natureza"};
      locations := {"Porto" |-> {"Porto", "Matosinhos", "Maia", "Vila
     Nova de Gaia"},
                    "Lisboa" |-> {"Lisboa", "Amadora", "Cascais", "
     Sintra"},
                    "Faro" |-> {"Faro", "Albufeira", "Portimao"}};

      return self;
    );
```

```
41
42      -- Adds user to Agenda's list of users
43      public addUser: User ==> ()
44      addUser(user) == users := users ++ {user.getEmail() |-> user}
45      pre user.getEmail() not in set dom users
46      post user.getEmail() in set dom users;
47
48      -- Logs the user in. Returns true if successful and updates the
    loggedInUser variable
49      public login: String * String ==> bool
50      login(email, password) == (
51        dcl user: User := users(email);
52
53        -- verifies if password matches to email
54        if user.checkLogin(email, password)
55        then (
56            loggedInUser := user;
57            return true;
58        );
59        return false;
60      )
61      pre email in set dom users and len password > 8
62      post (RESULT = true and loggedInUser <> nil) or RESULT = false;
63
64      -- *ADMINISTRATOR ONLY*
65
66      -- Adds event to Agenda's list of events
67      public addEvent: Event ==> ()
68      addEvent(event) == events := events union {event}
69      pre loggedInUser <> nil and loggedInUser.isAdmin() and
    existsCategory(event.getCategory()) and existsCity(event.getCity()
    ) and not existsEvent(event)
70      post event in set events;
71
72      -- Removes event from Agenda's list of proposed events
73      public rejectProposedEvent: Event ==> ()
74      rejectProposedEvent(event) == proposedEvents := proposedEvents \ {
    event}
75      pre loggedInUser <> nil and loggedInUser.isAdmin() and event in
    set proposedEvents
76      post event not in set proposedEvents;
77
78      -- Adds event to Agenda's list of events and removes from Agenda's
     list of proposed events
79      public acceptProposedEvent: Event ==> ()
80      acceptProposedEvent(event) == (
81        events := events union {event};
82        proposedEvents := proposedEvents \ {event};
83      )
```

```
84      pre loggedInUser <> nil and loggedInUser.isAdmin() and event in
        set proposedEvents and event not in set events
85      post event not in set proposedEvents and event in set events;
86
87      -- Returns most popular event
88      public mostPopularEvent: () ==> Event
89      mostPopularEvent () == (
90        dcl popularEvent: [Event] := nil;
91        dcl value: real := 0;
92
93        for all event in set events do (
94          if(event.getStats() >= value)
95          then (
96            value := event.getStats();
97            popularEvent := event;
98          );
99        );
100
101       return popularEvent;
102     )
103     pre loggedInUser <> nil and loggedInUser.isAdmin() and events <>
        {};
104
105     -- Returns most profitable event
106     public mostProfitableEvent: () ==> Event
107     mostProfitableEvent () == (
108       dcl profitableEvent: [Event] := nil;
109       dcl value: real := 0;
110
111       for all event in set events do (
112         if(event.getProfit() >= value)
113         then (
114           value := event.getProfit();
115           profitableEvent := event;
116         );
117       );
118
119       return profitableEvent;
120     )
121     pre loggedInUser <> nil and loggedInUser.isAdmin() and events <>
        {};
122
123     -- Returns most active user
124     public mostActiveUser: () ==> User
125     mostActiveUser () == (
126       dcl allUsers: set of User := rng users;
127       dcl activeUser: [User] := nil;
128       dcl value: nat := 0;
129
```

```
130       for all user in set allUsers do (
131         if(isofclass(Regular, user))
132         then (
133           if(user.getTicketsBought() >= value)
134           then (
135             value := user.getTicketsBought();
136             activeUser := user;
137           );
138         );
139       );
140 
141       return activeUser;
142     )
143     pre loggedInUser <> nil and loggedInUser.isAdmin() and users <>
    {|->};
144 
145     -- *REGULAR USER ONLY*
146 
147     -- Adds event to Agenda's list of proposed event
148     public proposeEvent: Event ==> ()
149     proposeEvent(event) == proposedEvents := proposedEvents union {
    event}
150     pre loggedInUser <> nil and not loggedInUser.isAdmin() and
    existsCategory(event.getCategory()) and existsCity(event.getCity()
    ) and not existsEvent(event)
151     post event in set proposedEvents;
152 
153     -- Returns if the purchase was successful
154     public buyTicket: nat * nat ==> bool
155     buyTicket(eventID, nTickets) == (
156       dcl eventBought : Event;
157       for all event in set events do (
158           if event.getID() = eventID
159           then eventBought := event;
160       );
161       -- verifies if there are enough tickets to sell
162       if eventBought.getTotalTickets() >= eventBought.getSoldTickets()
     + nTickets and
163           loggedInUser.getBalance() >= nTickets * eventBought.getPrice
    ()
164       then (
165         loggedInUser.buyTickets(nTickets, eventBought.getPrice());
166         eventBought.buy(nTickets);
167         return true;
168       )
169       else return false
170     )
171     pre loggedInUser <> nil and not loggedInUser.isAdmin();
172 
```

```
173     -- *FIND EVENTS*
174
175     -- Returns a set of events based on filters
176     public findEvents: String * String * String * [Event`Date] ==> set
        of Event
177     findEvents(city, district, category, date) == (
178       dcl foundEvents : set of Event := {};
179
180       if city <> ""
181         then foundEvents := findByCity(city)
182       elseif district <> ""
183         then foundEvents := findByDistrict(district);
184
185       if category <> ""
186       then (
187         if city = "" and district = ""
188         then foundEvents := findByCategory(category)
189         else foundEvents := foundEvents inter findByCategory(category)
    ;
190       );
191
192       if date <> nil
193       then (
194         if city = "" and district = "" and category = ""
195         then foundEvents := findByDate(date)
196         else foundEvents := foundEvents inter findByDate(date);
197       );
198
199       return foundEvents;
200     )
201     pre loggedInUser <> nil and events <> {} and
202         not (city <> "" and district <> "") and
203         not (city = "" and district = "" and category = "" and date =
    nil)
204     post (if city <> "" then forall e in set RESULT & e.getCity() =
    city else true) and
205           (if district <> "" then forall e in set RESULT &
    existsCityInDistrict(e.getCity(), district) else true) and
206           (if category <> "" then forall e in set RESULT & e.
    getCategory() = category else true) and
207           (if date <> nil then forall e in set RESULT & wantedDate(date
    , e.getDateStart(), e.getDateEnd()) else true);
208
209     -- by city
210     public findByCity: String ==> set of Event
211     findByCity(city) == (
212       dcl cityEvents : set of Event := {};
213       for all event in set events do (
214           if event.getCity() = city
```

```
215             then cityEvents := cityEvents union {event}
216       );
217       return cityEvents;
218     )
219     pre loggedInUser <> nil and existsCity(city) and events <> {}
220     post forall e in set RESULT & e.getCity() = city;
221
222     -- by district
223     public findByDistrict: String ==> set of Event
224     findByDistrict(district) == (
225       dcl districtEvents : set of Event := {};
226       if existsDistrict(district)
227       then (
228         for all event in set events do (
229           if existsCityInDistrict(event.getCity(), district)
230           then districtEvents := districtEvents union {event}
231         );
232         return districtEvents;
233       )
234       else return {}
235     )
236     pre loggedInUser <> nil and events <> {}
237     post forall e in set RESULT & existsCityInDistrict(e.getCity(),
    district);
238
239
240     -- by category
241     public findByCategory: String ==> set of Event
242     findByCategory(category) == (
243       dcl categoryEvents : set of Event := {};
244       for all event in set events do (
245           if category = event.getCategory()
246           then categoryEvents := categoryEvents union {event}
247       );
248       return categoryEvents;
249     )
250     pre loggedInUser <> nil and existsCategory(category) and events <>
      {}
251     post forall e in set RESULT & e.getCategory() = category;
252
253     -- by date
254     public findByDate: Event`Date ==> set of Event
255     findByDate(date) == (
256       dcl dateEvents : set of Event := {};
257       for all event in set events do (
258           if wantedDate(date, event.getDateStart(), event.getDateEnd()
    )
259           then dateEvents := dateEvents union {event}
260       );
```

```
261       return dateEvents;
262     )
263     pre loggedInUser <> nil and events <> {}
264     post forall e in set RESULT & wantedDate(date, e.getDateStart(), e
    .getDateEnd());
265
266     -- *AUX*
267
268     -- Returns if events exists in the agenda's list of events
269     public pure existsEvent: Event ==> bool
270     existsEvent(event) == (
271       for all e in set events do(
272         if (event.getTitle() = e.getTitle() and
273             event.getCategory() = e.getCategory() and
274             event.getDateStart() = e.getDateStart() and
275             event.getDateEnd() = e.getDateEnd() and
276             event.getCity() = e.getCity())
277         then return true
278       );
279       return false;
280     );
281
282     -- Returns if city exists in the agenda's list of cities
283     public pure existsCity: String ==> bool
284     existsCity(city) == (
285       dcl citiesSet : set of set of String := rng locations;
286       dcl cities: set of String := {};
287       for all cs in set citiesSet do(
288         cities := cities union cs
289       );
290       return city in set cities;
291     )
292     pre locations <> {|->};
293
294     -- Returns if district exists in the agenda's list of districts
295     public pure existsDistrict: String ==> bool
296     existsDistrict(district) == (
297       dcl districts : set of String := dom locations;
298       return district in set districts;
299     )
300     pre locations <> {|->};
301
302     -- Returns if city exists in a certain district
303     public pure existsCityInDistrict: String * String ==> bool
304     existsCityInDistrict(city, district) == (
305       dcl districtCities : set of String := locations(district);
306       return city in set districtCities;
307     )
308     pre existsDistrict(district) and locations <> {|->};
```

```
309
310     -- Returns if category exists in the agenda's list of categories
311     public pure existsCategory: String ==> bool
312     existsCategory(category) == return category in set categories
313     pre categories <> {};
314
315     -- Returns if date belongs to the interval [dateStart,dateEnd]
316     public pure wantedDate: Event`Date * Event`Date * Event`Date ==>
        bool
317     wantedDate(date, dateStart,dateEnd) == (
318       dcl natDateStart : nat := dateStart.year * 10000 + dateStart.
        month * 100 + dateStart.day;
319       dcl natDateEnd : nat := dateEnd.year * 10000 + dateEnd.month *
        100 + dateEnd.day;
320       dcl natDate : nat := date.year * 10000 + date.month * 100 + date
        .day;
321
322       return (natDateStart <= natDate) and (natDate <= natDateEnd);
323     );
324
325 end Agenda
```

3.2 Classe Event

```
1  class Event
2
3    types
4
5      public String = seq of char;
6
7      -- Represents a date
8      public Date :: day : nat1
9                     month: nat1
10                    year : nat1
11
12     -- Ensures a valid date
13     inv date == if date.year mod 400 = 0 or (date.year mod 100 <> 0
       and date.year mod 4 = 0)
14                 then date.month <= 12 and date.day <= [31, 29, 31, 30,
       31, 30, 31, 31, 30, 31, 30, 31](date.month)
15                 else date.month <= 12 and date.day <= [31, 28, 31, 30,
       31, 30, 31, 31, 30, 31, 30, 31](date.month);
16
17     -- Represents the event's state
18     public State = <Available> | <SoldOut>;
19
20   instance variables
```

```
21
22      private static eventId : nat := 1;
23
24      -- Event's id
25      private id : nat;
26
27      -- Event's title
28      private title: String;
29
30      -- Event's category
31      private category: String;
32
33      -- Event's state
34      private state: State;
35
36      -- Event's starting date
37      private dateStart: Date;
38
39      -- Event's ending date
40      private dateEnd: Date;
41
42      -- Event's description
43      private description: String;
44
45      -- Event's price
46      private price: real;
47
48      -- Event's city
49      private city: String;
50
51      -- Event's total amount of tickets
52      private totalTickets: nat1;
53
54      -- Event's amount of sold tickets
55      private soldTickets: nat;
56
57      -- Ensures inexistence of overbooking
58      inv totalTickets >= soldTickets;
59
60  operations
61
62      -- Constructor
63      public Event: String * String * Date * Date * String * real *
        String * nat1 ==> Event
64      Event(newTitle, newCategory, newStart, newEnd, newDescription,
        newPrice, newCity, newTotal) == (
65          id := eventId;
66          eventId := eventId + 1;
67
```

```
68        title := newTitle;
69        category := newCategory;
70        state := <Available>;
71        dateStart := newStart;
72        dateEnd := newEnd;
73        description := newDescription;
74        price := newPrice;
75        city := newCity;
76        totalTickets := newTotal;
77        soldTickets := 0;
78
79        return self;
80      )
81      pre len newTitle > 0 and validateDates(newStart, newEnd)
82      post title = newTitle;
83
84      -- Returns event's id
85      public pure getID: () ==> nat
86      getID () == return id;
87
88      -- Returns event's title
89      public pure getTitle: () ==> String
90      getTitle () == return title;
91
92      -- Returns event's category
93      public pure getCategory: () ==> String
94      getCategory () == return category;
95
96      -- Returns event's state
97      public pure getState: () ==> State
98      getState () == return state;
99
100     -- Returns event's starting date
101     public pure getDateStart: () ==> Date
102     getDateStart () == return dateStart;
103
104     -- Returns event's ending date
105     public pure getDateEnd: () ==> Date
106     getDateEnd () == return dateEnd;
107
108     -- Returns event's description
109     public pure getDescription: () ==> String
110     getDescription () == return description;
111
112     -- Returns event's price
113     public pure getPrice: () ==> real
114     getPrice () == return price;
115
116     -- Returns event's city
```

```
117     public pure getCity: () ==> String
118     getCity () == return city;
119 
120     -- Returns event's total tickets
121     public pure getTotalTickets: () ==> nat1
122     getTotalTickets () == return totalTickets;
123 
124     -- Returns event's sold tickets
125     public pure getSoldTickets: () ==> nat
126     getSoldTickets () == return soldTickets;
127 
128     -- Returns event's percentage of tickets sold
129     public pure getStats: () ==> real
130     getStats () == return soldTickets * 100 / totalTickets;
131 
132     -- Returns event's total money raised
133     public pure getProfit: () ==> real
134     getProfit () == return soldTickets * price;
135 
136     -- Updates event's sold tickets and state if needed
137     public buy: nat ==> ()
138     buy (nTickets) == (
139       soldTickets := soldTickets + nTickets;
140       if(soldTickets = totalTickets)
141       then state := <SoldOut>;
142     )
143     pre totalTickets >= soldTickets + nTickets and state = <Available>
144     post soldTickets = soldTickets~ + nTickets;
145 
146     -- *AUX*
147     -- Verifies if the start date is before the end date
148     public pure validateDates: Event`Date * Event`Date ==> bool
149     validateDates(startDate,endDate) == (
150       dcl natDateStart : nat := startDate.year * 10000 + startDate.
    month * 100 + startDate.day;
151       dcl natDateEnd : nat := endDate.year * 10000 + endDate.month *
    100 + endDate.day;
152 
153       return (natDateStart <= natDateEnd);
154     );
155 
156 end Event
```

3.3 Classe User

```
1 class User
2 
```

```
3   types
4
5     public String = seq of char;
6
7   instance variables
8
9     -- User's email
10    protected email: String := "";
11
12    -- User's password
13    protected password: String := "";
14
15  operations
16
17    -- Constructor
18    protected UserInit : String * String ==> ()
19    UserInit(m, p) == (
20      email := m;
21      password := p;
22    )
23    pre len m >= 5 and len p >= 8
24    post email = m and password = p;
25
26    -- Returns the user email
27    public pure getEmail: () ==> String
28    getEmail () == return email ;
29
30    -- Returns if user is admin
31    public pure isAdmin: () ==> bool
32    isAdmin() == is subclass responsibility;
33
34    -- Checks if the email and password combination matches
35    public checkLogin: String * String ==> bool
36    checkLogin(m, p) == return email = m and password = p
37    post RESULT = (email = m and password = p);
38
39    -- Returns number of bought tickets
40    public pure getTicketsBought: () ==> nat
41    getTicketsBought() == is subclass responsibility;
42
43    -- Returns balance
44    public pure getBalance: () ==> nat
45    getBalance() == is subclass responsibility;
46
47    -- Makes a purchase
48    public buyTickets: nat1 * real ==> ()
49    buyTickets(nTickets, priceTicket) == is subclass responsibility;
50
51  end User
```

3.4 Classe Admin

```
class Admin is subclass of User

  operations

    -- Constructor
    public Admin : String * String ==> Admin
    Admin(mail, passw) == (
      UserInit(mail, passw);
      return self;
    );

    -- Returns if user is admin
    public pure isAdmin: () ==> bool
    isAdmin() == return true;

    -- Returns number of bought tickets
    public pure getTicketsBought: () ==> nat
    getTicketsBought() == is not yet specified;

    -- Returns balance
    public pure getBalance: () ==> nat
    getBalance() == is not yet specified;

    -- Makes a purchase
    public buyTickets: nat1 * real ==> ()
    buyTickets(nTickets, priceTicket) == is not yet specified;

end Admin
```

3.5 Classe Regular

```
class Regular is subclass of User

  types

      public String = seq of char;

  instance variables

    private firstName: String;
    private lastName: String;
    private balance: real;
    private nTicketsBought: nat;

```

```
14   operations
15
16     -- Constructor
17     public Regular : String * String * String * String * real * real
      ==> Regular
18     Regular(m, p, fName, lName, b, nt) == (
19       UserInit(m, p);
20       firstName := fName;
21       lastName := lName;
22       balance := b;
23       nTicketsBought := nt;
24       return self;
25     );
26
27     -- Returns if user is admin
28     public pure isAdmin: () ==> bool
29     isAdmin() == return false;
30
31     -- Returns number of bought tickets
32     public pure getTicketsBought: () ==> nat
33     getTicketsBought() == return nTicketsBought;
34
35     -- Returns balance
36     public pure getBalance: () ==> nat
37     getBalance() == return balance;
38
39     -- Makes a purchase
40     public buyTickets: nat1 * real ==> ()
41     buyTickets(nTickets, priceTicket) == (
42       balance := balance - (nTickets * priceTicket);
43       nTicketsBought := nTicketsBought + nTickets
44     )
45     pre balance >= nTickets * priceTicket
46     post balance = balance~ - (nTickets * priceTicket);
47
48 end Regular
```

4 Validação do Modelo

4.1 Classes de Teste

4.1.1 Classe Tests

```
class Tests
  instance variables

  operations

    protected assertTrue: bool ==> ()
    assertTrue(arg) == return
    pre arg;

    public test: () ==> ()
    test() == (
      dcl agendaTest: AgendaTest := new AgendaTest();
      dcl eventTest: EventTest := new EventTest();
      dcl userTest: UserTest := new UserTest();
      agendaTest.test();
      eventTest.test();
      userTest.test();
    );

end Tests
```

4.1.2 Classe AgendaTest

```
class AgendaTest is subclass of Tests

  instance variables
    agenda: Agenda := new Agenda();
    userAdmin: Admin := new Admin("julieta@gmail.com", "julieta12345")
    ;
    user1: Regular := new Regular("sofia@gmail.com", "sofia12345", "
    Sofia", "Silva", 500, 30);
    user2: Regular := new Regular("bibi@gmail.com", "bibi12345", "
    Beatriz", "Baldaia", 200, 5);
    user3: Regular := new Regular("carlos@gmail.com", "carlos12345", "
    Carlos", "Freitas", 800, 10);
    user4: Regular := new Regular("vicente@gmail.com", "vicente12345",
    "Vicente", "Espinha", 100, 50);
    event1: Event := new Event("Twenty One Pilots", "Concertos",
```

```
    mk_Event`Date(17,3,2019), mk_Event`Date(17,3,2019), "Twenty One
    Pilots, o aclamado duo norte-americano constituido por Tyler
    Joseph e Josh Dun. The Bandito Tour sera a digressao mundial de
    apresentacao do album, com a estreia da banda ao vivo em Portugal,
     dia 17 de Marco, na Altice Arena.", 42, "Lisboa", 20000);
11   event2: Event := new Event("EXO", "Concertos", mk_Event`Date
    (20,5,2019), mk_Event`Date(20,5,2019), "EXO e um grupo masculino
    sino-coreano de Seul. Estreia em Portugal dia 20 de Maio, na
    Altice Arena.", 30, "Lisboa", 20000);
12   event3: Event := new Event("Aberturas: Tom Emerson em conversa com
     o arquivo Alvaro Siza", "Exposicoes", mk_Event`Date(6,1,2019),
    mk_Event`Date(6,2,2019), "Visita orientada a exposicao por Matilde
     Seabra, educadora. Localizacao: Biblioteca de Serralves", 2.5, "
    Porto", 200);
13   event4: Event := new Event("Brunch Mercearia Bio", "Gastronomia",
    mk_Event`Date(5,1,2019), mk_Event`Date(5,1,2019), "Ir as compras e
     aproveitar para tomar um pequeno-almoco reforcado ou antecipar a
    hora do almoco e a proposta do nosso Brunch, servido entre as 11h
    e as 16h.", 7.8, "Cascais", 30);
14   event5: Event := new Event("Porto VS Belenenses", "Desporto",
    mk_Event`Date(30,1,2019), mk_Event`Date(30,1,2019), "Lorem ipsum."
    , 35, "Porto", 50000);
15   event6: Event := new Event("Leixoes VS Famalicao", "Desporto",
    mk_Event`Date(7,4,2020), mk_Event`Date(7,4,2020), "Lorem ipsum.",
    12.5, "Matosinhos", 2);
16
17   proposed1: Event := new Event("Workshop Comida Saudavel daTerra",
    "Gastronomia", mk_Event`Date(15,7,2019), mk_Event`Date(15,7,2019),
     "Workshop de comida saudavel, daTerra baixa, 15h.", 5, "Porto",
    20);
18   proposed2: Event := new Event("Cozinhar Nunca Foi Facil", "
    Gastronomia", mk_Event`Date(20,12,2019), mk_Event`Date(20,12,2019)
    , "Lorem.", 10, "Lisboa", 35);
19
20 operations
21
22   public AgendaTest: () ==> AgendaTest
23   AgendaTest() == (
24     return self;
25   );
26
27   -- Creates agenda and verifies the parameters
28   private testCreateAgenda: () ==> ()
29   testCreateAgenda() == (
30     dcl a: Agenda := new Agenda();
31     assertTrue(a.categories = {"Concertos", "Exposicoes", "
    Gastronomia", "Moda", "Desporto", "Natureza"});
32     assertTrue(a.locations = {"Porto" |-> {"Porto", "Matosinhos", "
    Maia", "Vila Nova de Gaia"}, "Lisboa" |-> {"Lisboa", "Amadora", "
```

```
    Cascais", "Sintra"}, "Faro" |-> {"Faro", "Albufeira", "Portimao"
    }});
33      assertTrue(a.loggedInUser = nil);
34    );
35
36    private testAddUser: () ==> ()
37    testAddUser() == (
38      agenda.addUser(userAdmin);
39      agenda.addUser(user1);
40      agenda.addUser(user2);
41      agenda.addUser(user3);
42      agenda.addUser(user4);
43      assertTrue(agenda.users <> {|->});
44    );
45
46    private testLoginAdmin: () ==> ()
47    testLoginAdmin() == (
48      dcl outcome: bool;
49      agenda.loggedInUser := nil;
50      -- fail
51      outcome:= agenda.login("julieta@gmail.com", "person12345");
52      assertTrue(outcome = false);
53      assertTrue(agenda.loggedInUser = nil);
54
55      -- success
56      outcome := agenda.login("julieta@gmail.com", "julieta12345");
57      assertTrue(outcome = true);
58      assertTrue(agenda.loggedInUser <> nil);
59      assertTrue(agenda.loggedInUser = userAdmin);
60    );
61
62    private testAddEvent: () ==> ()
63    testAddEvent() == (
64      agenda.addEvent(event1);
65      agenda.addEvent(event2);
66      agenda.addEvent(event3);
67      agenda.addEvent(event4);
68      agenda.addEvent(event5);
69      agenda.addEvent(event6);
70      assertTrue(agenda.events <> {});
71      assertTrue(agenda.events = {event1, event2, event3, event4,
    event5, event6});
72    );
73
74    private testFindByCity: () ==> ()
75    testFindByCity() == (
76      dcl cityEvents : set of Event := {};
77      --fail
78      cityEvents := agenda.findByCity("Faro");
```

```
79        assertTrue(cityEvents = {});
80
81        --success
82        cityEvents := agenda.findByCity("Porto");
83        assertTrue(cityEvents <>{});
84      );
85
86      private testFindByDistrict: () ==> ()
87      testFindByDistrict() == (
88        dcl districtEvents : set of Event := {};
89        --fail
90        districtEvents := agenda.findByDistrict("Faro");
91        assertTrue(districtEvents = {});
92
93        districtEvents := agenda.findByDistrict("Braganca");
94        assertTrue(districtEvents = {});
95
96        --success
97        districtEvents := agenda.findByDistrict("Lisboa");
98        assertTrue(districtEvents <> {});
99      );
100
101     private testFindByCategory: () ==> ()
102     testFindByCategory() == (
103       dcl categoryEvents : set of Event := {};
104       --fail
105       categoryEvents := agenda.findByCategory("Moda");
106       assertTrue(categoryEvents = {});
107
108       --success
109       categoryEvents := agenda.findByCategory("Desporto");
110       assertTrue(categoryEvents <>{});
111     );
112
113     private testFindByDate: () ==> ()
114     testFindByDate() == (
115       dcl dateEvents : set of Event := {};
116       --fail
117       dateEvents := agenda.findByDate(mk_Event`Date(31,1,2018));
118       assertTrue(dateEvents = {});
119
120       --success
121       dateEvents := agenda.findByDate(mk_Event`Date(5,1,2019));
122       assertTrue(dateEvents <>{});
123
124     );
125
126     private testFindEvents: () ==> ()
127     testFindEvents() == (
```

```
128       dcl eventsFound : set of Event := {};
129       --fail
130       eventsFound := agenda.findEvents ("Matosinhos", "", "Moda",
      mk_Event`Date(7,4,2020));
131       assertTrue(eventsFound = {});
132
133       --success
134       eventsFound := agenda.findEvents("Matosinhos", "", "", nil);
135       assertTrue(eventsFound <>{});
136
137       eventsFound := agenda.findEvents("", "Porto", "", nil);
138       assertTrue(eventsFound <>{});
139
140       eventsFound := agenda.findEvents("", "", "Desporto", nil);
141       assertTrue(eventsFound <>{});
142
143       eventsFound := agenda.findEvents("", "", "", mk_Event`Date
      (7,4,2020));
144       assertTrue(eventsFound <>{});
145
146       eventsFound := agenda.findEvents("Matosinhos", "", "Desporto",
      mk_Event`Date(7,4,2020));
147       assertTrue(eventsFound <>{});
148
149     );
150
151     private testLoginRegular: () ==> ()
152     testLoginRegular() == (
153       dcl outcome: bool;
154       agenda.loggedInUser := nil;
155       -- fail
156       outcome := agenda.login("sofia@gmail.com", "person12345");
157       assertTrue(outcome = false);
158       assertTrue(agenda.loggedInUser = nil);
159
160       -- success
161       outcome := agenda.login("sofia@gmail.com", "sofia12345");
162       assertTrue(outcome = true);
163       assertTrue(agenda.loggedInUser <> nil);
164       assertTrue(agenda.loggedInUser = user1);
165     );
166
167     private testBuyTickets: () ==> ()
168     testBuyTickets() == (
169       --fail
170       dcl outcome: bool := agenda.buyTicket(1, 20001);
171       assertTrue(outcome = false);
172       assertTrue(event1.getSoldTickets() = 0);
173       assertTrue(agenda.loggedInUser.getTicketsBought() = 30);
```

```
174
175       --success
176       outcome := agenda.buyTicket(6, 2);
177       assertTrue(outcome = true);
178       assertTrue(event6.getSoldTickets() = 2);
179       assertTrue(agenda.loggedInUser.getTicketsBought() = 32);
180     );
181
182     private testExistsEvent: () ==> ()
183     testExistsEvent() == (
184       --fail
185       dcl outcome: bool := agenda.existsEvent(event1);
186       assertTrue(outcome = true);
187
188       --success
189       outcome := agenda.existsEvent(proposed2);
190       assertTrue(outcome = false);
191     );
192
193     private testProposeEvent: () ==> ()
194     testProposeEvent() == (
195       agenda.proposeEvent(proposed1);
196       agenda.proposeEvent(proposed2);
197       assertTrue(agenda.proposedEvents <> {});
198       assertTrue(agenda.proposedEvents = {proposed1, proposed2});
199     );
200
201     private testRejectProposedEvent: () ==> ()
202     testRejectProposedEvent() == (
203       agenda.rejectProposedEvent(proposed2);
204       assertTrue(agenda.proposedEvents = {proposed1});
205     );
206
207     private testAcceptProposedEvent: () ==> ()
208     testAcceptProposedEvent() == (
209       agenda.acceptProposedEvent(proposed1);
210       assertTrue(agenda.proposedEvents = {});
211       assertTrue(agenda.events = {event1, event2, event3, event4,
    event5, event6, proposed1});
212     );
213
214     private testMostPopularEvent: () ==> ()
215     testMostPopularEvent() == (
216       dcl outcome: Event := agenda.mostPopularEvent();
217       assertTrue(outcome = event6);
218     );
219
220     private testMostProfitableEvent: () ==> ()
221     testMostProfitableEvent() == (
```

```
222       dcl outcome: Event := agenda.mostProfitableEvent();
223       assertTrue(outcome = event6);
224     );
225
226     private testMostActiveUser: () ==> ()
227     testMostActiveUser() == (
228       dcl outcome: User := agenda.mostActiveUser();
229       assertTrue(outcome = user4);
230     );
231
232     public test: () ==> ()
233     test() == (
234       testCreateAgenda();
235       testAddUser();
236       testLoginAdmin();
237       testAddEvent();
238       testFindByCity();
239       testFindByDistrict();
240       testFindByCategory();
241       testFindByDate();
242       testFindEvents();
243       testLoginRegular();
244       testBuyTickets();
245       testExistsEvent();
246       testProposeEvent();
247       testLoginAdmin();
248       testRejectProposedEvent();
249       testAcceptProposedEvent();
250       testMostPopularEvent();
251       testMostProfitableEvent();
252       testMostActiveUser();
253     );
254
255 end AgendaTest
```

4.1.3 Classe EventTest

```
1 class EventTest is subclass of Tests
2
3   operations
4     public EventTest: () ==> EventTest
5     EventTest() == (
6       return self;
7     );
8
9     -- Creates an event, verifies the parameters and simulates a
    purchase
```

```
10      private testCreateEvent: () ==> ()
11      testCreateEvent() == (
12        dcl event: Event := new Event("Twenty One Pilots", "Concertos",
      mk_Event`Date(17,3,2019), mk_Event`Date(17,3,2019), "Lorem ipsum
      dolor sit amet.", 42, "Lisboa", 100);
13        assertTrue(event.getTitle() = "Twenty One Pilots");
14        assertTrue(event.getCategory() = "Concertos");
15        assertTrue(event.getState() = <Available>);
16        assertTrue(event.getDateStart() = mk_Event`Date(17,3,2019));
17        assertTrue(event.getDateEnd() = mk_Event`Date(17,3,2019));
18        assertTrue(event.getDescription() = "Lorem ipsum dolor sit amet.
      ");
19        assertTrue(event.getPrice() = 42);
20        assertTrue(event.getCity() = "Lisboa");
21        assertTrue(event.getTotalTickets() = 100);
22        assertTrue(event.getSoldTickets() = 0);
23
24        event.buy(100);
25        assertTrue(event.getSoldTickets() = 100);
26        assertTrue(event.getState() = <SoldOut>);
27        assertTrue(event.getStats() = 100);
28        assertTrue(event.getProfit() = 4200);
29      );
30
31      -- Creates an event, leap year
32      private testCreateEventLeap: () ==> ()
33      testCreateEventLeap() == (
34        dcl event: Event := new Event("Twenty One Pilots", "Concertos",
      mk_Event`Date(29,2,2020), mk_Event`Date(29,2,2020), "Lorem ipsum
      dolor sit amet.", 42, "Lisboa", 100);
35        assertTrue(event.getDateStart() = mk_Event`Date(29,2,2020));
36        assertTrue(event.getDateEnd() = mk_Event`Date(29,2,2020));
37      );
38
39      public test: () ==> ()
40      test() == (
41        testCreateEvent();
42        testCreateEventLeap();
43      );
44
45  end EventTest
```

4.1.4 Classe UserTest

```
class UserTest is subclass of Tests

  operations

    public UserTest: () ==> UserTest
    UserTest() == (
      return self;
    );

    -- Creates a user: admin
    private testCreateAdmin: () ==> ()
    testCreateAdmin() == (
      dcl user: User := new Admin("julieta@gmail.com", "julieta12345")
    ;
      assertTrue(user.getEmail() = "julieta@gmail.com");
      assertTrue(user.isAdmin() = true);
      assertTrue(user.checkLogin("julieta@gmail.com", "julieta12345")
    = true);
    );

    -- Creates a user: regular and tests buy tickets
    private testCreateRegular: () ==> ()
    testCreateRegular() == (
      dcl user: User := new Regular("sofia@gmail.com", "sofia12345", "
    Sofia", "Silva", 500, 30);
      assertTrue(user.getEmail() = "sofia@gmail.com");
      assertTrue(user.isAdmin() = false);
      assertTrue(user.checkLogin("sofia@gmail.com", "sofia12345") =
    true);
      assertTrue(user.getTicketsBought() = 30);

      user.buyTickets(2, 50);
      assertTrue(user.getBalance() = 400);
      assertTrue(user.getTicketsBought() = 32);
    );

    public test: () ==> ()
    test() == (
      testCreateAdmin();
      testCreateRegular();
    );

end UserTest
```

4.2 Coverage

4.2.1 Classe Agenda

Function or operation	Line	Coverage	Calls
Agenda	32	100.0%	2
acceptProposedEvent	79	100.0%	1
addEvent	67	100.0%	6
addUser	43	100.0%	5
buyTicket	154	100.0%	1
existsCategory	311	100.0%	26
existsCity	283	100.0%	39
existsCityInDistrict	303	100.0%	54
existsDistrict	295	100.0%	31
existsEvent	269	100.0%	1
findByCategory	241	100.0%	5
findByCity	210	100.0%	5
findByDate	254	100.0%	10
findByDistrict	223	100.0%	4
findEvents	176	100.0%	6
login	49	100.0%	3
mostActiveUser	124	100.0%	3
mostPopularEvent	88	100.0%	6
mostProfitableEvent	106	100.0%	1
proposeEvent	148	100.0%	2
rejectProposedEvent	73	100.0%	1
wantedDate	316	100.0%	36
Agenda.vdmpp		100.0%	248

4.2.2 Classe Event

Function or operation	Line	Coverage	Calls
Event	63	100.0%	10
buy	137	100.0%	2
getCategory	93	100.0%	50
getCity	117	100.0%	75
getDateEnd	105	100.0%	40
getDateStart	101	100.0%	40
getDescription	109	100.0%	1
getID	85	100.0%	12
getPrice	113	100.0%	3
getProfit	133	100.0%	14
getSoldTickets	125	100.0%	6
getState	97	100.0%	2
getStats	129	100.0%	14
getTitle	89	100.0%	69
getTotalTickets	121	100.0%	3
validateDates	148	100.0%	10
Event.vdmpp		100.0%	351

4.2.3 Classe User

Function or operation	Line	Coverage	Calls
UserInit	18	100.0%	7
buyTickets	48	100.0%	2
checkLogin	35	100.0%	8
getBalance	44	100.0%	2
getEmail	27	100.0%	17
getTicketsBought	40	100.0%	2
isAdmin	31	100.0%	2
User.vdmpp		100.0%	40

4.2.4 Classe Regular

Function or operation	Line	Coverage	Calls
Regular	17	100.0%	5
buyTickets	40	100.0%	2
getBalance	36	100.0%	2
getTicketsBought	32	100.0%	11
isAdmin	28	100.0%	5
Regular.vdmpp		100.0%	25

4.2.5 Classe Admin

Function or operation	Line	Coverage	Calls
Admin	6	100.0%	2
buyTickets	25	0.0%	0
getBalance	21	0.0%	0
getTicketsBought	17	0.0%	0
isAdmin	13	100.0%	12
Admin.vdmpp		100.0%	14

Nota: Algumas destas operações têm uma *coverage* de 0% devido à utilização de *is not yet specified*. Tratam-se se operações que um utilizador do tipo *Admin* não deveria ter, mas é obrigatória a sua presença por uma questão de herança de classes.

5 Verificação do Modelo

5.1 Exemplo de Verificação de um Domínio

5.1.1 Pré-condição

```
class Agenda
-- Returns if city exists in a certain district
    public pure existsCityInDistrict: String * String ==> bool
    existsCityInDistrict(city, district) == (
      dcl districtCities : set of String := locations(district);
      return city in set districtCities;
    )
    pre existsDistrict(district) and locations <> {|->};
```

A pré-condição em causa é *existsDistric* que requer a existência do distrito no sistema antes de se verificar se a cidade pertence a esse mesmo distrito.

5.1.2 Proof Obligation gerada pelo Overture

No.	Nome da Proof Obligation	Tipo
20	Agenda‘existsCityInDistrict(String, String), districCities	Legal Map Application

```
(forall city:Agenda‘String, district:Agenda‘String & ((existsDistrict(
   district) and (locations <> {|->})) => (district in set (dom
   locations))))
```

Com a expressão *(district in set (dom locations))*, a ferramenta Overture verifica se o distrito faz parte do set de keys do map *locations*, ou seja, verifica se o distrito existe no sistema.

5.1.3 Proof Sketch

```
class Agenda
  public findByDistrict: String ==> set of Event
    findByDistrict(district) == (
      dcl districtEvents : set of Event := {};
      if existsDistrict(district)
      then (
```

```
7         for all event in set events do (
8           if existsCityInDistrict(event.getCity(), district)
9           then districtEvents := districtEvents union {event}
10        );
11        return districtEvents;
12      )
13      else return {}
14    )
15    pre loggedInUser <> nil and events <> {}
16    post forall e in set RESULT & existsCityInDistrict(e.getCity(),
    district);
```

Antes da chamada à função *existsCityInDistrict* na linha 8, é feita a verificação *existsDistrict* na linha 5, deste modo a pré-condição dentro da respetiva função é sempre preservada.

5.2 Exemplo de Verificação de uma Invariante

5.2.1 Invariante

```
1 class Event
2   -- Ensures inexistence of overbooking
3   inv totalTickets >= soldTickets;
```

O invariante em causa evita o overbooking de eventos, evitando que o número de bilhetes vendidos nunca seja maior do que o número de bilhetes disponíveis.

5.2.2 Proof Obligation gerada pelo Overture

No.	Nome da Proof Obligation	Tipo
40	Event‘buy(nat)	State Invariant Holds

```
1 (forall nTickets:nat & (((totalTickets >= (soldTickets + nTickets))
    and (state = <Available>)) => ((totalTickets >= soldTickets) => (
    totalTickets >= (soldTickets + nTickets)))))
```

A ferramenta Overture verifica o invariante na função *buy*, visto que é nesta que a veracidade do invariante pode ser posta em causa.

5.2.3 Proof Sketch

```
1  class Agenda
2  -- Returns if the purchase was successful
3      public buyTicket: nat * nat ==> bool
4      buyTicket(eventID, nTickets) == (
5        dcl eventBought : Event;
6        for all event in set events do (
7            if event.getID() = eventID
8            then eventBought := event;
9        );
10       -- verifies if there are enough tickets to sell
11       if eventBought.getTotalTickets() >= eventBought.getSoldTickets()
     + nTickets and
12           loggedInUser.getBalance() >= nTickets * eventBought.getPrice
    ()
13       then (
14         loggedInUser.buyTickets(nTickets, eventBought.getPrice());
15         eventBought.buy(nTickets);
16         return true;
17       )
18       else return false
19     )
20     pre loggedInUser <> nil and not loggedInUser.isAdmin();
```

Antes da chamada à função crítica *buy* na linha 15, é feita a verificação do invariante na linha 11 com o objetivo de o preservar sempre.

6 Geração de Código

O código Java foi gerado com sucesso a partir do modelo VDM++.

6.1 Main

```java
package AgendaViral;

public class Main {

  public static void main(String[] args) {
    Agenda agenda = new Agenda();
    Interface gui = new Interface(agenda);
    gui.loginMenu();
  }
}
```

6.2 Interface

```java
package AgendaViral;

import java.util.Iterator;
import java.util.Scanner;

import org.overture.codegen.runtime.SetUtil;
import org.overture.codegen.runtime.VDMSet;

public class Interface {
  private Agenda agenda;
  Scanner scanner = new Scanner(System.in);

  public Interface(Agenda agenda) {
    this.agenda = agenda;

    Admin userAdmin = new Admin("julieta@gmail.com", "julieta12345");
    Regular user1 = new Regular("sofia@gmail.com", "sofia12345", "
    Sofia", "Silva", 500L, 30L);
    Regular user2 = new Regular("bibi@gmail.com", "bibi12345", "
    Beatriz", "Baldaia", 200L, 5L);
    Regular user3 = new Regular("carlos@gmail.com", "carlos12345", "
    Carlos", "Freitas", 800L, 10L);
    Regular user4 = new Regular("vicente@gmail.com", "vicente12345", "
    Vicente", "Espinha", 100L, 50L);
```

```
21      Event event1 = new Event("Twenty One Pilots", "Concertos", new
        Event.Date(17L, 3L, 2019L),
22          new Event.Date(17L, 3L, 2019L), "The Bandito Tour, dia 17 de
        Marco, na Altice Arena.", 42L, "Lisboa",
23          20000L);
24      Event event2 = new Event("EXO", "Concertos", new Event.Date(20L, 5
        L, 2019L), new Event.Date(20L, 5L, 2019L),
25          "Estreia em Portugal dia 20 de Maio, na Altice Arena.", 30L, "
        Lisboa", 20000L);
26      Event event3 = new Event("Aberturas: Tom Emerson em conversa com o
         arquivo Alvaro Siza", "Exposicoes",
27          new Event.Date(6L, 1L, 2019L), new Event.Date(6L, 2L, 2019L),
28          "Visita orientada a exposicao por Matilde Seabra, educadora.
        Localizacao: Biblioteca de Serralves", 2.5,
29          "Porto", 200L);
30      Event event4 = new Event("Brunch Mercearia Bio", "Gastronomia",
        new Event.Date(5L, 1L, 2019L),
31          new Event.Date(5L, 1L, 2019L),
32          "Ir as compras e aproveitar para tomar um pequeno-almoco
        reforcado ou antecipar a hora do almoco e a proposta do nosso
        Brunch, servido entre as 11h e as 16h.",
33          7.8, "Cascais", 30L);
34      Event event5 = new Event("Porto VS Belenenses", "Desporto", new
        Event.Date(30L, 1L, 2019L),
35          new Event.Date(30L, 1L, 2019L), "Lorem ipsum.", 35L, "Porto",
        50000L);
36      Event event6 = new Event("Leixoes VS Famalicao", "Desporto", new
        Event.Date(7L, 4L, 2020L),
37          new Event.Date(7L, 4L, 2020L), "Lorem ipsum.", 12.5, "
        Matosinhos", 2000L);
38      Event proposed1 = new Event("Workshop Comida Saudavel daTerra", "
        Gastronomia", new Event.Date(15L, 7L, 2019L),
39          new Event.Date(15L, 7L, 2019L), "Workshop de comida saudavel,
        daTerra baixa, 15h.", 5L, "Porto", 20L);
40      Event proposed2 = new Event("Cozinhar Nunca Foi Facil", "
        Gastronomia", new Event.Date(20L, 12L, 2019L),
41          new Event.Date(20L, 12L, 2019L), "Lorem.", 10L, "Lisboa", 35L)
        ;
42
43      agenda.addUser(userAdmin);
44      agenda.addUser(user1);
45      agenda.addUser(user2);
46      agenda.addUser(user3);
47      agenda.addUser(user4);
48
49      agenda.login("julieta@gmail.com", "julieta12345");
50      agenda.addEvent(event1);
51      agenda.addEvent(event2);
52      agenda.addEvent(event3);
```

```
53      agenda.addEvent(event4);
54      agenda.addEvent(event5);
55      agenda.addEvent(event6);
56
57      agenda.login("carlos@gmail.com", "carlos12345");
58      agenda.buyTicket(2, 5);
59
60      agenda.login("sofia@gmail.com", "sofia12345");
61      agenda.proposeEvent(proposed1);
62      agenda.proposeEvent(proposed2);
63      agenda.buyTicket(1, 5);
64
65      loginMenu();
66    }
67
68    public void loginMenu() {
69      System.out.println(" ---------------------------------------- "
        );
70      System.out.println("|                  Login                 |"
        );
71      System.out.println(" ---------------------------------------- "
        );
72
73      System.out.print(" > Email: ");
74      String email = scanner.nextLine();
75
76      System.out.print(" > Password: ");
77      String password = scanner.nextLine();
78
79      System.out.println("");
80
81      if (agenda.login(email, password))
82        mainMenu();
83      else
84        System.out.println(" > Error: login failed");
85    }
86
87    public void mainMenu() {
88      if (agenda.loggedInUser.isAdmin())
89        mainMenuAdmin();
90      else
91        mainMenuRegular();
92    }
93
94    /*
95     * ADMIN
96     */
97
98    public void mainMenuAdmin() {
```

```
99      System.out.println(" ------------------------------------------ "
        );
100     System.out.println("|                AGENDA VIRAL               |"
        );
101     System.out.println(" ------------------------------------------ "
        );
102     System.out.println(" 1.  Add Event                    0.  Logout")
        ;
103     System.out.println(" 2.  Proposed Events");
104     System.out.println(" 3.  Find by District");
105     System.out.println(" 4.  Find by City");
106     System.out.println(" 5.  Find by Category");
107     System.out.println(" 6.  Find by Date");
108     System.out.println(" 7.  Find by Multiple Filters");
109     System.out.println(" 8.  Most Popular Event");
110     System.out.println(" 9.  Most Profitable Event");
111     System.out.println(" 10. Most Active User");
112     System.out.println(" ------------------------------------------ "
        );
113
114     System.out.print(" > Option: ");
115     int option = scanner.nextInt();
116     scanner.nextLine();
117
118     System.out.println("");
119
120     switch (option) {
121     case 1:
122       addEventMenu();
123       break;
124     case 2:
125       proposedEventsMenu();
126       break;
127     case 3:
128       findByDistrictMenu();
129       break;
130     case 4:
131       findByCityMenu();
132       break;
133     case 5:
134       findByCategoryMenu();
135       break;
136     case 6:
137       findByDateMenu();
138       break;
139     case 7:
140       findByMultipleFiltersMenu();
141       break;
142     case 8:
```

```
143       mostPopularMenu();
144       break;
145     case 9:
146       mostProfitableMenu();
147       break;
148     case 10:
149       mostActiveMenu();
150       break;
151     case 0:
152       loginMenu();
153       break;
154     default:
155       loginMenu();
156       break;
157     }
158   }
159
160   public void addEventMenu() {
161     System.out.println(" ------------------------------------------ "
      );
162     System.out.println("|                 Add Event                 |"
      );
163     System.out.println(" ------------------------------------------ "
      );
164
165     System.out.print(" > Title: ");
166     String title = scanner.nextLine();
167
168     System.out.print(" > Category: ");
169     String category = scanner.nextLine();
170
171     System.out.print(" > Starting Date[dd/mm/yy]: ");
172     String startDate = scanner.nextLine();
173     String[] part1 = startDate.split("/");
174
175     int day1 = Integer.parseInt(part1[0]);
176     int month1 = Integer.parseInt(part1[1]);
177     int year1 = Integer.parseInt(part1[2]);
178
179     System.out.print(" > Ending Date[dd/mm/yy]: ");
180     String endDate = scanner.nextLine();
181     String[] part2 = endDate.split("/");
182
183     int day2 = Integer.parseInt(part2[0]);
184     int month2 = Integer.parseInt(part2[1]);
185     int year2 = Integer.parseInt(part2[2]);
186
187     System.out.print(" > Description: ");
188     String description = scanner.nextLine();
```

```
189
190     System.out.print(" > Price: ");
191     int price = scanner.nextInt();
192     scanner.nextLine();
193
194     System.out.print(" > City: ");
195     String city = scanner.nextLine();
196
197     System.out.print(" > Total Tickets: ");
198     int tickets = scanner.nextInt();
199     scanner.nextLine();
200
201     Event event = new Event(title, category, new Event.Date(day1,
    month1, year1),
202         new Event.Date(day2, month2, year2), description, price, city,
     tickets);
203
204     agenda.addEvent(event);
205
206     mainMenuAdmin();
207   }
208
209   public void proposedEventsMenu() {
210     System.out.println(" ---------------------------------------- "
    );
211     System.out.println("|              Proposed Events            |"
    );
212     System.out.println(" ---------------------------------------- "
    );
213     for (Iterator iter = agenda.proposedEvents.iterator(); iter.
    hasNext();) {
214       Event event = (Event) iter.next();
215       System.out.println(" Id: " + event.getID());
216       System.out.println(" Title: " + event.getTitle());
217
218       if (iter.hasNext())
219         System.out.println("");
220     }
221
222     if (agenda.proposedEvents.isEmpty())
223       System.out.println("              No proposed events
      ");
224
225     System.out.println(" ---------------------------------------- "
    );
226     System.out.println("                                  0. Return "
    );
227     System.out.println(" ---------------------------------------- "
    );
```

```
228
229     System.out.print(" > Event Id: ");
230     int option = scanner.nextInt();
231     scanner.nextLine();
232
233     System.out.println("");
234
235     if (option == 0) {
236       mainMenu();
237     }
238
239     for (Iterator iter = agenda.proposedEvents.iterator(); iter.
    hasNext();) {
240       Event event = (Event) iter.next();
241
242       if (option == event.getID().intValue()) {
243         proposedEventMenu(event);
244       }
245     }
246
247     mainMenuAdmin();
248   }
249
250   public void proposedEventMenu(Event event) {
251     System.out.println(" ------------------------------------------ "
    );
252     System.out.println("|               Proposed Event             |")
    ;
253     System.out.println(" ------------------------------------------ "
    );
254
255     System.out.println(" Id: " + event.getID());
256     System.out.println(" Title: " + event.getTitle());
257     System.out.println(" Category: " + event.getCategory());
258     System.out.println(" City: " + event.getCity());
259     System.out.println(" Date: from " + event.getDateStart().day + "/"
    + event.getDateStart().month + "/"
260         + event.getDateStart().year + " to " + event.getDateEnd().day
    + "/" + event.getDateEnd().month + "/"
261         + event.getDateEnd().year);
262     System.out.println(" Price: " + event.getPrice() + " euros");
263     System.out.println(" Total Tickets: " + event.getTotalTickets() +
    " | Sold Tickets: " + event.getSoldTickets());
264     System.out.println(" Descriprion: " + event.getDescription());
265
266     System.out.println(" ------------------------------------------ "
    );
267
268     System.out.print(" > Accept or Reject (A/R): ");
```

```
269     String action = scanner.nextLine();
270
271     System.out.println("");
272
273     if (action.equals("A"))
274       agenda.acceptProposedEvent(event);
275     else if (action.equals("R"))
276       agenda.rejectProposedEvent(event);
277
278     mainMenuAdmin();
279   }
280
281   public void mostPopularMenu() {
282     Event event = agenda.mostPopularEvent();
283
284     System.out.println(" ------------------------------------------ "
      );
285     System.out.println("|             Most Popular Event            |"
      );
286     System.out.println(" ------------------------------------------ "
      );
287
288     System.out.println(" Id: " + event.getID());
289     System.out.println(" Title: " + event.getTitle());
290     System.out.println(" Category: " + event.getCategory());
291     System.out.println(" City: " + event.getCity());
292     System.out.println(" Date: from " + event.getDateStart().day + "/"
      + event.getDateStart().month + "/"
293         + event.getDateStart().year + " to " + event.getDateEnd().day
     + "/" + event.getDateEnd().month + "/"
294         + event.getDateEnd().year);
295     System.out.println(" Price: " + event.getPrice() + " euros");
296     System.out.println(" Total Tickets: " + event.getTotalTickets() +
     " | Sold Tickets: " + event.getSoldTickets());
297     System.out.println(" Descriprion: " + event.getDescription());
298
299     System.out.println(" ------------------------------------------ "
      );
300
301     System.out.println(" > Press Enter to continue...");
302     try {
303       System.in.read();
304     } catch (Exception e) {
305     }
306     mainMenuAdmin();
307   }
308
309   public void mostProfitableMenu() {
310     Event event = agenda.mostProfitableEvent();
```

```
311
312     System.out.println(" ---------------------------------------- ");
313     System.out.println("|           Most Profitable Event          |");
314     System.out.println(" ---------------------------------------- ");
315
316     System.out.println(" Id: " + event.getID());
317     System.out.println(" Title: " + event.getTitle());
318     System.out.println(" Category: " + event.getCategory());
319     System.out.println(" City: " + event.getCity());
320     System.out.println(" Date: from " + event.getDateStart().day + "/"
        + event.getDateStart().month + "/"
321         + event.getDateStart().year + " to " + event.getDateEnd().day
      + "/" + event.getDateEnd().month + "/"
322         + event.getDateEnd().year);
323     System.out.println(" Price: " + event.getPrice() + " euros");
324     System.out.println(" Total Tickets: " + event.getTotalTickets() +
      " | Sold Tickets: " + event.getSoldTickets());
325     System.out.println(" Descriprion: " + event.getDescription());
326
327     System.out.println(" ---------------------------------------- ");
328
329     System.out.println(" > Press Enter to continue...");
330     try {
331       System.in.read();
332     } catch (Exception e) {
333     }
334     mainMenuAdmin();
335   }
336
337   public void mostActiveMenu() {
338     User user = agenda.mostActiveUser();
339
340     System.out.println(" ---------------------------------------- ");
341     System.out.println("|              Most Active User            |");
342     System.out.println(" ---------------------------------------- ");
343
344     System.out.println(" Email: " + user.getEmail());
345     System.out.println(" No. Bought Tickets: " + user.getTicketsBought
      ());
346
347     System.out.println(" ---------------------------------------- ");
```

```
348
349     System.out.println(" > Press Enter to continue...");
350     try {
351       System.in.read();
352     } catch (Exception e) {
353     }
354     mainMenuAdmin();
355   }
356
357   /*
358    * REGULAR USER
359    */
360
361   public void mainMenuRegular() {
362     System.out.println(" ------------------------------------------ "
      );
363     System.out.println("|                AGENDA VIRAL               |"
      );
364     System.out.println(" ------------------------------------------ "
      );
365     System.out.println(" 1.  Propose Event                0.  Logout")
      ;
366     System.out.println(" 2.  Find by District");
367     System.out.println(" 3.  Find by City");
368     System.out.println(" 4.  Find by Category");
369     System.out.println(" 5.  Find by Date");
370     System.out.println(" 6.  Find by Multiple Filters");
371     System.out.println(" ------------------------------------------ "
      );
372     System.out.println(" ! Balance: " + agenda.loggedInUser.getBalance
      () + " euros");
373
374     System.out.print(" > Option: ");
375     int option = scanner.nextInt();
376     scanner.nextLine();
377
378     System.out.println("");
379
380     switch (option) {
381     case 1:
382       proposeEventMenu();
383       break;
384     case 2:
385       findByDistrictMenu();
386       break;
387     case 3:
388       findByCityMenu();
389       break;
390     case 4:
```

```
391         findByCategoryMenu();
392         break;
393       case 5:
394         findByDateMenu();
395         break;
396       case 6:
397         findByMultipleFiltersMenu();
398         break;
399       case 0:
400         loginMenu();
401         break;
402       default:
403         loginMenu();
404         break;
405       }
406     }
407
408     public void proposeEventMenu() {
409       System.out.println(" ------------------------------------------ "
          );
410       System.out.println("|               Propose Event               |"
          );
411       System.out.println(" ------------------------------------------ "
          );
412
413       System.out.print(" > Title: ");
414       String title = scanner.nextLine();
415
416       System.out.print(" > Category: ");
417       String category = scanner.nextLine();
418
419       System.out.print(" > Starting Date[dd/mm/yy]: ");
420       String startDate = scanner.nextLine();
421       String[] part1 = startDate.split("/");
422
423       int day1 = Integer.parseInt(part1[0]);
424       int month1 = Integer.parseInt(part1[1]);
425       int year1 = Integer.parseInt(part1[2]);
426
427       System.out.print(" > Ending Date[dd/mm/yy]: ");
428       String endDate = scanner.nextLine();
429       String[] part2 = endDate.split("/");
430
431       int day2 = Integer.parseInt(part2[0]);
432       int month2 = Integer.parseInt(part2[1]);
433       int year2 = Integer.parseInt(part2[2]);
434
435       System.out.print(" > Description: ");
436       String description = scanner.nextLine();
```

```
437
438     System.out.print(" > Price: ");
439     int price = scanner.nextInt();
440     scanner.nextLine();
441
442     System.out.print(" > City: ");
443     String city = scanner.nextLine();
444
445     System.out.print(" > Total Tickets: ");
446     int tickets = scanner.nextInt();
447     scanner.nextLine();
448
449     Event event = new Event(title, category, new Event.Date(day1,
    month1, year1),
450         new Event.Date(day2, month2, year2), description, price, city,
    tickets);
451
452     agenda.proposeEvent(event);
453
454     mainMenuRegular();
455   }
456
457   public void findByDistrictMenu() {
458     System.out.println(" ------------------------------------------ "
    );
459     System.out.println("|              Find By District             |"
    );
460     System.out.println(" ------------------------------------------ "
    );
461     System.out.println(" 1.  Porto                          0. Return")
    ;
462     System.out.println(" 2.  Lisboa");
463     System.out.println(" 3.  Faro");
464     System.out.println(" ------------------------------------------ "
    );
465
466     System.out.print(" > District: ");
467     int option = scanner.nextInt();
468     scanner.nextLine();
469
470     System.out.println("");
471
472     VDMSet events = SetUtil.set();
473
474     switch (option) {
475     case 1:
476       events = agenda.findByDistrict("Porto");
477       break;
478     case 2:
```

```
479       events = agenda.findByDistrict("Lisboa");
480       break;
481     case 3:
482       events = agenda.findByDistrict("Faro");
483       break;
484     case 0:
485       mainMenu();
486       break;
487     default:
488       mainMenu();
489       break;
490     }
491
492     foundEventsMenu(events);
493   }
494
495   public void findByCityMenu() {
496     System.out.println(" ------------------------------------------ "
      );
497     System.out.println("|                Find By City               |"
      );
498     System.out.println(" ------------------------------------------ "
      );
499     System.out.println(" - PORTO                            0. Return")
      ;
500     System.out.println(" 1.  Porto");
501     System.out.println(" 2.  Matosinhos");
502     System.out.println(" 3.  Maia");
503     System.out.println(" 4.  Vila Nova de Gaia");
504     System.out.println(" - LISBOA");
505     System.out.println(" 5.  Lisboa");
506     System.out.println(" 6.  Amadora");
507     System.out.println(" 7.  Cascais");
508     System.out.println(" 8.  Sintra");
509     System.out.println(" - FARO");
510     System.out.println(" 9.  Faro");
511     System.out.println(" 10. Albufeira");
512     System.out.println(" 11. Portimao");
513     System.out.println(" ------------------------------------------ "
      );
514
515     System.out.print(" > City: ");
516     int option = scanner.nextInt();
517     scanner.nextLine();
518
519     System.out.println("");
520
521     VDMSet events = SetUtil.set();
522     switch (option) {
```

```
523       case 1:
524         events = agenda.findByCity("Porto");
525         break;
526       case 2:
527         events = agenda.findByCity("Matosinhos");
528         break;
529       case 3:
530         events = agenda.findByCity("Maia");
531         break;
532       case 4:
533         events = agenda.findByCity("Vila Nova de Gaia");
534         break;
535       case 5:
536         events = agenda.findByCity("Lisboa");
537         break;
538       case 6:
539         events = agenda.findByCity("Amadora");
540         break;
541       case 7:
542         events = agenda.findByCity("Cascais");
543         break;
544       case 8:
545         events = agenda.findByCity("Sintra");
546         break;
547       case 9:
548         events = agenda.findByCity("Faro");
549         break;
550       case 10:
551         events = agenda.findByCity("Albufeira");
552         break;
553       case 11:
554         events = agenda.findByCity("Portimao");
555         break;
556       case 0:
557         mainMenu();
558         break;
559       default:
560         mainMenu();
561         break;
562       }
563
564       foundEventsMenu(events);
565     }
566
567     public void findByCategoryMenu() {
568       System.out.println(" ---------------------------------------- "
      );
569       System.out.println("|              Find By Category             |"
      );
```

```
570     System.out.println(" ---------------------------------------- "
        );
571     System.out.println(" 1.  Concertos                      0. Return")
        ;
572     System.out.println(" 2.  Exposicoes");
573     System.out.println(" 3.  Gastronomia");
574     System.out.println(" 4.  Moda");
575     System.out.println(" 5.  Desporto");
576     System.out.println(" 6.  Natureza");
577     System.out.println(" ---------------------------------------- "
        );
578
579     System.out.print(" > Category: ");
580     int option = scanner.nextInt();
581     scanner.nextLine();
582
583     System.out.println("");
584
585     VDMSet events = SetUtil.set();
586
587     switch (option) {
588     case 1:
589       events = agenda.findByCategory("Concertos");
590       break;
591     case 2:
592       events = agenda.findByCategory("Exposicoes");
593       break;
594     case 3:
595       events = agenda.findByCategory("Gastronomia");
596       break;
597     case 4:
598       events = agenda.findByCategory("Moda");
599       break;
600     case 5:
601       events = agenda.findByCategory("Desporto");
602       break;
603     case 6:
604       events = agenda.findByCategory("Natureza");
605       break;
606     case 0:
607       mainMenu();
608       break;
609     default:
610       mainMenu();
611       break;
612     }
613
614     foundEventsMenu(events);
615   }
```

```
616
617   public void findByDateMenu() {
618     System.out.println(" ------------------------------------------ "
      );
619     System.out.println("|                Find By Date               |"
      );
620     System.out.println(" ------------------------------------------ "
      );
621     System.out.println("                                  0. Main Menu")
      ;
622
623     System.out.print(" > Date[dd/mm/yy]: ");
624     String date = scanner.nextLine();
625     System.out.println("");
626
627     if (date.equals("0")) {
628       mainMenu();
629     }
630
631     String[] parts = date.split("/");
632
633     int day = Integer.parseInt(parts[0]);
634     int month = Integer.parseInt(parts[1]);
635     int year = Integer.parseInt(parts[2]);
636
637     VDMSet events = agenda.findByDate(new Event.Date(day, month, year)
      );
638
639     foundEventsMenu(events);
640   }
641
642   public void findByMultipleFiltersMenu() {
643     System.out.println(" ------------------------------------------ "
      );
644     System.out.println("|                 Find Event                |"
      );
645     System.out.println(" ------------------------------------------ "
      );
646     System.out.println(" - CITIES");
647     System.out.println(" Porto, Matosinhos, Maia, Vila Nova de Gaia");
648     System.out.println(" Lisboa, Amadora, Cascais, Sintra");
649     System.out.println(" Faro, Albufeira, Portimao");
650     System.out.println("");
651     System.out.println(" - DISTRICTS");
652     System.out.println(" Porto, Lisboa, Faro");
653     System.out.println("");
654     System.out.println(" - CATEGORIES");
655     System.out.println(" Concertos, Exposicoes, Gastronomia, Moda");
656     System.out.println(" Desporto, Natureza");
```

```
657     System.out.println(" ------------------------------------------ ");
658
659     System.out.print(" > City: ");
660     String city = scanner.nextLine();
661
662     String district = "";
663     if (city.equals("")) {
664       System.out.print(" > District: ");
665       district = scanner.nextLine();
666     }
667
668     System.out.print(" > Category: ");
669     String category = scanner.nextLine();
670
671     System.out.print(" > Date[dd/mm/yy]: ");
672     String date = scanner.nextLine();
673
674     VDMSet events = null;
675
676     if (date.equals("")) {
677       events = agenda.findEvents(city, district, category, null);
678     } else {
679       String[] parts = date.split("/");
680
681       int day = Integer.parseInt(parts[0]);
682       int month = Integer.parseInt(parts[1]);
683       int year = Integer.parseInt(parts[2]);
684       events = agenda.findEvents(city, district, category, new Event.
    Date(day, month, year));
685     }
686
687     foundEventsMenu(events);
688   }
689
690   public void foundEventsMenu(VDMSet events) {
691     System.out.println(" ------------------------------------------ ");
692     System.out.println("|                Found Events              |");
693     System.out.println(" ------------------------------------------ ");
694     for (Iterator iter = events.iterator(); iter.hasNext();) {
695       Event event = (Event) iter.next();
696       System.out.println(" Id: " + event.getID());
697       System.out.println(" Title: " + event.getTitle());
698
699       if (iter.hasNext())
700         System.out.println("");
```

```
701     }
702
703     if (events.isEmpty())
704       System.out.println("                  No events                  ");
705
706     System.out.println(" ------------------------------------------ "
        );
707     System.out.println("                                      0. Return "
        );
708     System.out.println(" ------------------------------------------ "
        );
709
710     System.out.print(" > Event Id: ");
711     int option = scanner.nextInt();
712     scanner.nextLine();
713
714     System.out.println("");
715
716     if (option == 0) {
717       mainMenu();
718     }
719
720     for (Iterator iter = events.iterator(); iter.hasNext();) {
721       Event event = (Event) iter.next();
722
723       if (option == event.getID().intValue()) {
724         foundEventMenu(event, events);
725       }
726     }
727   }
728
729   public void foundEventMenu(Event event, VDMSet events) {
730     System.out.println(" ------------------------------------------ "
      );
731     System.out.println("|                   Event                   |"
      );
732     System.out.println(" ------------------------------------------ "
      );
733
734     System.out.println(" Id: " + event.getID());
735     System.out.println(" Title: " + event.getTitle());
736     System.out.println(" Category: " + event.getCategory());
737     System.out.println(" City: " + event.getCity());
738     System.out.println(" Date: from " + event.getDateStart().day + "/"
      + event.getDateStart().month + "/"
739         + event.getDateStart().year + " to " + event.getDateEnd().day
     + "/" + event.getDateEnd().month + "/"
740         + event.getDateEnd().year);
741     System.out.println(" Price: " + event.getPrice() + " euros");
```

```
742     System.out.println(" Total Tickets: " + event.getTotalTickets() +
        " | Sold Tickets: " + event.getSoldTickets());
743     System.out.println(" Descriprion: " + event.getDescription());
744
745     System.out.println(" ------------------------------------------ "
        );
746     if (!agenda.loggedInUser.isAdmin())
747       System.out.println(" 1.  Buy Tickets                      0.
        Return");
748     else
749       System.out.println("                                      0. Return
        ");
750     System.out.println(" ------------------------------------------ "
        );
751
752     System.out.print(" > Option: ");
753     int option = scanner.nextInt();
754     scanner.nextLine();
755
756     System.out.println("");
757
758     if (option == 1) {
759       System.out.print(" > Number of Tickets: ");
760       int nTickets = scanner.nextInt();
761       scanner.nextLine();
762       boolean res = agenda.buyTicket(event.getID(), nTickets);
763       if (res)
764         System.out.println("Tickets bought with success!");
765       else
766         System.out.println("Error buying tickets!");
767       mainMenu();
768     } else {
769       foundEventsMenu(events);
770     }
771   }
772 }
```

7 Conclusões

7.1 Resultados Obtidos

A equipa conseguiu implementar toda a lista de requisitos estabelecida no ponto 1.1 e concluiu todos os objetivos delineados para o desenvolvimento esta aplicação.

Este projeto contribuiu para uma excelente aprendizagem de VDM++ e maior aprofundamento da matéria lecionada na unidade curricular de Métodos Formais em Engenharia de Software.

7.2 Possíveis Melhoramentos

O projeto poderia ter sido bem mais complexo, no sentido de ter mais classes e interação entre as mesmas, mas foi decidido optar pelas funcionalidades mais essenciais para o bom funcionamento da aplicação e pelas que faziam mais sentido. No entanto, tendo em conta o que foi desenvolvido, seria interessante implementar o registo de utilizadores.

7.3 Contribuição

O trabalho foi igualmente dividido pelos membros do grupo.

8 Referências

1. Material disponibilizado no Moodle
2. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
3. Overture tool website, `http://overturetool.org`
4. Agenda Viral website, `https://www.viralagenda.com`