U. PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Formal Modeling of Jyve in VDM++

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Ilona Generalova

Luís Barbosa

December 03, 2018

Contents

1. Informal system description and list of requirements

1.1 Informal system description

In this project, we wanted to model the social network Jyve. This social network allows to:

- Find artists/bands.
- Find places where is possible to listen to music.
- Find shows.
- See a calendar with shows.
- Create public profiles of artists/bands.
- Create public profiles of places.
- Announce shows.
- Chat with artists/bands/users (to negotiate shows…).

1.2 List of requirements

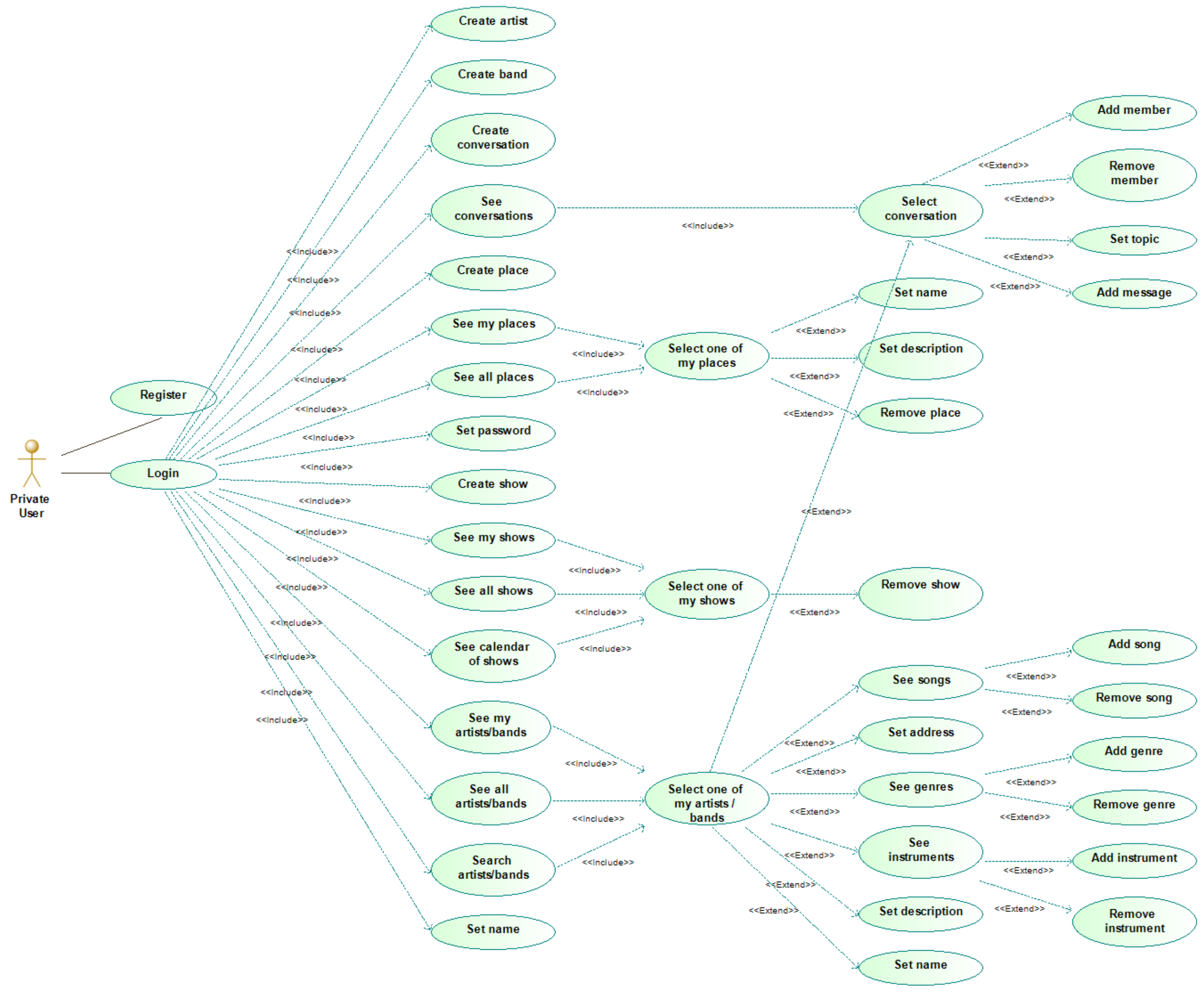
Id	Priority	Description
R001	Mandatory	As an external user/guest, I want to be able to register.
R002	Mandatory	As an external user/guest, I want to be able to login.
R003	Mandatory	As a private user, I want to be able to define the public profile of my artists.
R004	Mandatory	As a private user, I want to be able to find artists.
R005	Mandatory	As a private user, I want to be able to see the public profiles of artists.
R006	Mandatory	As a private user, I want to be able to find the band(s) of an artist.
R007	Optional	As a private user, I want to be able to find the genders of an artist.
R008	Optional	As a private user, I want to be able to find the instruments that an artist plays.
R009	Mandatory	As a private user, I want to be able to define the public profile of my bands.
R010	Mandatory	As a private user, I want to be able to find bands.
R011	Mandatory	As a private user, I want to be able to see the public profiles of bands.
R012	Mandatory	As a private user, I want to be able to find the members of a band.
R013	Optional	As a private user, I want to be able to find the genders of a band.
R014	Optional	As a private user, I want to be able to find the instruments that a band plays.
R015	Mandatory	As a private user, I want to be able to access a calendar with the shows already scheduled.
R016	Mandatory	As a private user, I want to be able to announce shows.
R017	Mandatory	As a private user, I want to be able to find shows and know some information about them as the performers, date, description and address.
R018	Mandatory	As a private user, I want to be able to have a conversation with other users (private users, artists and bands).
R019	Mandatory	As an artist, I want to be able to have a conversation with other users (private users, artists and bands).
R020	Mandatory	As a band, I want to be able to have a conversation with other users (private users, artists and bands).
R021	Mandatory	As a private user, I want to know the date of a message.
R022	Mandatory	As a private user, I want to be able to change the members and the topic of a conversation.
R023	Optional	As a private user, I want to be able to get a link to a song of an artist/band.
R024	Mandatory	As a private user, I want to be able to find places where I can listen to music.

R025	Mandatory	As a private user, I want to be able to define the public profile of my places.
R026	Mandatory	As a private user, I want to be able to see information related to a place.
R027	Mandatory	As a private user, I want to easily go to my artists, bands, places and shows.
R028	Mandatory	As a private user, I want to be able to change my password and name.

These requirements are directly translated onto use cases as shown next.

2. Visual UML model

2.1 Use case model



The major use case scenarios (to be used later as test scenarios) are described next.

Scenario	**Create the public profile of an artist**
Description	Normal scenario to create the public profile of an artist.
Pre-conditions	(unspecified)
Post-conditions	1. The name, address and description of the artist are defined. 2. The artist has no conversations, songs, genres, instruments and bands.

	3. The artist is part of the set of artists. 4. The artist is part of the set of artists of its private user.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to create the public profile of an artist. 3. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Create the public profile of a band**
Description	Normal scenario to create the public profile of a band.
Pre-conditions	1. The minimum number of members of the band is 2. *(input)*
Post-conditions	1. The name, address, description and members of the band are defined. 2. The band has no conversations, songs, genres, instruments. 3. The band is part of the set of bands of each member. 4. The band is part of the set of bands. 5. The band is part of the set of bands of its private user.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to create the public profile of a band. 3. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Create conversation**
Description	Normal scenario to create a conversation with other user(s) (public and private users)
Pre-conditions	1. The minimum number of members of the conversation is 1. *(input)*
Post-conditions	1. The members of the conversation are defined. 2. The topic of the conversation is defined. 3. The conversation is part of the set of conversations of each member.
Steps	Situation 1: 1. The private user performs the login on the system. 2. The private user selects the option that serves to create a conversation. 3. The private user gives the necessary data. Situation 2: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to create a conversation. 5. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Add a member to a conversation**
Description	Normal scenario to add a member to a conversation.
Pre-conditions	(unspecified)
Post-conditions	1. The new member is part of the set of members of the conversation. 2. The previous members still are members of the conversation. 3. The conversation is part of the set of conversations of the new member.
Steps	Situation 1: 1. The private user performs the login on the system.

	2. The private user selects the option that serves to see its conversations. 3. The private user selects one conversation. 4. The private user selects the option that serves to add a member to the conversation. 5. The private user gives the necessary data. Situation 2: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the conversations of the artist/band. 5. The private user selects one conversation. 6. The private user selects the option that serves to add a member to the conversation. 7. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Remove a member from a conversation**
Description	Normal scenario to remove a member from a conversation.
Pre-conditions	1. The user is part of the set of members of the conversation.
Post-conditions	1. The old member isn't part of the set of members of the conversation. 2. All other members still are members of the conversation. 3. The conversation isn't part of the set of conversations of the old member.
Steps	Situation 1: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its conversations. 3. The private user selects one conversation. 4. The private user selects the option that serves to remove a member from the conversation. 5. The private user gives the necessary data. Situation 2: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the conversations of the artist/band. 5. The private user selects one conversation. 6. The private user selects the option that serves to remove a member from the conversation. 7. The private user gives the necessary data.
Exceptions	1. If the conversation had 1 member and now has 0, the conversation is deleted.

Scenario	**Set the topic of a conversation**
Description	Normal scenario to set the new topic of a conversation.
Pre-conditions	(unspecified)

Post-conditions	1. The new topic is defined.
Steps	Situation 1: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its conversations. 3. The private user selects one conversation. 4. The private user selects the option that serves to set the new topic of the conversation. 5. The private user gives the necessary data. Situation 2: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the conversations of the artist/band. 5. The private user selects one conversation. 6. The private user selects the option that serves to set the new topic of the conversation. 7. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Add a message to a conversation**
Description	Normal scenario to add a message to a conversation.
Pre-conditions	1. The sender is part of the set of members of the conversation.
Post-conditions	1. All old messages are preserved. 2. The new message is part of the map of messages.
Steps	Situation 1: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its conversations. 3. The private user selects one conversation. 4. The private user selects the option that serves to add a new message to the conversation. 5. The private user gives the necessary data. Situation 2: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the conversations of the artist/band. 5. The private user selects one conversation. 6. The private user selects the option that serves to add a new message to the conversation. 7. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Create the public profile of a place**
Description	Normal scenario to create the public profile of a place.
Pre-conditions	(unspecified)
Post-conditions	1. The name, description and address are defined. 2. The new place is part of the set of places. 3. The new place is part of the set of places of its private user.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to create the public profile of a place. 3. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Set the name of a place**
Description	Normal scenario to set the new name of a place.
Pre-conditions	(unspecified)
Post-conditions	1. The new name of the place is defined.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its places or the option to see all places. 3. The private user selects one of him places. 4. The private user selects the option that serves to set the new name of the place. 5. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Set the description of a place**
Description	Normal scenario to set the new description of a place.
Pre-conditions	(unspecified)
Post-conditions	1. The new description of the place is defined.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its places or the option to see all places. 3. The private user selects one of him places. 4. The private user selects the option that serves to set the new description of the place. 5. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Disconnect a place from its owner**
Description	Normal scenario to disconnect the public profile of a place from its owner.
Pre-conditions	1. The place is part of the set of places. 2. The place is part of the set of places of its private user.
Post-conditions	1. The place isn't part of the set of places. 2. The place isn't part of the set of places of its private user.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its places or the option to see all places. 3. The private user selects one of him places. 4. The private user selects the option that removes the place.

Exceptions	(unspecified)

Scenario	**Create the private profile of a private user (register)**
Description	Normal scenario to define the private profile of a private user.
Pre-conditions	1. The email must be unique. (input)
Post-conditions	1. The private user name, email and password are defined. 2. The private user has no conversations, artists, bands, places and shows. 3. The private user is part of the set of private users.
Steps	1. The user selects the option that serves to register on the system. 2. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Set the password of a private user**
Description	Normal scenario to set the new password of a private user.
Pre-conditions	1. The password has 8 or more characters.
Post-conditions	1. The password is defined.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to change its password. 3. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Create a show**
Description	Normal scenario to create a new show.
Pre-conditions	1. The show has at least one performer. 2. The start date of the show is before or equal to its end date.
Post-conditions	1. The name, performers, start date, end date, place and description of the show are defined. *(input)* 2. The show is part of the set of shows. 3. The show is part of the set of shows of its private user.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to create a new show. 3. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Remove a show**
Description	Normal scenario to remove a show.
Pre-conditions	1. The show is part of the set of shows. 2. The show is part of the set of shows of its private user.
Post-conditions	1. The show isn't part of the set of shows. 2. The show isn't part of the set of shows of its private user.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its shows or the option to see all shows in the calendar. 3. The private user selects one of him shows. 4. The private user selects the option that removes the show.
Exceptions	(unspecified)

Scenario	**Access a calendar with all shows**
Description	Normal scenario to see all shows in an ordered manner.
Pre-conditions	(unspecified)
Post-conditions	1. The shows are sorted by start date.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see the calendar of shows.
Exceptions	(unspecified)

Scenario	**Add a song to the profile of an artist or band**
Description	Normal scenario to add a new song to the profile of an artist or band.
Pre-conditions	1. The link of the new song is unique relatively to all songs of all users. *(input)*
Post-conditions	1. The song is part of the set of songs of the artist/band. 2. The song is part of the set that contains all songs.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the songs of the artist/band. 5. The private user selects the option that serves to add a new song. 6. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Remove a song from the profile of an artist or band**
Description	Normal scenario to remove a song from the profile of an artist or band.
Pre-conditions	1. The song is part of the set of songs of the artist/band. 2. The song is part of the set that contains all songs.
Post-conditions	1. The song isn't part of the set of songs of the artist/band. 2. The song isn't part of the set that contains all.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the songs of the artist/band. 5. The private user selects the option that removes the song.
Exceptions	(unspecified)

Scenario	**Set the address of an artist or band**
Description	Normal scenario to set the new address of an artist or band.
Pre-conditions	(unspecified)
Post-conditions	1. The address is defined.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands.

	3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to change the address of the artist/band. 5. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Add a genre to the profile of an artist or band**
Description	Normal scenario to add a genre to the public profile of an artist or band.
Pre-conditions	1. The genre isn't part of the set of genres of the artist/band.
Post-conditions	1. The genre is part of the set of genres of the artist/band.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the genres of the artist/band. 5. The private user selects the option that serves to add a genre. 6. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Remove a genre from the profile of an artist or band**
Description	Normal scenario to remove a genre from the public profile of an artist or band.
Pre-conditions	1. The genre is part of the set of genres of the artist/band.
Post-conditions	1. The genre isn't part of the set of genres of the artist/band.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the genres of the artist/band. 5. The private user selects the option that removes the genre.
Exceptions	(unspecified)

Scenario	**Add an instrument to the profile of an artist or band**
Description	Normal scenario to add an instrument to the public profile of an artist or band.
Pre-conditions	1. The instrument isn't part of the set of instruments of the artist/band.
Post-conditions	1. The instrument is part of the set of instruments of the artist/band.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the instruments of the artist/band. 5. The private user selects the option that serves to add an instrument. 6. The private user gives the necessary data.
Exceptions	(unspecified)

Scenario	**Remove an instrument from the profile of an artist or band**
Description	Normal scenario to remove an instrument from public profile of an artist or band.
Pre-conditions	1. The instrument is part of the set of instruments of the artist/band.
Post-conditions	1. The instrument isn't part of the set of instruments of the artist/band.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to see the instruments of the artist/band. 5. The private user selects the option that removes the instrument.
Exceptions	(unspecified)

Scenario	**Set the description of the profile of an artist or band**
Description	Normal scenario to set the new description of the public profile of an artist or band.
Pre-conditions	(unspecified)
Post-conditions	1. The description is defined.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to set the description of the artist/band. 5. The private user gives the necessary data.
Exceptions	(unspecified)

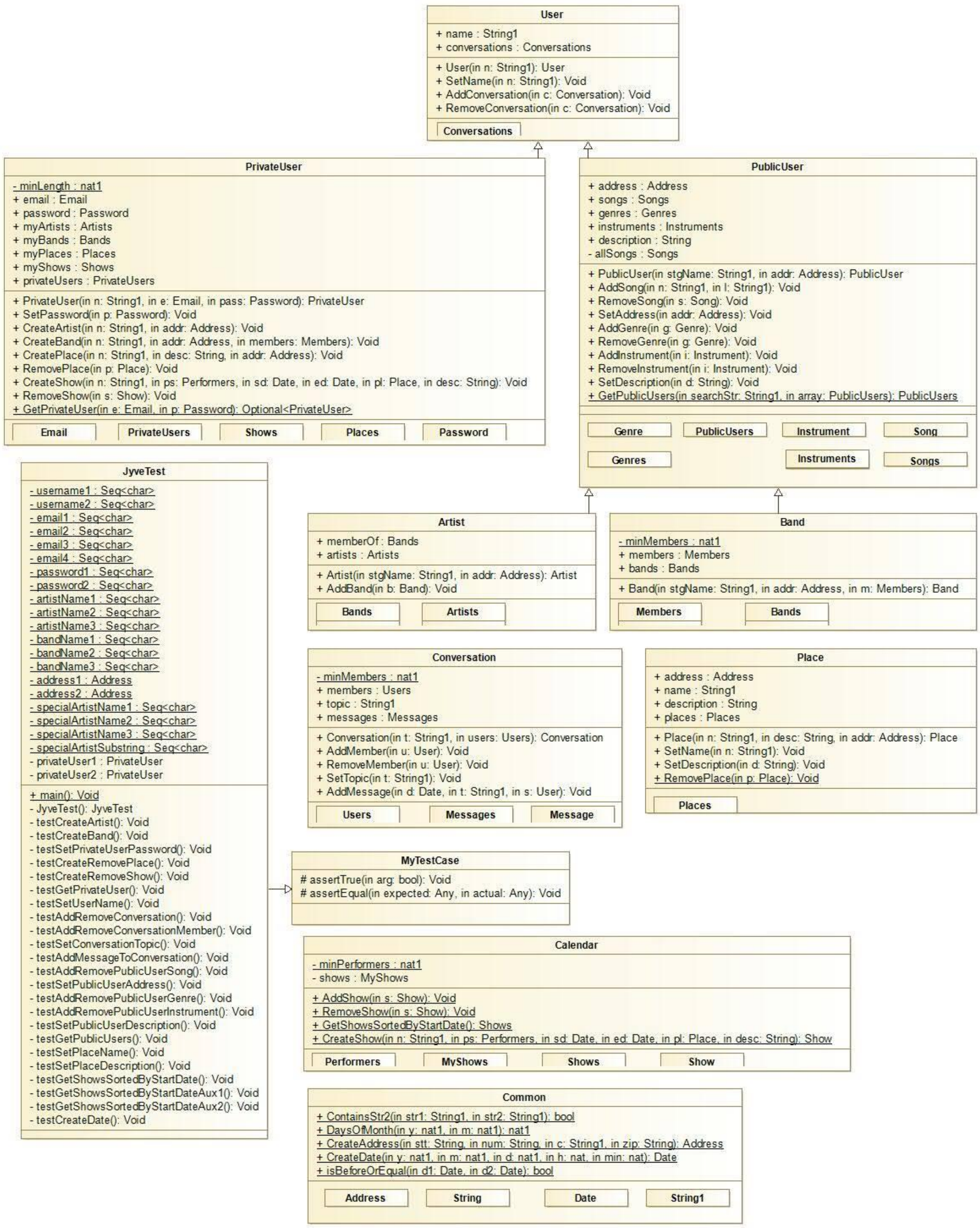
Scenario	**Search for artists or bands**
Description	Normal scenario to search for the public profile of artists or bands.
Pre-conditions	(unspecified)
Post-conditions	1. The result of the search is a subset of the set of artists/bands.
Steps	1. The private user performs the login on the system. 2. The private user selects the option that serves to search for artists/bands. 3. The private user gives the data necessary to perform the search.
Exceptions	(unspecified)

Scenario	**Set the name of a private user, artist or band**
Description	Normal scenario to set the new name of any user in its profile.
Pre-conditions	(unspecified)
Post-conditions	1. The name is defined.
Steps	Situation 1: 1. The private user performs the login on the system. 2. The private user selects the option that serves to set its name. 3. The private user gives the necessary data.

	Situation 2: 1. The private user performs the login on the system. 2. The private user selects the option that serves to see its artists/bands, the option to see all artists/bands or the option to search for artists/bands. 3. The private user selects one of its artists/bands. 4. The private user selects the option that serves to set the name of the artist/band. 5. The private user gives the necessary data.
Exceptions	(unspecified)

There are some use case scenarios present in the use case model that are not present in the above-detailed section because they are scenarios related with the visualization of information which is done by the interface by directly accessing the variables related to the information.

2.2 Class model


User
+ name : String1
+ conversations : Conversations
+ User(in n: String1): User
+ SetName(in n: String1): Void
+ AddConversation(in c: Conversation): Void
+ RemoveConversation(in c: Conversation): Void
Conversations
PrivateUser
- minLength : nat1
+ email : Email
+ password : Password
+ myArtists : Artists
+ myBands : Bands
+ myPlaces : Places
+ myShows : Shows
+ privateUsers : PrivateUsers
+ PrivateUser(in n: String1, in e: Email, in pass: Password): PrivateUser
+ SetPassword(in p: Password): Void
+ CreateArtist(in n: String1, in addr: Address): Void
+ CreateBand(in n: String1, in addr: Address, in members: Members): Void
+ CreatePlace(in n: String1, in desc: String, in addr: Address): Void
+ RemovePlace(in p: Place): Void
+ CreateShow(in n: String1, in ps: Performers, in sd: Date, in ed: Date, in pl: Place, in desc: String): Void
+ RemoveShow(in s: Show): Void
+ GetPrivateUser(in e: Email, in p: Password): Optional<PrivateUser>
Email
PrivateUsers
Shows
Places
Password
PublicUser
+ address : Address
+ songs : Songs
+ genres : Genres
+ instruments : Instruments
+ description : String
- allSongs : Songs
+ PublicUser(in stgName: String1, in addr: Address): PublicUser
+ AddSong(in n: String1, in l: String1): Void
+ RemoveSong(in s: Song): Void
+ SetAddress(in addr: Address): Void
+ AddGenre(in g: Genre): Void
+ RemoveGenre(in g: Genre): Void
+ AddInstrument(in i: Instrument): Void
+ RemoveInstrument(in i: Instrument): Void
+ SetDescription(in d: String): Void
+ GetPublicUsers(in searchStr: String1, in array: PublicUsers): PublicUsers
Genre
PublicUsers
Instrument
Song
Genres
Instruments
Songs
JyveTest
- username1 : Seq<char>
- username2 : Seq<char>
- email1 : Seq<char>
- email2 : Seq<char>
- email3 : Seq<char>
- email4 : Seq<char>
- password1 : Seq<char>
- password2 : Seq<char>
- artistName1 : Seq<char>
- artistName2 : Seq<char>
- artistName3 : Seq<char>
- bandName1 : Seq<char>
- bandName2 : Seq<char>
- bandName3 : Seq<char>
- address1 : Address
- address2 : Address
- specialArtistName1 : Seq<char>
- specialArtistName2 : Seq<char>
- specialArtistName3 : Seq<char>
- specialArtistSubstring : Seq<char>
- privateUser1 : PrivateUser
- privateUser2 : PrivateUser
+ main(): Void
- JyveTest(): JyveTest
- testCreateArtist(): Void
- testCreateBand(): Void
- testSetPrivateUserPassword(): Void
- testCreateRemovePlace(): Void
- testCreateRemoveShow(): Void
- testGetPrivateUser(): Void
- testSetUserName(): Void
- testAddRemoveConversation(): Void
- testAddRemoveConversationMember(): Void
- testSetConversationTopic(): Void
- testAddMessageToConversation(): Void
- testAddRemovePublicUserSong(): Void
- testSetPublicUserAddress(): Void
- testAddRemovePublicUserGenre(): Void
- testAddRemovePublicUserInstrument(): Void
- testSetPublicUserDescription(): Void
- testGetPublicUsers(): Void
- testSetPlaceName(): Void
- testSetPlaceDescription(): Void
- testGetShowsSortedByStartDate(): Void
- testGetShowsSortedByStartDateAux1(): Void
- testGetShowsSortedByStartDateAux2(): Void
- testCreateDate(): Void
Artist
+ memberOf : Bands
+ artists : Artists
+ Artist(in stgName: String1, in addr: Address): Artist
+ AddBand(in b: Band): Void
Bands
Artists
Band
- minMembers : nat1
+ members : Members
+ bands : Bands
+ Band(in stgName: String1, in addr: Address, in m: Members): Band
Members
Bands
Conversation
- minMembers : nat1
+ members : Users
+ topic : String1
+ messages : Messages
+ Conversation(in t: String1, in users: Users): Conversation
+ AddMember(in u: User): Void
+ RemoveMember(in u: User): Void
+ SetTopic(in t: String1): Void
+ AddMessage(in d: Date, in t: String1, in s: User): Void
Users
Messages
Message
Place
+ address : Address
+ name : String1
+ description : String
+ places : Places
+ Place(in n: String1, in desc: String, in addr: Address): Place
+ SetName(in n: String1): Void
+ SetDescription(in d: String): Void
+ RemovePlace(in p: Place): Void
Places
MyTestCase
# assertTrue(in arg: bool): Void
# assertEqual(in expected: Any, in actual: Any): Void
Calendar
- minPerformers : nat1
- shows : MyShows
+ AddShow(in s: Show): Void
+ RemoveShow(in s: Show): Void
+ GetShowsSortedByStartDate(): Shows
+ CreateShow(in n: String1, in ps: Performers, in sd: Date, in ed: Date, in pl: Place, in desc: String): Show
Performers
MyShows
Shows
Show
Common
+ ContainsStr2(in str1: String1, in str2: String1): bool
+ DaysOfMonth(in y: nat1, in m: nat1): nat1
+ CreateAddress(in stt: String, in num: String, in c: String1, in zip: String): Address
+ CreateDate(in y: nat1, in m: nat1, in d: nat1, in h: nat, in min: nat): Date
+ isBeforeOrEqual(in d1: Date, in d2: Date): bool
Address
String
Date
String1

Class	Description
`Artist`	Represents the public profile of an artist.
Band	Represents the public profile of a band.

Calendar	Represents a calendar with all shows.
Common	Contains types, operations, and functions that are used in several classes.
Conversation	Represents a conversation between elements of the class 'User'.
JyveTest	Contains the test cases for Jyve, covering all usage scenarios.
MyTestCase	Superclass for test classes; defines `assertEquals` and `assertTrue`.
Place	Represents the public profile of a place.
PrivateUser	Represents the private profile of a user which is able to log in.
PublicUser	Represents any public profile (this class could be abstract).
User	Represents any user (this class could be abstract).

3. Formal VDM++ model

The coverage tables indicate 100% of coverage for all classes of this section.

3.1 Class Artist

```
/* Represents the public profile of an artist. */
class Artist is subclass of PublicUser
types
        public static Bands = set of Band;
        public static Artists = set of Artist;
values
instance variables
        public memberOf : Bands;
        public static artists : Artists := {};
operations
        public Artist : Common`String1 * Common`Address ==> Artist
        Artist(stgName,addr) == (
                memberOf := {};
                artists := artists union {self};
                PublicUser(stgName,addr);
        )
        post memberOf = {} and artists~ = artists \ {self};

        public AddBand : Band ==> ()
        AddBand(b) == memberOf := memberOf union {b}
        pre b not in set memberOf and self in set b.members
        post b in set memberOf and memberOf~ = memberOf \ {b} and self in set b.members;
functions
traces
end Artist
```

Function or operation	Line	Coverage	Calls
AddBand	19	100%	42
Artist	11	100%	81
Artist.vdmpp		100%	123

3.2 Class Band

```
/* Represents the public profile of a band. */
class Band is subclass of PublicUser
types
        public static Members = set1 of Artist
                inv m == card m >= minMembers;
        public static Bands = set of Band;
```

```
values
	private static minMembers = 2;
instance variables
	public members : Members;
	public static bands : Bands := {};
	inv forall band in set bands & (forall artist in set band.members & band in set
artist.memberOf);
operations
	public Band : Common`String1 * Common`Address * Members ==> Band
	Band(stgName,addr,m) == (
		members := m;
		for all member in set members do
			member.AddBand(self);
		bands := bands union {self};
		PublicUser(stgName,addr);
	)
	pre card m >= minMembers  -- Verification of the invariant.
	post members = m    -- Verification of the invariant by verifying that the
assignment is correct (already passed precondition).
		and forall member in set members & self in set member.memberOf
		and self in set bands;
functions
traces
end Band
```

Function or operation	Line	Coverage	Calls
Band	14	100%	14
Band.vdmpp		100%	14

3.3 Class Calendar

```
/* Represents a calendar with all shows. */
class Calendar
types
	public static Performers = set1 of PublicUser;
	public static Show ::
		name : Common`String1
		performers : Performers
		startDate : Common`Date
		endDate : Common`Date
		place : Place
		description : Common`String;
	public static Shows = seq of Show;
	private static MyShows = set of Show;
values
	private minPerformers = 1;
instance variables
	private static shows : MyShows := {};
operations
	public static AddShow : Show ==> ()
	AddShow(s) == shows := shows union {s}
	pre s not in set shows
	post s in set shows and shows~ = shows \ {s};

	public static RemoveShow : Show ==> ()
	RemoveShow(s) == shows := shows \ {s}
	pre s in set shows
	post s not in set shows and shows~ = shows union {s};

	-- Get shows ordered by start date (based on selection sort:
https://www.geeksforgeeks.org/selection-sort/).
	public static GetShowsSortedByStartDate : () ==> Shows
	GetShowsSortedByStartDate() == (
		dcl newShows : Shows := [];
		for all s in set shows do
			newShows := newShows ^ [s];

		for i=1 to len newShows - 1 do (
			dcl minIdx : nat1 := i;
			dcl t : Show := newShows(i);

			for j=i+1 to len newShows do

	if(Common`isBeforeOrEqual(newShows(j).startDate,newShows(minIdx).startDate)) then
					minIdx := j;

			newShows(i) := newShows(minIdx);
			newShows(minIdx) := t;
		);
		return newShows;
	)
	post forall i, j in set inds RESULT & i < j =>
Common`isBeforeOrEqual(RESULT(i).startDate, RESULT(j).startDate);

functions
	public static CreateShow : Common`String1 * Performers * Common`Date * Common`Date
* Place * Common`String +> Show
	CreateShow(n,ps,sd,ed,pl,desc) == mk_Show(n,ps,sd,ed,pl,desc)
	pre card ps >= minPerformers and Common`isBeforeOrEqual(sd,ed)
	post RESULT.name = n and RESULT.performers = ps and RESULT.startDate = sd
		and RESULT.endDate = ed and RESULT.place = pl and RESULT.description =
desc;
traces
end Calendar
```

Function or operation	Line	Coverage	Calls
AddShow	19	100%	134
CreateShow	52	100%	67
GetShowsSortedByStartDate	30	100%	13
RemoveShow	24	100%	67
Calendar.vdmpp		100%	281

3.4 Class Common

```
/* Contains types, operations, and functions that are used in several classes. */
class Common
types
	public static String = seq of char;
	public static String1 = seq1 of char;
	public static Address ::
		street : String
		number : String
		city : String1
		zipcode : String;
	public static Date ::
		year : nat1
		month : nat1
		day : nat1
		hour : nat
		minute : nat
		inv d == d.month  <= 12 and d.day <= DaysOfMonth(d.year, d.month) and
d.hour < 24 and d.minute < 60;
values
instance variables
operations
	public static ContainsStr2 : String1 * String1 ==> bool
	ContainsStr2(str1,str2) == (
		for i = 1 to len str1 - len str2 do (
			dcl res : bool := true;
			for j = 1 to len str2 do (
				if (str1(i+j-1) <> str2(j)) then
					res := false;
			);
			if(res) then
				return res;
		);
		return false;
	);
functions
	public static DaysOfMonth : nat1 * nat1 +> nat1
	DaysOfMonth(y,m) == (
		if (m = 2) then (
			if (y mod 400 = 0) or ((y mod 4 = 0) and (y mod 100 <> 0))
				then 29
			else 28
		)
		else 31 - (m - 1) mod 7 mod 2
	)
	pre m <= 12
	post if m = 2 then RESULT in set {28,29} else RESULT in set {30,31};

	public static CreateAddress : String * String * String1 * String +> Address
	CreateAddress(stt,num,c,zip) == mk_Address(stt,num,c,zip)
	post RESULT.street = stt and RESULT.number = num and RESULT.city = c and
RESULT.zipcode = zip;

	public static CreateDate : nat1 * nat1 * nat1 * nat * nat +> Date
	CreateDate(y,m,d,h,min) == mk_Date(y,m,d,h,min)
	pre m >= 1 and m <= 12 and d >=1 and d <= DaysOfMonth(y,m) and h >= 0 and h < 24
and min >= 0 and min < 60 -- Verification of the invariant.
	post RESULT.year = y and RESULT.month = m and RESULT.day = d and RESULT.hour = h
and RESULT.minute = min; -- Verification of the invariant by verifying that the
assignment is correct (already passed precondition).

	public static isBeforeOrEqual : Date * Date +> bool
	isBeforeOrEqual(d1,d2) == d2.year > d1.year or (d2.year = d1.year and (d2.month >
d1.month or d2.month = d1.month and d2.day >= d1.day));
traces
end Common
```

Function or operation	Line	Coverage	Calls
ContainsStr2	21	100%	1287

CreateAddress	47	100%	28
CreateDate	51	100%	266
DaysOfMonth	35	100%	80
isBeforeOrEqual	56	100%	184
Common.vdmpp		100%	1845

3.5 Class Conversation

```
/* Represents a conversation between elements of the class 'User'. */
class Conversation
types
	public static Users = set of User;
	public static Message ::
		date : Common`Date
		text : Common`String1
		sender : User;
	public static Messages = inmap nat1 to Message;
values
	private static minMembers = 1;
instance variables
	public members : Users;
	public topic : Common`String1;
	public messages : Messages := { |-> };
operations
	public Conversation : Common`String1 * Users ==> Conversation
	Conversation(t, users) == (
		topic := t;
		members := users;
		for all user in set members do
			user.AddConversation(self);
	)
	pre card users >= minMembers
	post members = users
		and topic = t
		and forall user in set members & self in set user.conversations;

	public AddMember : User ==> ()
	AddMember(u) == (
		if (u not in set members) then (
			members := members union {u};
			u.AddConversation(self);
		);
	)
	post u in set members and (members~ = members or members~ = members \ {u}) and
self in set u.conversations;

	-- If 'members' becomes empty, the conversation is expected to be lost and
automatically destroyed by JVM.
	public RemoveMember : User ==> ()
	RemoveMember(u) == (
		if (u in set members) then (
			members := members \ {u};
			u.RemoveConversation(self);
		);
	)
	post u not in set members and members~ = members union {u} and self not in set
u.conversations;
```

```
        public SetTopic : Common`String1 ==> ()
        SetTopic(t) == topic := t
        post topic = t;

        public AddMessage : Common`Date * Common`String1 * User ==> ()
        AddMessage(d,t,s) == (
                dcl msg : Message := mk_Message(d,t,s);
                dcl myMap : Messages := { card dom messages + 1 |-> msg };
                messages := messages munion myMap;
        )
        pre s in set members
        post card dom messages = card dom messages~ + 1;
functions
traces
end Conversation
```

Function or operation	Line	Coverage	Calls
AddMember	29	100%	84
AddMessage	52	100%	14
Conversation	17	100%	112
RemoveMember	39	100%	70
SetTopic	48	100%	14
Conversation.vdmpp		100%	294

3.6 Class Place

```
/* Represents the public profile of a place. */
class Place
types
        public static Places = set of Place;
values
instance variables
        public name : Common`String1;
        public description : Common`String;
        public address : Common`Address;
        public static places : Places := {};
operations
        public Place : Common`String1 * Common`String * Common`Address ==> Place
        Place(n,desc,addr) == (
                name := n;
                description := desc;
                address := addr;
                places := places union {self};
        )
        post name = n and description = desc and address = addr and places~ = places \
{self};

        public SetName : Common`String1 ==> ()
        SetName(n) == name := n
        post name = n;

        public SetDescription : Common`String ==> ()
        SetDescription(d) == description := d
        post description = d;

        public static RemovePlace : Place ==> ()
        RemovePlace(p) == places := places \ {p}
        pre p in set places
        post p not in set places and places~ = places union {p};
functions
traces
end Place
```

Function or operation	Line	Coverage	Calls
Place	12	100%	67
RemovePlace	29	100%	134
SetDescription	25	100%	13
SetName	21	100%	13
Place.vdmpp		100%	227

3.7 Class PrivateUser

```
/* Represents the private profile of a user which is able to log in. */
class PrivateUser is subclass of User
types
        public static Email = Common`String1;
        public static Password = Common`String1
                inv p == len p >= minLength;
        public static Places = set of Place;
        public static Shows = set of Calendar`Show;
        private static PrivateUsers = set of PrivateUser;
values
        private minLength = 8;
instance variables
        public email : Email;
        public password : Password;
        public myArtists : Artist`Artists;
        public myBands : Band`Bands;
        public myPlaces : Places;
        public myShows : Shows;
        public static privateUsers : PrivateUsers := {};
operations
        public PrivateUser : Common`String1 * Email * Password ==> PrivateUser
        PrivateUser(n,e,pass) == (
                email := e;
                password := pass;
                myArtists := {};
                myBands := {};
                myPlaces := {};
                myShows := {};
                privateUsers := privateUsers union {self};
                User(n);
        )
        pre len pass >= minLength and forall u in set privateUsers & u.email <> e     --
Verification of the invariant on 'pass'.
        post email = e and password = pass and myArtists = {} and myBands = {} and
myPlaces = {}  -- Verification of the invariant on 'pass' by verifying that the
assignment is correct (already passed precondition).
                and myShows = {} and self in set privateUsers;

        public SetPassword : Password ==> ()
        SetPassword(p) == password := p
        pre len p >= minLength -- Verification of the invariant.
```

```
	post password = p; -- Verification of the invariant by verifying that the
assignment is correct (already passed precondition).

	public CreateArtist : Common`String1 * Common`Address ==> ()
	CreateArtist(n,addr) == myArtists := myArtists union {new Artist(n,addr)}
	post exists1 a in set myArtists & a.name = n and a.address = addr;

	public CreateBand : Common`String1 * Common`Address * Band`Members ==> ()
	CreateBand(n,addr,members) == myBands := myBands union {new Band(n,addr,members)}
	post exists1 b in set myBands & b.name = n and b.address = addr and b.members =
members;

	public CreatePlace : Common`String1 * Common`String * Common`Address ==> ()
	CreatePlace(n,desc,addr) == myPlaces := myPlaces union {new Place(n,desc,addr)}
	post exists1 p in set myPlaces & p.name = n and p.description = desc and p.address
= addr;

	public RemovePlace : Place ==> ()
	RemovePlace(p) == (
		myPlaces := myPlaces \ {p};
		Place`RemovePlace(p);
	)
	pre p in set myPlaces
	post p not in set myPlaces and myPlaces~ = myPlaces union {p};

	public CreateShow : Common`String1 * Calendar`Performers * Common`Date *
Common`Date * Place * Common`String ==> ()
	CreateShow(n,ps,sd,ed,pl,desc) == (
		dcl s : Calendar`Show := Calendar`CreateShow(n,ps,sd,ed,pl,desc);
		myShows := myShows union {s};
		Calendar`AddShow(s);
	)
	post exists1 s in set myShows & s.name = n and s.performers = ps and s.startDate =
sd and s.endDate = ed and s.place = pl and s.description = desc;

	public RemoveShow : Calendar`Show ==> ()
	RemoveShow(s) == (
		myShows := myShows \ {s};
		Calendar`RemoveShow(s);
	)
	pre s in set myShows
	post s not in set myShows and myShows~ = myShows union {s};

	public static GetPrivateUser : Email * Password ==> [PrivateUser]
	GetPrivateUser(e,p) == (
		for all u in set privateUsers do
			if (u.email = e and u.password = p) then
				return u;
		return nil;
	)
	pre len p >= minLength    -- Verification of the invariant.
	post RESULT = nil or (RESULT in set privateUsers and RESULT.email = e and
RESULT.password = p); -- Verification of the invariant by comparing the result with the
input (already passed precondition).
functions
traces
end PrivateUser
```

Function or operation	Line	Coverage	Calls

CreateArtist	41	100%	81
CreateBand	45	100%	14
CreatePlace	49	100%	67
CreateShow	61	100%	67
GetPrivateUser	77	100%	14
PrivateUser	21	100%	28
RemovePlace	53	100%	67
RemoveShow	69	100%	67
SetPassword	36	100%	28
PrivateUser.vdmpp		100%	433

3.8 Class PublicUser

```
/*
	Represents any public profile and, among other
	features, illustrates the usage of 'atomic'.
*/
class PublicUser is subclass of User
types
	public static Genre = Common`String1;
	public static Genres = set of Genre;
	public static Instrument = Common`String1;
	public static Instruments = set of Instrument;
	public static Song ::
		name : Common`String1
		link : Common`String1;
	public static Songs = set of Song;
	public static PublicUsers = set of PublicUser;
values
instance variables
	public songs : Songs;
	public address : Common`Address;
	public genres : Genres;
	public instruments : Instruments;
	public description : Common`String;
	private static allSongs : Songs := {};
	inv forall s in set songs & s in set allSongs;
operations
	public PublicUser : Common`String1 * Common`Address ==> PublicUser
	PublicUser(stgName, addr) == (
		songs := {};
		address := addr;
		genres := {};
		instruments := {};
		description := "";
		User(stgName);
	)
	post songs = {} and address = addr and genres = {} and instruments = {} and
description = "";

	public AddSong : Common`String1 * Common`String1 ==> ()
	AddSong(n,l) == (
		dcl s : Song := mk_Song(n,l);
		atomic(
			songs := songs union {s};
			allSongs := allSongs union {s};
```

```
                );
        )
        pre forall s1 in set songs & s1.link <> 1     -- Double check relatively to
'allSongs'.
                and forall s2 in set allSongs & s2.link <> 1
        post exists1 s1 in set songs & s1.name = n and s1.link = 1
                and exists1 s2 in set allSongs & s2.name = n and s2.link = 1
                and s1 = s2;

        public RemoveSong : Song ==> ()
        RemoveSong(s) == (
                atomic(
                        songs := songs \ {s};
                        allSongs := allSongs \ {s};
                );
        )
        pre s in set songs and s in set allSongs
        post s not in set songs and songs~ = songs union {s}
                and s not in set allSongs and allSongs~ = allSongs union {s};

        public SetAddress : Common`Address ==> ()
        SetAddress(addr) == address := addr
        post address = addr;

        public AddGenre : Genre ==> ()
        AddGenre(g) == genres := genres union {g}
        pre g not in set genres
        post g in set genres and genres~ = genres \ {g};

        public RemoveGenre : Genre ==> ()
        RemoveGenre(g) == genres := genres \ {g}
        pre g in set genres
        post g not in set genres and genres~ = genres union {g};

        public AddInstrument : Instrument ==> ()
        AddInstrument(i) == instruments := instruments union {i}
        pre i not in set instruments
        post i in set instruments and instruments~ = instruments \ {i};

        public RemoveInstrument : Instrument ==> ()
        RemoveInstrument(i) == instruments := instruments \ {i}
        pre i in set instruments
        post i not in set instruments and instruments~ = instruments union {i};

        public SetDescription : Common`String ==> ()
        SetDescription(d) == description := d
        post description = d;

        public static GetPublicUsers : Common`String1 * PublicUsers ==> PublicUsers
        GetPublicUsers(searchStr,array) == (
                dcl results : PublicUsers := {};
                for all u in set array do
                        if(Common`ContainsStr2(u.name,searchStr)) then
                                results := results union {u};
                return results;
        )
        post RESULT subset array;
functions
traces
```

```
end PublicUser
```

Function or operation	Line	Coverage	Calls
AddGenre	66	100%	10
AddInstrument	76	100%	10
AddSong	37	100%	27
GetPublicUsers	90	100%	30
PublicUser	26	100%	95
RemoveGenre	71	100%	10
RemoveInstrument	81	100%	10
RemoveSong	51	100%	20
SetAddress	62	100%	20
SetDescription	86	100%	20
PublicUser.vdmpp		100%	252

3.9 Class User

```
/* Represents any user. */
class User
types
	public static Conversations = set of Conversation;
values
instance variables
	public name : Common`String1;
	public conversations : Conversations;
operations
	public User : Common`String1 ==> User
	User(n) == (
		name := n;
		conversations := {};
	)
	post name = n and conversations = {};

	public SetName : Common`String1 ==> ()
	SetName(n) == name := n
	post name = n;

	public AddConversation : Conversation ==> ()
	AddConversation(c) == (
		if (c not in set conversations) then (
			conversations := conversations union {c};
			c.AddMember(self);
		);
	)
	post c in set conversations and (conversations~ = conversations or conversations~
= conversations \ {c}) and self in set c.members;

	public RemoveConversation : Conversation ==> ()
	RemoveConversation(c) == (
		if (c in set conversations) then (
			conversations := conversations \ {c};
			c.RemoveMember(self);
		);
	)
	post c not in set conversations and (conversations~ = conversations or
conversations~ = conversations union {c}) and self not in set c.members;
```

```
functions
traces
end User
```

Function or operation	Line	Coverage	Calls
AddConversation	21	100%	70
RemoveConversation	30	100%	70
SetName	17	100%	28
User	10	100%	123
User.vdmpp		100%	291

4. Model validation

The coverage tables indicate 100% of coverage for class "JyveTest" and all requirements are covered by the tests.

4.1 Class MyTestCase

```
class MyTestCase
/*
 Superclass for test classes, simpler but more practical than VDMUnit`TestCase.
 For proper use, you have to do: New -> Add VDM Library -> IO.
 JPF, FEUP, MFES, 2014/15.
*/
operations
       -- Simulates assertion checking by reducing it to pre-condition checking.
       -- If 'arg' does not hold, a pre-condition violation will be signaled.
       protected assertTrue: bool ==> ()
       assertTrue(arg) == return
       pre arg;

       -- Simulates assertion checking by reducing it to post-condition checking.
       -- If values are not equal, prints a message in the console and generates
       -- a post-conditions violation.
       protected assertEqual: ? * ? ==> ()
       assertEqual(expected, actual) ==
       if expected <> actual then (
        IO`print("Actual value (");
        IO`print(actual);
        IO`print(") different from expected (");
        IO`print(expected);
        IO`println(")\n")
       )
       post expected = actual

end MyTestCase
```

Function or operation	Line	Coverage	Calls
assertEqual	17	38.8%	190
assertTrue	10	100%	414
MyTestCase.vdmpp		45.0%	604

4.2 Class JyveTest

```
class JyveTest is subclass of MyTestCase
/*
 Contains the test cases for Jyve.
 The test cases cover all usage scenarios,
 merging the creations and deletions in the
 same test to maintain tests integrity.
 Some tests verify if the variables aren't
 'undefined' to assure that the interface
 can access the variables to show the information.
*/
types
values
      -- All values in this section must be distinct.
      private static username1 = "luis";
      private static username2 = "luis2";
      private static email1 = "luis@fe.up.pt";
      private static email2 = "luis2@fe.up.pt";
      private static email3 = "luis3@fe.up.pt";
      private static email4 = "luis4@fe.up.pt";
      private static password1 = "12345678";
      private static password2 = "1234567";
      private static artistName1 = "artist1";
      private static artistName2 = "artist2";
      private static artistName3 = "artist3";
      private static bandName1 = "band1";
      private static bandName2 = "band2";
      private static bandName3 = "band3";
      private static address1 = Common`CreateAddress("","","Porto","");
      private static address2 = Common`CreateAddress("","","Viana do Castelo","");
      private static specialArtistName1 = "SpecialArtist1";
      private static specialArtistName2 = "SpecialArtist2";
      private static specialArtistName3 = "SpecialArtist3";
      private static specialArtistSubstring = "ecialArtist";
instance variables
      private privateUser1 : PrivateUser;
      private privateUser2 : PrivateUser;
operations
      public static main: () ==> ()
      main() ==
      (
            dcl jyveTest: JyveTest := new JyveTest();

            IO`print("testCreateArtist: ");
            jyveTest.testCreateArtist();
            IO`println("Success");

            IO`print("testCreateBand: ");
            jyveTest.testCreateBand();
            IO`println("Success");

            IO`print("testSetPrivateUserPassword: ");
            jyveTest.testSetPrivateUserPassword();
            IO`println("Success");

            IO`print("testCreateRemovePlace: ");
            jyveTest.testCreateRemovePlace();
            IO`println("Success");
```

```
IO`print("testCreateRemoveShow: ");
jyveTest.testCreateRemoveShow();
IO`println("Success");

IO`print("testGetPrivateUser: ");
jyveTest.testGetPrivateUser();
IO`println("Success");

IO`print("testSetUserName: ");
jyveTest.testSetUserName();
IO`println("Success");

IO`print("testAddRemoveConversation: ");
jyveTest.testAddRemoveConversation();
IO`println("Success");

IO`print("testAddRemoveConversationMember: ");
jyveTest.testAddRemoveConversationMember();
IO`println("Success");

IO`print("testSetConversationTopic: ");
jyveTest.testSetConversationTopic();
IO`println("Success");

IO`print("testAddMessageToConversation: ");
jyveTest.testAddMessageToConversation();
IO`println("Success");

IO`print("testAddRemovePublicUserSong: ");
jyveTest.testAddRemovePublicUserSong();
IO`println("Success");

IO`print("testSetPublicUserAddress: ");
jyveTest.testSetPublicUserAddress();
IO`println("Success");

IO`print("testAddRemovePublicUserGenre: ");
jyveTest.testAddRemovePublicUserGenre();
IO`println("Success");

IO`print("testAddRemovePublicUserInstrument: ");
jyveTest.testAddRemovePublicUserInstrument();
IO`println("Success");

IO`print("testSetPublicUserDescription: ");
jyveTest.testSetPublicUserDescription();
IO`println("Success");

IO`print("testGetPublicUsers: ");
jyveTest.testGetPublicUsers();
IO`println("Success");

IO`print("testSetPlaceName: ");
jyveTest.testSetPlaceName();
IO`println("Success");

IO`print("testSetPlaceDescription: ");
jyveTest.testSetPlaceDescription();
```

```
            IO`println("Success");

            IO`print("testGetShowsSortedByStartDate: ");
            jyveTest.testGetShowsSortedByStartDate();
            IO`println("Success");

            IO`print("testCreateDate: ");
            jyveTest.testCreateDate();
            IO`println("Success");

            IO`println("All tests completed with success");
        );


        /* Constructor */
        -- Covering requirements R001 and R027.
        private JyveTest : () ==> JyveTest
        JyveTest() == (
            -- The next line should make the test fail: password length lower than 8.
            --privateUser1 := new PrivateUser(username1,email3,password2);
            privateUser1 := new PrivateUser(username1,email1,password1);
            -- The next line should make the test fail: email already used.
            --privateUser2 := new PrivateUser(username1,email1,password1);
            privateUser2 := new PrivateUser(username1,email2,password1);
            assertTrue(privateUser1 <> privateUser2);

            -- Verifying that the variables are defined so that they can be accessed to
    get information:
            assertTrue(undefined <> privateUser1.email);
            assertTrue(undefined <> privateUser1.password);
            assertTrue(undefined <> privateUser1.myArtists);
            assertTrue(undefined <> privateUser1.myBands);
            assertTrue(undefined <> privateUser1.myPlaces);
            assertTrue(undefined <> privateUser1.myShows);
            assertTrue(undefined <> privateUser1.privateUsers);
        );


        /* Tests to methods of PrivateUser */
        -- Covering requirements R003, R005, R006, R007, R008 and R023.
        private testCreateArtist : () ==> ()
        testCreateArtist() == (
            -- This three artists are used on other tests.
            privateUser1.CreateArtist(artistName1,address1);
            privateUser1.CreateArtist(artistName2,address1);
            privateUser1.CreateArtist(artistName3,address1);

            -- Verifying that the variables are defined so that they can be accessed to
    get information:
            let a in set privateUser1.myArtists be st a.name = artistName1 in (
                assertTrue(undefined <> a.songs);
                assertTrue(undefined <> a.address);
                assertTrue(undefined <> a.genres);
                assertTrue(undefined <> a.instruments);
                assertTrue(undefined <> a.description);
                assertTrue(undefined <> a.memberOf);
                assertTrue(undefined <> a.artists);
            );
        )
```

```
	pre card privateUser1.myArtists = 0
	post card privateUser1.myArtists = 3;

	-- Covering requirements R009, R011, R012, R013, R014 and R023.
	private testCreateBand : () ==> ()
	testCreateBand() == (
		privateUser1.CreateBand(bandName1,address1,privateUser1.myArtists);
		-- The next line should make the test fail: the same artist in a set that 
must have at least two distict artists.
		--let artist1 in set Artist`artists in 
privateUser1.CreateBand(bandName2,address1,{artist1,artist1});

		-- Verifying that the variables are defined so that they can be accessed to 
get information:
		let b in set privateUser1.myBands be st b.name = bandName1 in (
			assertTrue(undefined <> b.songs);
			assertTrue(undefined <> b.address);
			assertTrue(undefined <> b.genres);
			assertTrue(undefined <> b.instruments);
			assertTrue(undefined <> b.description);
			assertTrue(undefined <> b.members);
			assertTrue(undefined <> b.bands);
		);
	)
	pre card privateUser1.myArtists = 3    -- This is the expected length, but it can 
have 2 or more elements.
		and card privateUser1.myBands = 0
	post card privateUser1.myBands = 1;

	-- Covering requirement R028.
	private testSetPrivateUserPassword : () ==> ()
	testSetPrivateUserPassword() == (
		dcl oldPass : PrivateUser`Password := privateUser1.password;
		dcl newPass : PrivateUser`Password := "123456789";
		privateUser1.SetPassword(newPass);
		assertEqual(newPass,privateUser1.password);
		-- The next line should make the test fail: password length lower than 8.
		--privateUser1.SetPassword(password2);
		privateUser1.SetPassword(oldPass);
	)
	post privateUser1~.password = privateUser1.password;

	-- Covering requirement R024, R025 and R026.
	private testCreateRemovePlace : () ==> ()
	testCreateRemovePlace() == (
		privateUser1.CreatePlace("FEUP","",address1);
		-- The next line should make the test fail: empty name.
		--privateUser1.CreatePlace("","",address1);

		-- Verifying that the variables are defined so that they can be accessed to 
get information:
		let p in set privateUser1.myPlaces in (
			assertTrue(undefined <> p.name);
			assertTrue(undefined <> p.description);
			assertTrue(undefined <> p.address);
			assertTrue(undefined <> p.places);
		);

		let p in set privateUser1.myPlaces in (
                privateUser1.RemovePlace(p);
                -- The next line should make the test fail: non-existent place.
                --privateUser1.RemovePlace(p);
            );
        )
        pre card privateUser1.myPlaces = 0
        post card privateUser1.myPlaces = 0;

        -- Covering requirements R016 and R017.
        private testCreateRemoveShow : () ==> ()
        testCreateRemoveShow() == (
            dcl date1 : Common`Date := Common`CreateDate(2016,2,29,0,0);
            dcl date2 : Common`Date := Common`CreateDate(2017,12,30,0,0);
            privateUser1.CreatePlace("FEUP","",address1);
            let place in set privateUser1.myPlaces in (
                dcl name1 : Common`String1 := "MFES show1";
                dcl name2 : Common`String1 := "MFES show2";

        privateUser1.CreateShow(name1,privateUser1.myArtists,date2,date2,place,"");

        privateUser1.CreateShow(name2,privateUser1.myBands,date1,date2,place,"");
                -- The next line should make the test fail: end date before start
    date.
                --let artist1, artist2 in set Artist`artists be st artist1 <> artist2
    in privateUser1.CreateShow("MFES show",{artist1,artist2},date2,date1,place,"");

                -- Verifying that the variables are defined so that they can be
    accessed to get information:
                let s in set privateUser1.myShows be st s.name = name1 in (
                    assertTrue(undefined <> s.name);
                    assertTrue(undefined <> s.performers);
                    assertTrue(undefined <> s.startDate);
                    assertTrue(undefined <> s.endDate);
                    assertTrue(undefined <> s.place);
                    assertTrue(undefined <> s.description);
                );

                privateUser1.RemovePlace(place);-- The place can still exist
    associated to shows, but it is not 'active' and will not appear anywhere else.
            );

            let show in set privateUser1.myShows in (
                privateUser1.RemoveShow(show);
                -- The next line should make the test fail: non-existent show.
                --privateUser1.RemoveShow(show);
            );
            let show in set privateUser1.myShows in privateUser1.RemoveShow(show);
        )
        pre card privateUser1.myArtists = 3    -- This is the expected length, but it can
    have 1 or more elements.
            and card privateUser1.myBands = 1
            and card privateUser1.myPlaces = 0
            and card privateUser1.myShows = 0
        post card privateUser1.myPlaces = 0 and card privateUser1.myShows = 0;

        -- Covering requirement R002.
        private testGetPrivateUser : () ==> ()
        testGetPrivateUser() == (
            dcl privUser1 : [PrivateUser] :=
PrivateUser`GetPrivateUser(email1,password1);
            dcl privUser2 : [PrivateUser] :=
PrivateUser`GetPrivateUser(email2,password1);
            dcl privUser3 : [PrivateUser] :=
PrivateUser`GetPrivateUser(email4,password1);
            assertEqual(privUser1,privateUser1);
            assertEqual(privUser2,privateUser2);
            assertEqual(privUser3,nil);     -- Non-existent private user.
      );


      /* Tests to methods of User */
      -- Covering requirements R003, R009 and R028.
      private testSetUserName : () ==> ()
      testSetUserName() == (
            dcl oldName : Common`String1 := privateUser1.name;
            privateUser1.SetName(username2);
            assertEqual(username2,privateUser1.name);
            privateUser1.SetName(oldName);
      )
      pre privateUser1.name <> username2
      post privateUser1~.name = privateUser1.name;

      -- Covering requirements R018, R019 and R020.
      private testAddRemoveConversation : () ==> ()
      testAddRemoveConversation() == (
            dcl members : Conversation`Users := {privateUser1};
            dcl c1 : Conversation := new Conversation("Test conversation",members);
            let c2 in set privateUser1.conversations in (
                  assertEqual(c1,c2);
                  assertEqual(c2.members,members);
            );

            let c in set privateUser1.conversations in
privateUser1.RemoveConversation(c);
      )
      pre card privateUser1.conversations = 0
      post card privateUser1.conversations = 0;

      /* Tests to methods of Conversation */
      -- Covering requirement R022.
      private testAddRemoveConversationMember : () ==> ()
      testAddRemoveConversationMember() == (
            dcl topic1 : Common`String1 := "Topic";
            dcl c1 : Conversation := new Conversation(topic1,{privateUser1});
            c1.AddMember(privateUser2);
            assertEqual(c1.members,{privateUser1,privateUser2});
            let c in set privateUser1.conversations in
privateUser1.RemoveConversation(c);
            let c in set privateUser2.conversations in
privateUser2.RemoveConversation(c);
      )
      pre card privateUser1.conversations = 0 and card privateUser2.conversations = 0
      post card privateUser1.conversations = 0 and card privateUser2.conversations = 0;

      -- Covering requirement R022.
      private testSetConversationTopic : () ==> ()
      testSetConversationTopic() == (
            dcl topic1 : Common`String1 := "Topic1";
            dcl topic2 : Common`String1 := "Topic2";
            dcl c1 : Conversation := new Conversation(topic1,{privateUser1});
            c1.SetTopic(topic2);
            assertEqual(topic2,c1.topic);
            let c in set privateUser1.conversations in
privateUser1.RemoveConversation(c);
        )
        pre card privateUser1.conversations = 0
        post card privateUser1.conversations = 0;

        -- Covering requirements R018, R019, R020 and R021.
        private testAddMessageToConversation : () ==> ()
        testAddMessageToConversation() == (
            dcl date : Common`Date := Common`CreateDate(2017,12,30,0,0);
            dcl text : Common`String1 := "Text";
            dcl c1 : Conversation := new Conversation("Topic",{privateUser1});
            c1.AddMessage(date,text,privateUser1);
            let m1 in set rng c1.messages in assertTrue(m1.date = date and m1.text =
text and m1.sender = privateUser1);
            let c in set privateUser1.conversations in
privateUser1.RemoveConversation(c);
        )
        pre card privateUser1.conversations = 0
        post card privateUser1.conversations = 0;

        /* Tests to methods of PublicUser */
        -- Covering requirements R003 and R009.
        private testAddRemovePublicUserSong : () ==> ()
        testAddRemovePublicUserSong() == (
            let artist in set privateUser1.myArtists in (
                dcl name1 : Common`String1 := "Song1";
                dcl link1 : Common`String1 :=
"https://www.youtube.com/watch?v=2tqQcIBhSOE";
                dcl name2 : Common`String1 := "Song2";
                dcl link2 : Common`String1 := "https://www.youtube.com/watch?v=Kq-
DsCRVma0";
                artist.AddSong(name1,link1);
                let s1 in set artist.songs in (
                    assertTrue(s1.name = name1 and s1.link = link1);
                    artist.AddSong(name2,link2);     -- Will test if 'link2' has
never been used (precondition).
                    -- The next line should make the test fail: link already used
for other song.
                    --artist.AddSong(name1,link1);
                    artist.RemoveSong(s1);
                    -- The next line should make the test fail: non-existent song.
                    --artist.RemoveSong(s1);
                    let s2 in set artist.songs be st (s2.name = name2 and s2.link
= link2) in artist.RemoveSong(s2);
                );
            );
        )
        pre forall artist in set privateUser1.myArtists & card artist.songs = 0
        post forall artist in set privateUser1.myArtists & card artist.songs = 0;

        -- Covering requirements R003 and R009.
        private testSetPublicUserAddress : () ==> ()
        testSetPublicUserAddress() == (
		let artist in set privateUser1.myArtists in (
			dcl oldAddress : Common`Address := artist.address;
			assertTrue(oldAddress <> address2);
			artist.SetAddress(address2);
			assertEqual(artist.address,address2);
			artist.SetAddress(oldAddress);
		);
	);

	-- Covering requirements R003 and R009.
	private testAddRemovePublicUserGenre : () ==> ()
	testAddRemovePublicUserGenre() == (
		let artist in set privateUser1.myArtists in (
			dcl testGenre : Artist`Genre := "Country";
			artist.AddGenre(testGenre);
			let g in set artist.genres in (
				assertEqual(g,testGenre);
				-- The next line should make the test fail: genre already 
added.
				--artist.AddGenre(testGenre);
				artist.RemoveGenre(g);
				-- The next line should make the test fail: non-existent 
genre.
				--artist.RemoveGenre(g);
			);
		);
	)
	pre forall artist in set privateUser1.myArtists & card artist.genres = 0
	post forall artist in set privateUser1.myArtists & card artist.genres = 0;

	-- Covering requirements R003 and R009.
	private testAddRemovePublicUserInstrument : () ==> ()
	testAddRemovePublicUserInstrument() == (
		let artist in set privateUser1.myArtists in (
			dcl testInstrument : Artist`Genre := "Flute";
			artist.AddInstrument(testInstrument);
			let i in set artist.instruments in (
				assertEqual(i,testInstrument);
				-- The next line should make the test fail: link already used 
for other instrument.
				--artist.AddInstrument(testInstrument);
				artist.RemoveInstrument(i);
				-- The next line should make the test fail: non-existent 
instrument.
				--artist.RemoveInstrument(i);
			);
		);
	)
	pre forall artist in set privateUser1.myArtists & card artist.instruments = 0
	post forall artist in set privateUser1.myArtists & card artist.instruments = 0;

	-- Covering requirements R003 and R009.
	private testSetPublicUserDescription : () ==> ()
	testSetPublicUserDescription() == (
		let artist in set privateUser1.myArtists in (
			dcl oldDescription : Common`String := artist.description;
			dcl newDescription : Common`String := "Test description";
			assertTrue(oldDescription <> newDescription);
			artist.SetDescription(newDescription);
```

```
                assertEqual(artist.description,newDescription);
                artist.SetDescription(oldDescription);
            );
        );

        -- Covering requirements R004 and R010.
        private testGetPublicUsers : () ==> ()
        testGetPublicUsers() == (
            privateUser1.CreateArtist(specialArtistName1,address1);
            privateUser1.CreateArtist(specialArtistName2,address1);
            privateUser1.CreateArtist(specialArtistName3,address1);
            assertTrue(exists a1,a2,a3 in set
Artist`GetPublicUsers(specialArtistSubstring,Artist`artists) & (a1 <> a2 and a1 <> a3
and a2 <> a3)
                    and a1.name = specialArtistName1 and a2.name = specialArtistName2
and a3.name = specialArtistName3);
        );


        /* Tests to methods of Place */
        -- Covering requirement R025.
        private testSetPlaceName : () ==> ()
        testSetPlaceName() == (
            privateUser1.CreatePlace("FEUP","",address1);
            let place in set privateUser1.myPlaces in (
                dcl oldName : Common`String1 := place.name;
                place.SetName("FEUP2");
                assertTrue(place.name <> oldName);
            );
            let place in set privateUser1.myPlaces in privateUser1.RemovePlace(place);
        )
        pre card privateUser1.myPlaces = 0
        post card privateUser1.myPlaces = 0;

        -- Covering requirement R025.
        private testSetPlaceDescription: () ==> ()
        testSetPlaceDescription() == (
            privateUser1.CreatePlace("FEUP","",address1);
            let place in set privateUser1.myPlaces in (
                dcl oldDescription : Common`String := place.description;
                place.SetDescription("New description");
                assertTrue(place.description <> oldDescription);
            );
            let place in set privateUser1.myPlaces in privateUser1.RemovePlace(place);
        )
        pre card privateUser1.myPlaces = 0
        post card privateUser1.myPlaces = 0;


        /* Tests to methods of Calendar */
        -- Covering requirements R015 and R017.
        private testGetShowsSortedByStartDate : () ==> ()
        testGetShowsSortedByStartDate() == (
            testGetShowsSortedByStartDateAux1();
            testGetShowsSortedByStartDateAux2();
        )
        pre card privateUser1.myPlaces = 0 and card privateUser1.myShows = 0
        post card privateUser1.myPlaces = 0 and card privateUser1.myShows = 0;
```

```
	private testGetShowsSortedByStartDateAux1 : () ==> ()
	testGetShowsSortedByStartDateAux1() == (
		dcl date1 : Common`Date := Common`CreateDate(2016,2,29,0,0);
		dcl date2 : Common`Date := Common`CreateDate(2017,12,30,0,0);
		dcl date3 : Common`Date := Common`CreateDate(2017,12,31,0,0);
		privateUser1.CreatePlace("FEUP","",address1);
		let place in set privateUser1.myPlaces in (

	privateUser1.CreateShow("Show1",privateUser1.myBands,date3,date3,place,"");

	privateUser1.CreateShow("Show2",privateUser1.myArtists,date2,date2,place,"");

	privateUser1.CreateShow("Show3",privateUser1.myBands,date1,date2,place,"");
		);
	);

	private testGetShowsSortedByStartDateAux2 : () ==> ()
	testGetShowsSortedByStartDateAux2() == (
		dcl shows : Calendar`Shows := Calendar`GetShowsSortedByStartDate();
		for i=1 to len shows do
			for j=i+1 to len shows do
				assertTrue(Common`isBeforeOrEqual(shows(i).startDate,
shows(j).startDate));

		while(card privateUser1.myPlaces > 0) do
			let place in set privateUser1.myPlaces in
privateUser1.RemovePlace(place);

		while(card privateUser1.myShows > 0) do
			let show in set privateUser1.myShows in
privateUser1.RemoveShow(show);
	);


	/* Tests to methods of Common */
	private testCreateDate : () ==> ()
	testCreateDate() == (
		dcl dates : set of Common`Date := {};
		dates := dates union {Common`CreateDate(2016,2,29,0,0)};
		dates := dates union {Common`CreateDate(2017,2,28,23,0)};
		dates := dates union {Common`CreateDate(2016,11,30,23,0)};
		dates := dates union {Common`CreateDate(2016,12,31,23,59)};
		-- The next line should make the test fail: not leap year.
		--dates := dates union {Common`CreateDate(2017,2,29,23,0)};
		-- The next line should make the test fail: month 11 does not have 31 days.
		--dates := dates union {Common`CreateDate(2016,11,31,23,0)};
		-- The next line should make the test fail: there is no month with 32 days.
		--dates := dates union {Common`CreateDate(2016,12,32,23,0)};
		-- The next line should make the test fail: hour 24 does not exist.
		--dates := dates union {Common`CreateDate(2016,11,30,24,0)};
		-- The next line should make the test fail: minute 60 does not exist.
		--dates := dates union {Common`CreateDate(2016,11,30,23,60)};
		assertEqual(dates,dates);
	);

functions
traces
end JyveTest
```

Function or operation	Line	Coverage	Calls
JyveTest	129	100%	1
main	35	100%	1
testAddMemberToConversation	258	100%	2
testAddMessagerToConversation	283	100%	1
testAddRemoveConversation	243	100%	1
testAddRemoveConversationMember	262	100%	1
testAddRemoveMemberToConversation	262	100%	2
testAddRemovePublicUserGenre	332	100%	1
testAddRemovePublicUserInstrument	351	100%	1
testAddRemovePublicUserSong	296	100%	1
testCreateArtist	143	100%	1
testCreateBand	154	100%	2
testCreateDate	462	100%	1
testCreateRemovePlace	177	100%	13
testCreateRemoveShow	193	100%	2
testGetPrivateUser	219	100%	1
testGetPublicUsers	383	100%	12
testGetShowsSortedByStartDate	425	100%	1
testGetShowsSortedByStartDateAux1	433	100%	1
testGetShowsSortedByStartDateAux2	446	100%	3
testPrivateUserVariables	233	100%	1
testPublicUserVariables	410	100%	2
testSetConversationTopic	270	100%	1
testSetPlaceDesciption	409	100%	1
testSetPlaceName	395	100%	1
testSetPrivateUserPassword	164	100%	2
testSetPublicUserAddress	320	100%	1
testSetPublicUserDescription	370	100%	1
testSetUserName	232	100%	2
JyveTest.vdmpp		100%	61

5. Model verification

5.1 Example of domain verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
35	Conversation`AddMessage	legal map application

The code under analysis (with the relevant domain verification underlined) is:

```
public AddMessage : Common`Date * Common`String1 * User ==> ()
AddMessage(d,t,s) == (
        dcl msg : Message := mk_Message(d,t,s);
        dcl myMap : Messages := { card dom messages + 1 |-> msg };
        messages := messages munion myMap;
)
pre s in set members
```

```
	post card dom messages = card dom messages~ + 1;
```

In this case the proof is trivial because the verification '`pre s in set members`' assures that the sender is a member of the conversation, assuring that a message added is from a valid user, i.e. a user that is part of the domain 'members'.

Proof obligation generated by the tool:

```
(forall d:Common`Date, t:Common`String1, s:User & ((s in set members) => is_((messages
munion myMap), inmap (nat1) to (Conversation`Message))))
```

5.2 Example of invariant verification

Another proof obligation generated by Overture is:

No.	PO Name	Type
122	PublicUser`AddSong	state invariant holds

The code under analysis (with the relevant changes underlined) is:

```
	public AddSong : Common`String1 * Common`String1 ==> ()
	AddSong(n,l) == (
		dcl s : Song := mk_Song(n,l);
		atomic(
			songs := songs union {s};
			allSongs := allSongs union {s};
		);
	)
	pre forall s1 in set songs & s1.link <> l     -- Double check relatively to
'allSongs'.
		and forall s2 in set allSongs & s2.link <> l
	post exists1 s1 in set songs & s1.name = n and s1.link = l
		and exists1 s2 in set allSongs & s2.name = n and s2.link = l
		and s1 = s2;
```

The relevant invariant under analysis is:

```
   inv forall s in set songs & s in set allSongs;
```

After the execution of the '`atomic`' block we have (technically, this is the post-condition of the block):

```
	exists1 s1 in set songs & s1.name = n and s1.link = l
		and exists1 s2 in set allSongs & s2.name = n and s2.link = l
		and s1 = s2;
```

We have to prove that this implies that the invariant holds, i.e., that the following condition holds:

```
	(exists1 s1 in set songs & s1.name = n and s1.link = l
		and exists1 s2 in set allSongs & s2.name = n and s2.link = l
		and s1 = s2)  =>  forall s in set songs & s in set allSongs)
```

Looking at the post-condition, we can verify that s1 and s2 have the same attributes (name and link) and are the same object, so, if the post-condition is true, the object is part of both sets. Following this reasoning, if the invariant wasn't broken before entering on the function, the invariant still holds by consequence of the post-condition being true for the new input.

Proof obligation generated by the tool:

```
(forall n:Common`String1, l:Common`String1 & ((forall s1 in set songs & (((s1.link) <>
l) and (forall s2 in set allSongs & ((s2.link) <> l)))) => (exists1 s1 in set songs &
(((s1.name) = n) and (((s1.link) = l) and (exists1 s2 in set allSongs & (((s2.name) = n)
and (((s2.link) = l) and (s1 = s2)))))))))
```

6. Code generation

To generate Java code from the VDM++ model, we used the code generation tool from Overture. First of all, a new run configuration with a console launch mode was added. Then we used the option Code Generation -> Generate Java (Launch configuration Based) and chose the created configuration. At the end, 11 files were generated (one for each vdmpp file) with all data and functionality of original files.

The generated code was tested during the development of the interface, using the options available in the interface, and no problems have been found.

We've found that the tool generates some additional methods and functions and we've found that part of the code generated can be a little bit hard to read due to the names generated by the generator and because Overture doesn't allow the use of "self.var" to access a class variable, being necessary to give names different than "var" to the arguments of the functions. However, the work did not become much more complicated due to this.

Also, we had problems with the code generated to verify if an email had the correct syntax, being necessary to remove the VDM++ code that was responsible for the verification. The Java generator was trying to use an iterator to traverse the object of type String, assuming that the string was an object of type Vector, and because the class String does not provide the method iterator, the code was not compilable.

7. Conclusions

The model that was developed covers all the requirements, but, because there is always space to do a better work, it could be a good idea to add more functionality to project.

As future work, it would be useful to better test the variables that are accessed to get information (e.g. those used to show the profile of an artist).

This project took approximately 75 hours to develop.

The division of effort and contributions among team members has been 75% for Luís and 25% for Ilona.

In general, we became more aware of how to implement a program with a low risk of failure, how difficult it is and that there are good tools to support this type of development.

8. References

1. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
2. Overture tool web site, http://overturetool.org
3. Resources about VDM++ and Overture available in Moodle
4. Selection sort algorithm present on the web site https://www.geeksforgeeks.org/selection-sort/