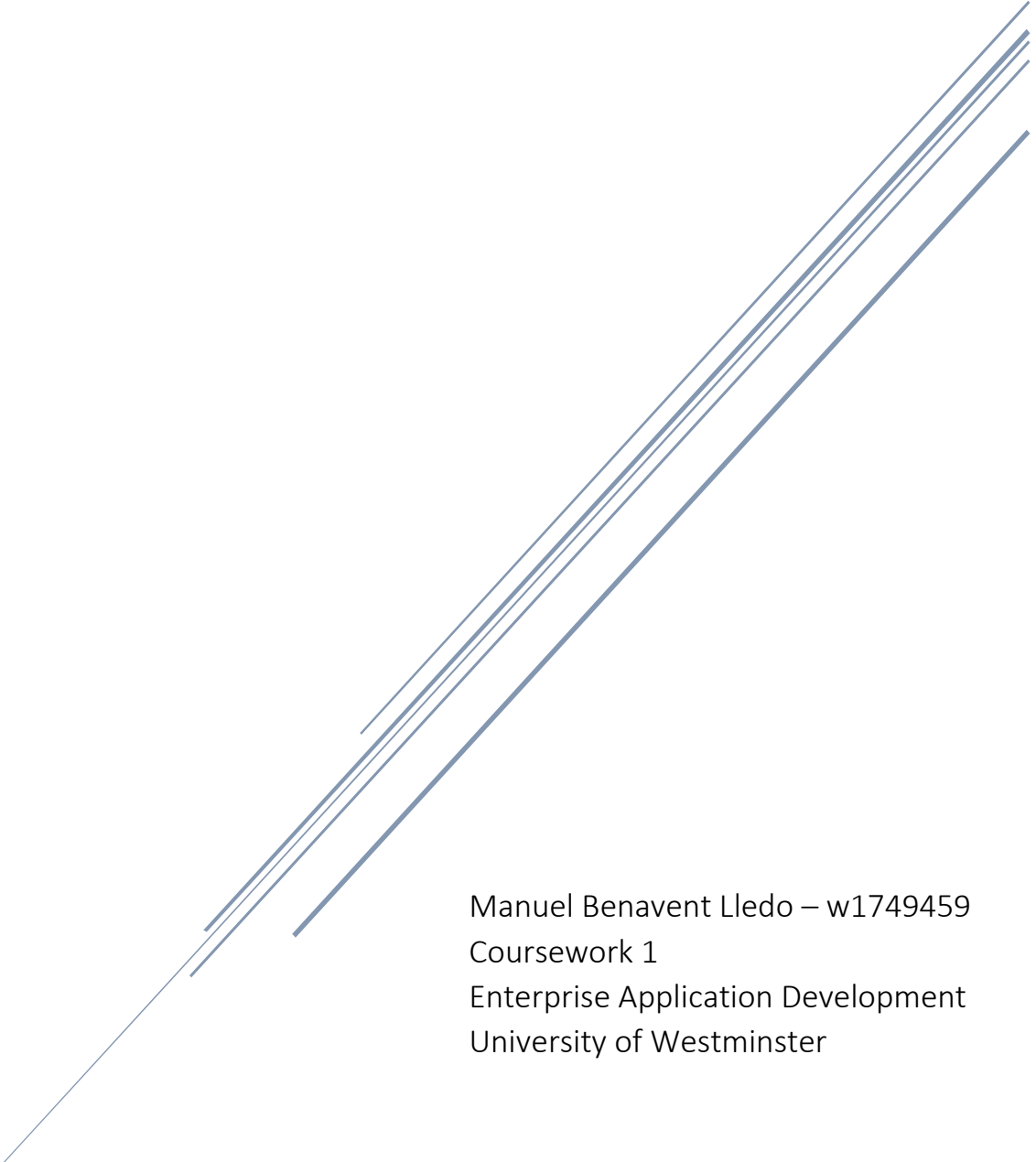


# COURSEWORK 1

## Enterprise Application Development



Manuel Benavent Lledo – w1749459  
Coursework 1  
Enterprise Application Development  
University of Westminster

## Index

1. Part A – Requirements .....	2
2. Part B – Use Case Diagrams.....	4
3. Part C – Classes.....	8
1) CRC table .....	8
2) Domain model.....	10
4. Part D – Sequence diagrams .....	12
5. Part E – Activity Diagram.....	14
6. Part F – Reports.....	15

## 1. Part A – Requirements

These are the requirements needed for building the time management and reporting tool for personal use:

### **R1. The software shall allow the user to insert a new contact and edit the existing ones.**

- R1.1. The information about the contact will be stored in a data base
- R1.2. The contact must have a first and last name and the system must validate it, that is, it can be composed only by alphabetic characters.
- R1.3. The contact may have an email address, if so, the software will validate it.
- R1.4. The software will optionally allow the user to add a telephone number, it will check that is only composed by numbers (no more validations will be required since each country has a different format).
- R1.5. The user will be able to delete contacts.

### **R2. The software shall allow the user to create (or edit existing) events.**

- R2.1. All events must have a name.
- R2.2. All events must have a start and an end date and time.
- R2.3. Events may be one-off or recurring. If the event is recurring, the user will have to introduce the number of days, weeks, months or years that it will be recurring for.
- R2.4. There will be different types of events: appointments, tasks, lectures and tutorials.
  - R2.4.1. Tasks can be marked as completed and, in that case, they will be shown 'greyed out' in the calendar.
  - R2.4.2. Lectures will have a field to introduce the lecturer.
  - R2.4.3. Tutorial will have a field to introduce the lecturer and the lab number.
- R2.5. These events will be stored in the database.
- R2.6. An event may have one or more contacts associated.
- R2.7. An event may have a location.
  - R2.7.1. A location must have a name, address line, postcode (that will be verified), a city or town and a country. And optionally a second address line.
- R2.8. The user will be able to delete an event and if it is recurring, he will be asked if he wants to delete one or all the future ones.

### **R3. The software shall display the user interface with the following options:**

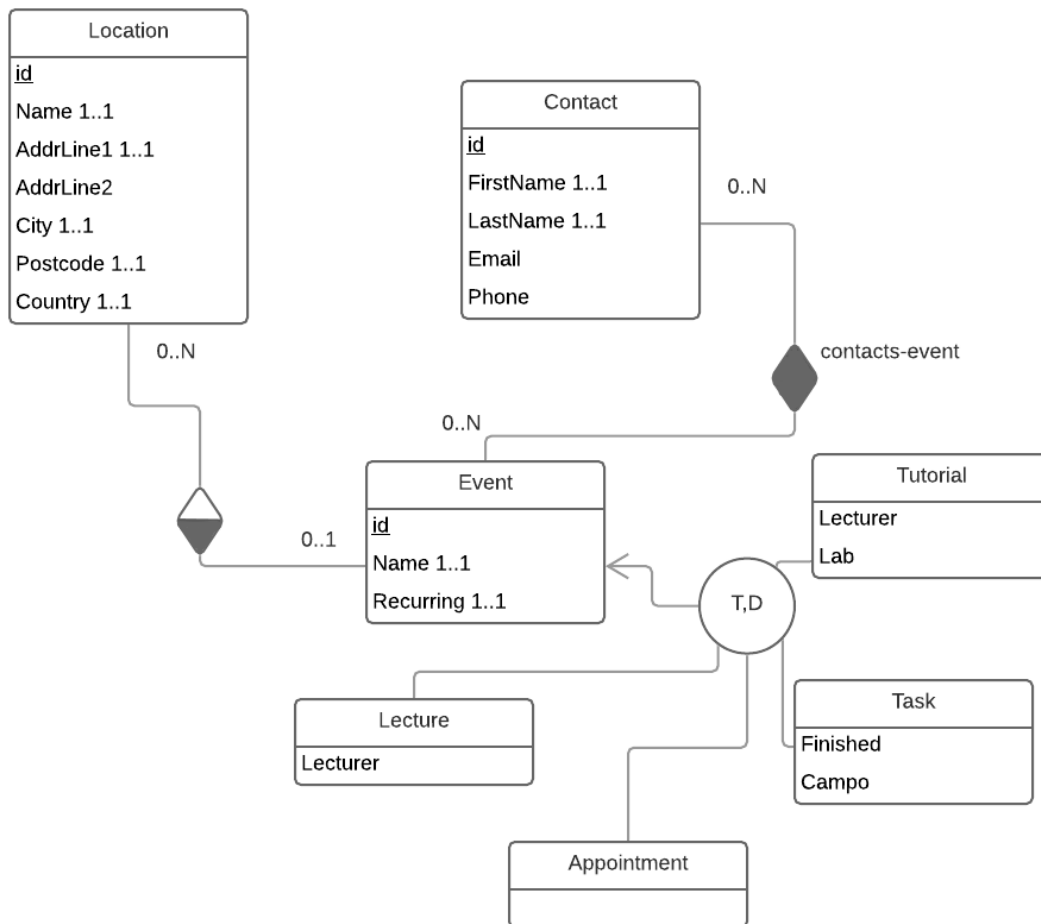
- R3.1. The user interface will allow the user to perform the operations above mentioned.
- R3.2. The system will provide the user with different views of the calendar: weekly and schedule.

R3.3. The system will provide an option to show a list of the contacts and an option to edit each of them.

R3.4. The system will provide the user with a time usage report for the next 4 weeks based on the previous weeks' time usage.

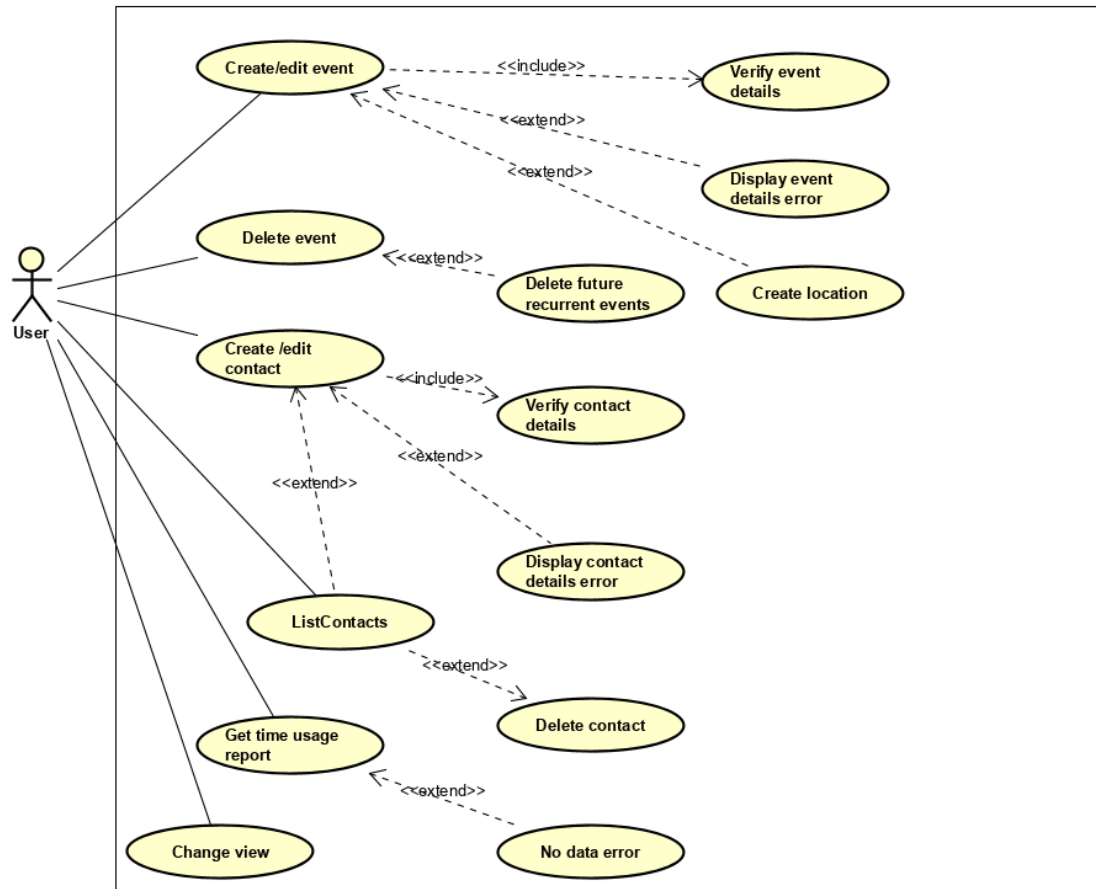
**R4. The system will use a DDBB as a form of storing data in a persistent way.**

R4.1. The DDBB scheme will be as follows:



## 2. Part B – Use Case Diagrams

The designed system has the following use case diagram:



The uses cases shown above have the following descriptions:

### **Use Case: Create/edit event**

**Id:** UC-001

**Description:** The user wants to create or edit an event.

**Primary Actor:** User (the user of the system).

### **Stakeholders and interests**

Verify event details will appear as soon as the user tries to save the information and display event details error will appear if they were not correct.

**Pre-conditions:** none for creating an event, for updating the event must be created.

**Post-conditions:** Event has been created/updated.

**Trigger:** Updates the information in the current main view (weekly or schedule).

### **Main Success Scenario:**

1. User selects add event or selects an event to be edited.
2. User enter event details (name, date and time, location and contacts).

3. Event details verification is correct.
4. Event has been created.

#### Variations

If the event details are not correct an exception will be thrown, and the user will have to correct the details.

#### **Use Case: Delete event**

Id: UC-002

Description: The user wants to delete an event.

Primary Actor: User (the user of the system).

Pre-conditions: there must be an event

Post-conditions: Event has been deleted.

Trigger: Updates the information in the current main view (weekly or schedule).

#### Main Success Scenario:

1. User selects an event to be removed.
2. User confirms that he wants to delete.
3. Event has been removed.

#### **Use Case: Create/edit contact**

Id: UC-003

Description: The user wants to create or edit a contact.

Primary Actor: User (the user of the system).

#### Stakeholders and interests

Verify event details will appear as soon as the user tries to save the information and display contact details error will appear if they were not correct.

Pre-conditions: none for creation, for updating there must be a contact.

Post-conditions: Contact has been created/updated.

#### Main Success Scenario:

1. User selects add contact or selects a contact to be edited.
2. User enter contact details (first name, last name and, optionally, email and phone number).
3. Contact details verification is correct.
4. Contact has been created.

#### Variations

If the contact details are not correct an exception will be thrown, and the user will have to correct the details.

**Use Case: Delete contact**

Id: UC-004

Description: The user wants to delete a contact.

Primary Actor: User (the user of the system).

Pre-conditions: there must be a contact.

Post-conditions: Contact has been deleted.

Main Success Scenario:

1. User selects a contact to be removed.
2. User confirms that he wants to remove the contact.
3. Contact has been removed.

**Use Case: ListContacts**

Id: UC-005

Description: The user wants to see all the contacts stored in the data base.

Primary Actor: User (the user of the system).

Pre-conditions: None, if there are no contacts there will be a message warning about it.

Post-conditions: The user might return to the main view or might want to get more information about a contact, delete or edit it.

Main Success Scenario:

1. User clicks on the list contacts option.
2. Contacts are displayed in a new window.
3. User can close the window or edit/delete contacts.

**Use Case: GetTimeUsageReport**

Id: UC-006

Description: The user wants to get the time usage prediction for the following 4 weeks.

Primary Actor: User (the user of the system).

Main Success Scenario:

1. User selects the option to get the time usage report
2. The system calculates the prediction using the algorithm described in the activity diagram in part E.
3. A graph will be displayed with detailed information about the time prediction

Variations:

If there are not any event in the data base an exception will be thrown, and an error message will be displayed.

**Use Case: ChangeView**

Id: UC-007

Description: The user wants to change from the weekly view to the schedule view or vice versa.

Primary Actor: User (the user of the system).

Main Success Scenario:

1. User selects to change view
2. Current view is closed
3. New view is displayed.



### 3. Part C – Classes

#### 1) CRC table

Class Name	Type	Responsibility	Collaborations
<b>Event</b>	Model	Abstract class that has the main features for an event	EventView, EventEditView and its subclasses
<b>Task</b>	Model	Type of event	DAC_Task and super class collaborations
<b>Appointment</b>	Model	Type of event	DAC_Appointment and super class collaborations
<b>Lecture</b>	Model	Type of event	DAC_Lecture and super class collaborations
<b>Tutorial</b>	Model	Type of event	DAC_Tutorial and super class collaborations
<b>DAC_Task</b>	Model	Connects to the DDBB and obtains the desired information	Task
<b>DAC_Appointment</b>	Model	Connects to the DDBB and obtains the information	Appointment
<b>DAC_Lecture</b>	Model	Connects to the DDBB and obtains the information	Lecture
<b>DAC_Tutorial</b>	Model	Connects to the DDBB and obtains the information	Tutorial
<b>EventView</b>	View and controller (Form*)	Display detailed information for an event	Event and its subclasses and EventEditView
<b>EventEditView</b>	View and controller (Form*)	View displayed when creating or editing events	Event and its subclasses
<b>Location</b>	Model	Stores the information for a location	DAC_Location, Event
<b>DAC_Location</b>	Model	Stores the information of a location in the DDBB	Location
<b>LocationEditView</b>	View and controller (Form*)	View displayed when creating a location	Location, EventEditView

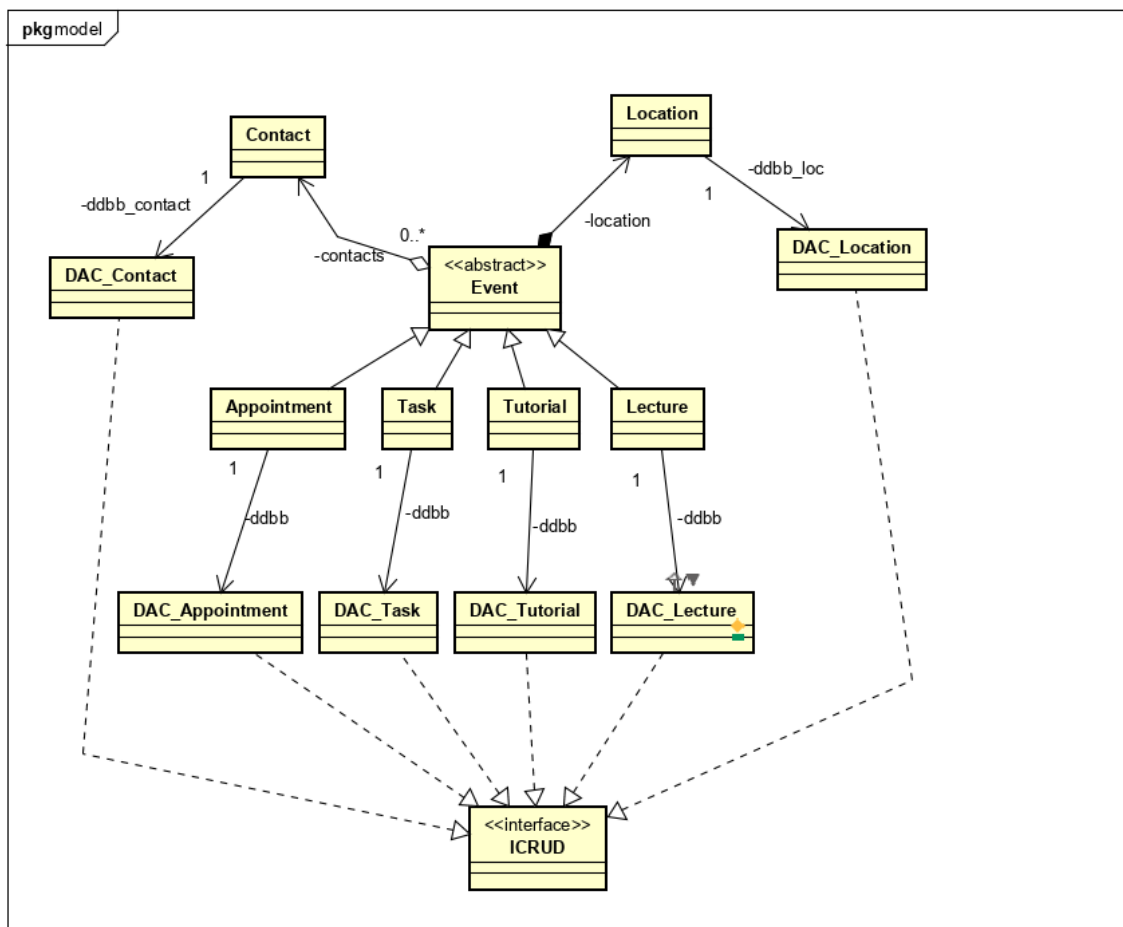
<b>Contact</b>	Model	Stores the information for contacts	Event, EventEditView, DAC_Contact, ContactView, ContactEditView and ListContactView
<b>DAC_Contact</b>	Model	Connects to the DDBB and obtains the information	Contact
<b>ContactView</b>	View and controller (Form*)	Displays detailed information for a contact	Contact, ContactEditView
<b>ContactEditView</b>	View and controller (Form*)	View displayed when creating or editing contacts	Contact
<b>ListContactView</b>	View and controller (Form*)	Displays a list with all the contacts so the user can view, edit or delete.	Contact, ContactView
<b>WeeklyView</b>	View and controller (Form*)	Displays a weekly view with all the events belonging to that week and the user is able to perform different operations from this view since it's the main view	Event, Contact, ListContactView, ScheduleView, ReportView, EventView
<b>ScheduleView</b>	View and controller (Form*)	Alternative view for the main one, it gives us the same options but the events are displayed in a schedule mode	Event, Contact, ListContactView, WeeklyView, ReportView, EventView
<b>ReportView</b>	View and controller (Form*)	Displays the time usage prediction report window	Event
<b>WrongNameException</b>	Model	Exception thrown by the model when the introduced name is not correct	Contact
<b>WrongEmailException</b>	Model	Exception thrown by the model when the email format is not correct	Contact

<b>WrongPhoneException</b>	Model	Exception thrown by the model when the phone number format is not correct	Contact
<b>WrongPostCodeException</b>	Model	Exception thrown by the model when the post code format is not correct	Location
<b>NoDataException</b>	Model	Exception thrown by the model when we try to calculate the time usage prediction and there is no data	Event

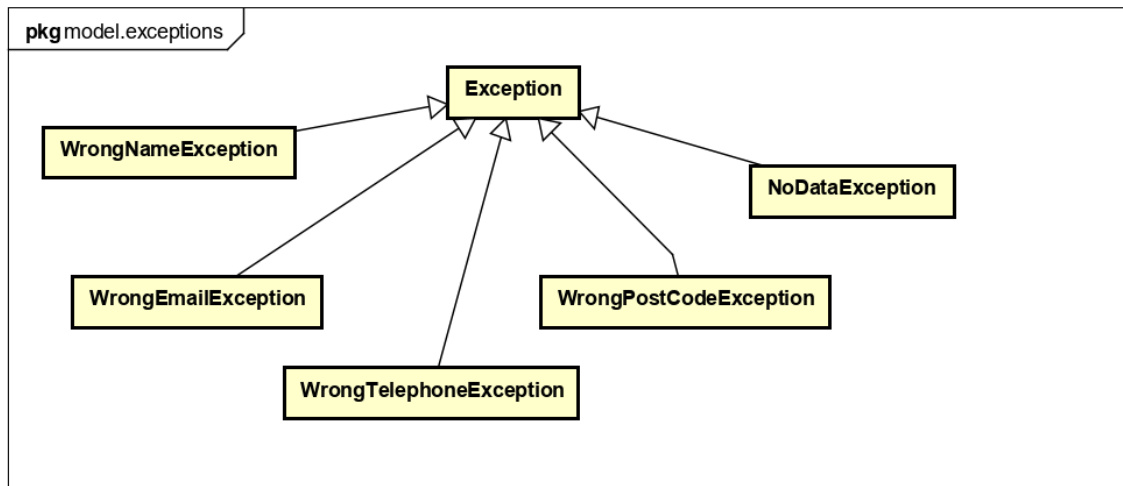
\*Form: In Windows forms, Visual Studio will create 2 different files (.cs and designer.cs) belonging to the same class but on separated files fulfilling the MVC requirements.

## 2) Domain model

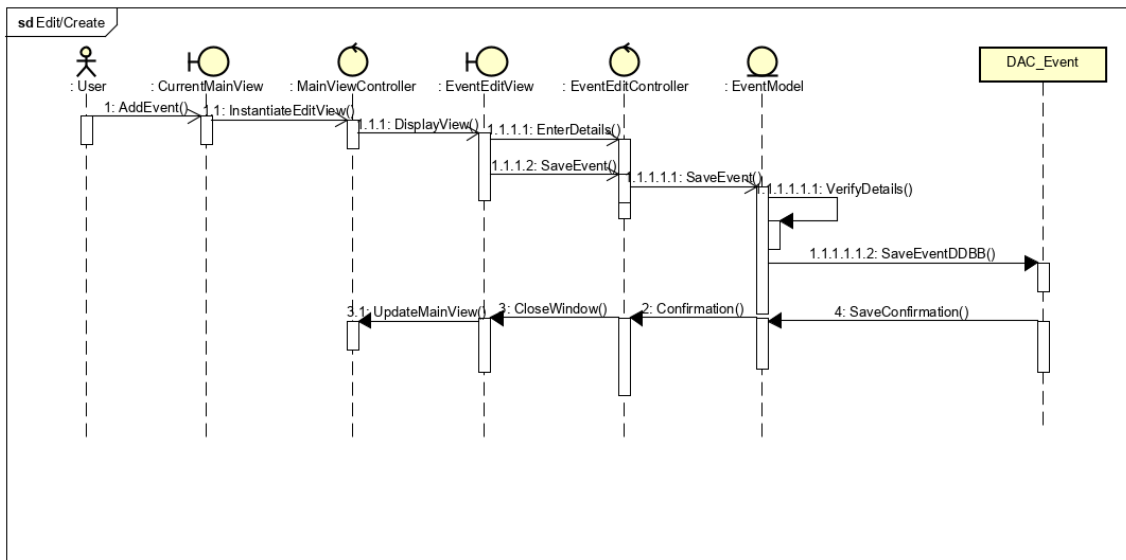
The system will be using the following domain model:



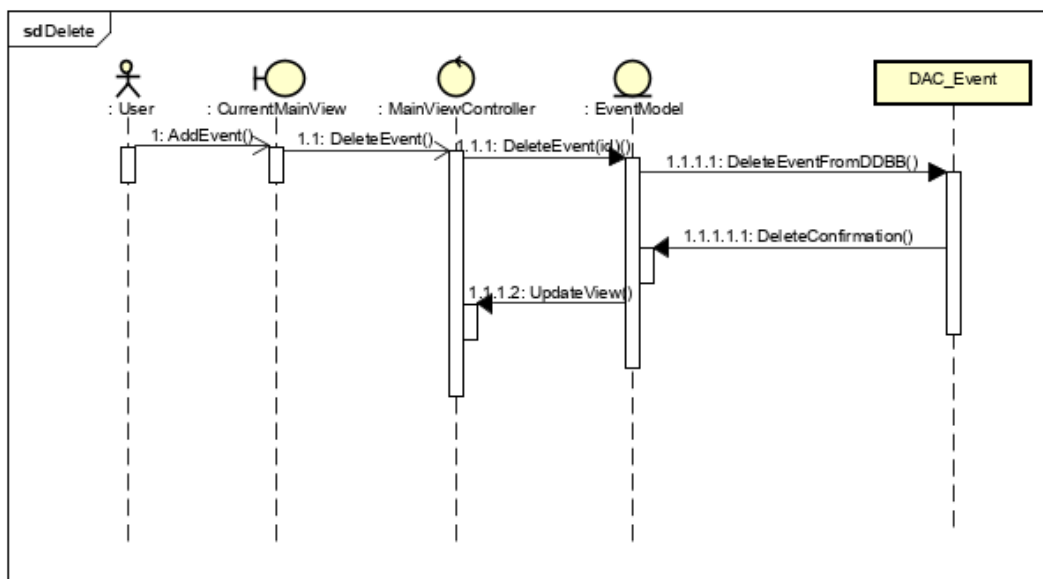
The classes that implement part of the view or the controller are not included in the domain model. However, since some of these classes will be able to throw different types of exceptions there is a domain model for the exceptions:



## 4. Part D – Sequence diagrams

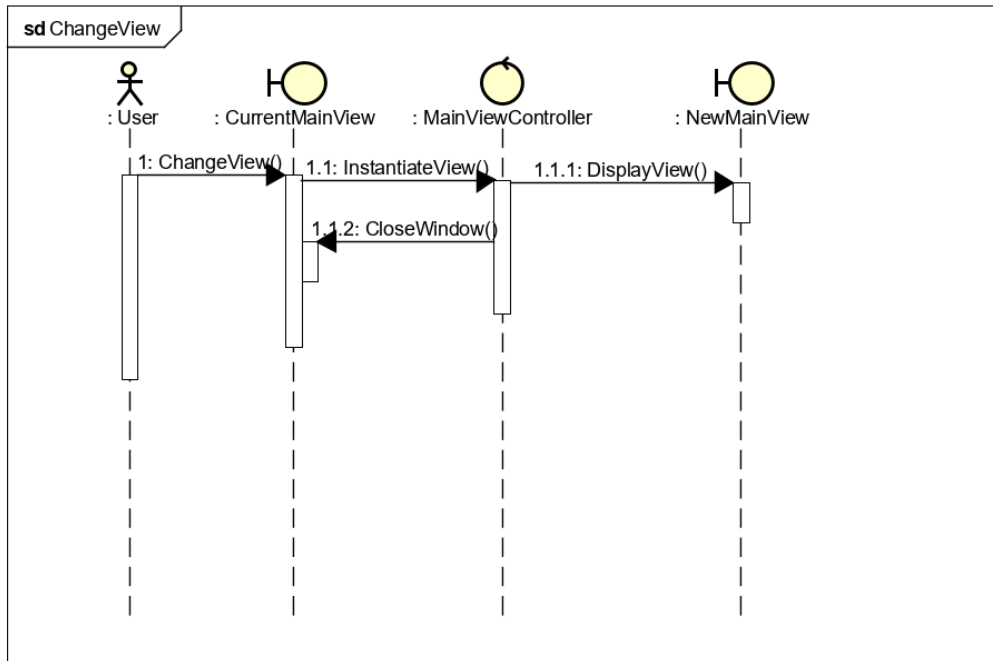


This sequence diagram shows the message exchange between the different parts of the system when creating a new event. The same process is followed when creating a contact but with its classes.

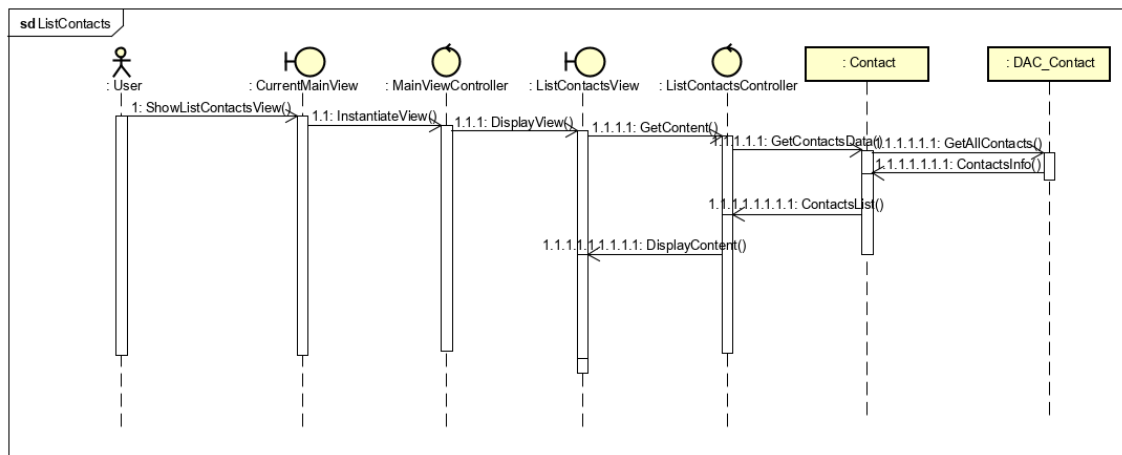


This second diagram shows the messages when trying to delete an event, as in the previous case it works for deleting a contact too.

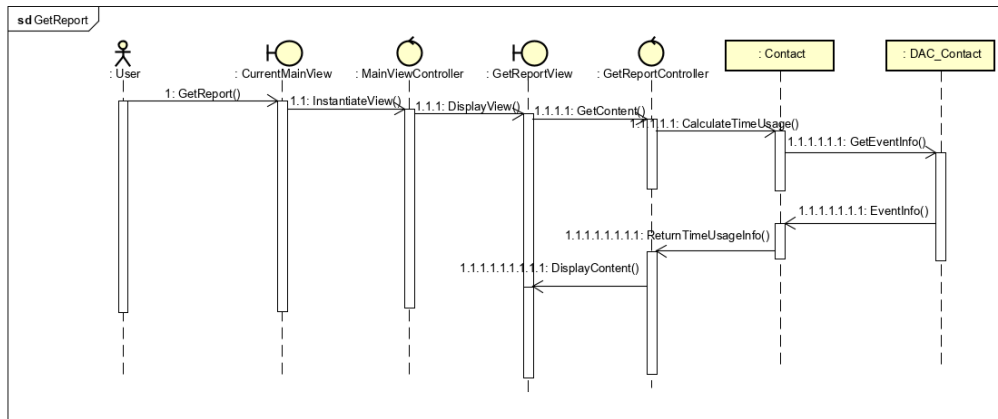
This is the sequence diagram when the user want to change the main view:



This is the message exchange between classes when displaying the contact list:

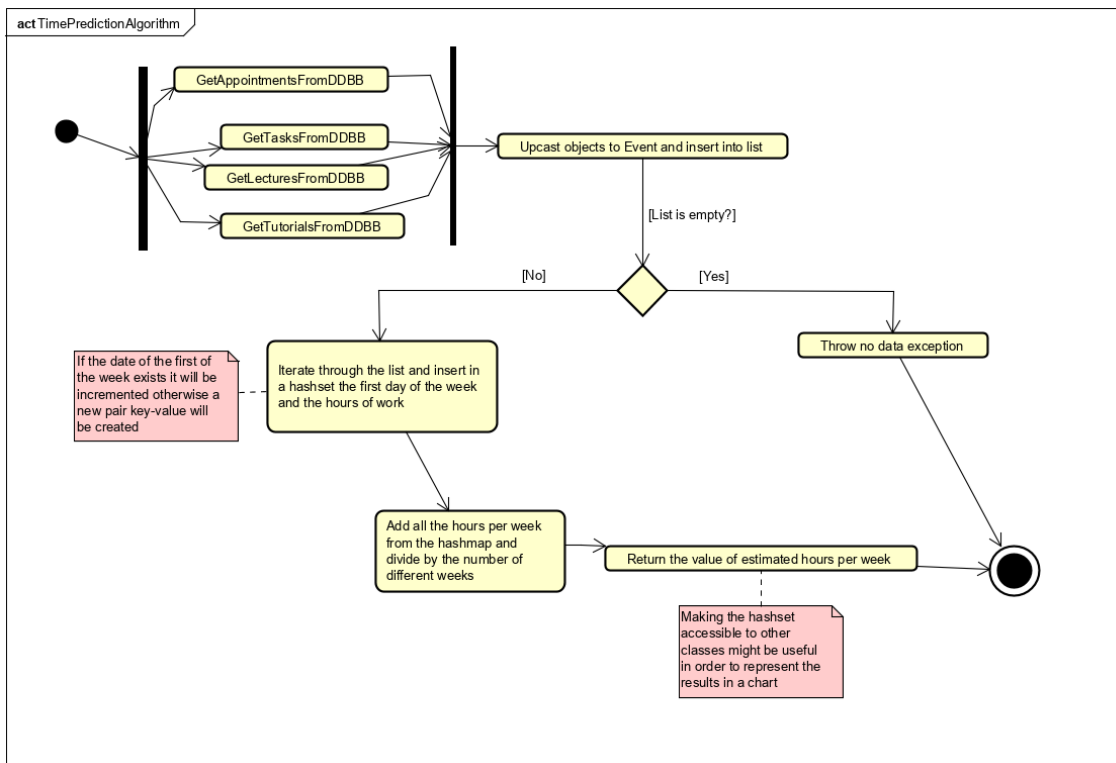


And the last one, the sequence diagram when displaying the time usage report:



## 5. Part E – Activity Diagram

This is the activity diagram describing the time usage prediction algorithm:



## 6. Part F – Reports

### DESIGN DECISIONS REPORT

The decisions taken for the design of the time management and reporting tool have been taken aiming the best performance of the system with both a good UI and UX and a good code maintainability as well as good readability of the code.

Firstly, the requirements have been written fulfilling the different functions that would give the user an enjoyable experience when using the system. The decision of implementing a data base instead of using XML files is due to the fact found in the book *Effective XML: 50 Specific Ways to Improve Your XML*: “XML is not a database. It was never meant to be a database. It is never going to be a database. Relational databases are proven technology with more than 20 years of implementation experience. They are solid, stable, useful products. They are not going away. XML is a very useful technology for moving data between different databases or between databases and other programs. However, it is not itself a database. Don't use it like one. “

Secondly, the use case diagrams as well as the sequence diagrams and the activity diagram have been designed in order to provide ease of implementation in the future. All of them try to give detailed information about how the implementation should be and primarily those parts that may lead to a confusion when it comes to programming, such as algorithms or the process to access data keeping the code in separated layers as explained ahead.

Thirdly, the decisions taken on the domain model for the implementation of the different classes belonging to the model. These decisions have been taken following some design patterns in order to get a better implementation later on: MVC pattern, unique access to data, code readability, code reusability and code maintenance.

The MVC pattern is shown in the CRC table where we can see all the different classes divided in the 3 different types (model, view or controller). The model part will be described in the following paragraphs but for the view and the controller, thanks to C# partial classes we can divide the code into 2 separated files implementing a partial class. Then, we will have the code for the controller in the .cs file and the code for the view in the .designer.cs file as explained in part C.

The model part is detailed in the domain model and is in this class diagram where we ensure a unique access to data by using a DAC (Data Access Component) class for every object that needs to store data in the data base. In the Data Access Component classes is where the connection to the data base is done and where the different types of SQL queries are performed. The methods will return Boolean values with the result of insert, update or delete queries or lists containing the information in the different rows to the model class that can be called by the controller. This classes are only called by the methods in the model class that will be called by the controller, in order to ensure that in the implementation, this classes will have package visibility. We are also fulfilling one of the characteristics of Object Oriented Design, a class must have only on purpose.

We can ensure code reusability, readability and maintenance by using inheritance. All events have some common attributes and methods that will be implemented only once so we don't need to copy and paste the code and therefore is also easier to maintain, only one file will need to be changed. Each event specific characteristics and methods will be implemented in the subclasses because they are different for each of them. The event class is abstract so it cannot be instantiated, because of the necessity of specifying the event type every time we create an event.



And finally, we are using an interface that needs to be implemented by all the DAC classes to make sure that all of them provide the main 4 methods for a DDBB: Create, Read, Update and Delete (CRUD).

In this part has also been included a simple domain model for the exceptions so that the program flow is more controlled.

The last part corresponds to the activity diagram describing the time prediction algorithm describes the flow the algorithm should follow when running but it is an implementation decision the data structures and methods which will be used to do so.

## **SUITABILITY EVALUATION FOR IMPLEMENTATION**

The suitability evaluation for the implementation of this designed will be done going through the different parts of this design.

Firstly, the requirements have been written giving enough details to fulfil the problem statement given for this system. Furthermore, a data base entity-relation model has been provided so that the programmer does not have to take those decisions on his own. It is similar to the domain model so that is easier to transform the data base information into objects in the system.

Secondly, the use case diagram and the use cases' descriptions, the use case diagrams are showing clearly the different functionalities that the program will need to provide the user. Besides it includes some use cases specifying where data validation is needed. The descriptions are simple as well as the system, so they do not need too much detail.

The third part is related to the classes, all the given classes have a unique purpose meeting the object-oriented principles. The access to the data is separated to the object which improves the maintainability of the code. If the data base needs to be modified, we will only need to modify the classes that connect to the data base. Furthermore, an interface has been included in the design so that all DAC classes provide the system with a set of basics methods. Also, in case any modification in the signature of these methods needs to be performed it can be modified in the interface and then adapt all the code that would not compile, this is improving the code maintainability.

Besides, thanks to the partial classes of C# the MVC pattern is also part of the design. However, some more detail about the prediction algorithm's implementation could have been given in this part. Finally, a good point about this design is that the exceptions have been already included so the programmer does not have to think which error should be thrown when some details are not provided, or not enough data has been provided.

In the fourth part, we can find the sequence diagrams, they describe the main flow of the execution of the different areas of the system. Although they do not provide details about variations in the main flow that may happen. During the implementation it will be necessary to take a look at both the use case's description and the sequence diagram.

And in the last part, we can find the activity diagram describing the time prediction's algorithm that will be used to get the report. It is simple, efficient and easy to understand and implement. The details about the implementation are not given in the design and is up to the programmer but by taking a look at the algorithm and the rest of the design we can claim that using lists and hash sets will be the best options.

So, in conclusion, even though some parts could include some more detailed descriptions, the design for the system would be easy to understand and implement for a programmer who has not participated in the design.