PROVISION OF AN AIR TRAFFIC CONTROL SERVICES IN AN AIRPORT ENVIRONMENT

Design and development of a Java application through the Hibernate persistence framework

Università La Sapienza di Roma

Written by:
Enrique Galduf Tel

Directed by:
Domenico Lembo
Javier Jaén

Roma 2010
1. Introduction

Actually, in the modern airports, a big amount of traffic of planes, persons, means of support is generated, and this will be managed by the controllers that need to organize all this resources to face problems, and to manage as efficient as possible the existing means.

The controllers and operators need a tool to simplify their work because of the huge amount of information generated. That's what we're going to work about, the management of the information relative to the operations that are occurring or have occurred into an airport structure. There are different operations we need to store like the landings and the take off of flights. And also some information that we've to put in inventory like the persons who work in the airport or the means of support in the airport.

This project is based on an existing real project, it concerns the design of a database in SQL and the design of an application interacting with it. In this project we provide a detailed process of construction of the database. All the construction of the entity relationship diagram is detailed and optimized.

From this documentation we've followed another approach to the problem, we're going to work with the Java and the API Hibernate. Our purpose is to make a visual application that allows us to access an equivalent database. For this, we made the classes respecting as far as possible the structure of the database. After that we use Hibernate for link the Java with the database.

Hibernate is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions.

With Java and Hibernate we got together the benefits from object-oriented languages and persistence management. After this part we make an interface with Swing, an API for providing a graphical user interface for Java programs.

Now we know better what's the problem. Let's talk in the next lines about the structure of this document with the purpose of start knowing all the technological and functional issues, the definitions and all the relationships between the elements that compose this project. We're going to follow the lines that software engineering describe, our goal is to follow a structure that talks about all the steps of the application development process in the clearest way is possible. Then the parts are the following:

- In the next part we're going to introduce the project by the specification of the
requirements. This will take about what the client and the application needs. All the information that is putted here will help the programmer to understand what's exactly what they have to develop and what requirements has to satisfy the application. We're talking about the user who uses each functionality, the functions that may execute the system, every type of restrictions, outputs.. For this purpose we base our description in the guidelines of the standard IEEE-830, etc.

- After the specification we know what the client wants and now we go to the analysis phase. In this phase we're going to extract the classes and the relationships between classes according to the requirements specified in the previous phase. We must create a clear, realistic and verifiable model, to help us with this work we can use the UML diagrams, that allows us to show the interactions inside and outside the program and to explain the behaviour of the program.

- The next step in the software engineering is the design phase. As we have taken and existing design in this phase we're going to summarize and resume the project in which we've based adding some comments about the classes and the database generated by hibernate.

- In the final part we're talking about the languages and libraries used in the implementation, with special care to the Hibernate API.
1.1 Concepts about the airport

First of all we're going to understand a little bit better how operates an airport and how the people works inside of it. We're going to explain the structures in the airport that are relevant for our application, the different types of employees working in the airport, the means of support, the airplanes and other technical concepts.

1.1.1 The airport

The composition of a modern airport, designed for the international traffic commercial and freight traffic, consist in a runway, normally in asphalt of at least 2000 meters of length, a parking area for the airplanes, a passengers terminal and hangar for technical assistance services, also the control tower.

In the biggest airports the runaways are more than one, for separate landings and take off or there are in different directions to allow the operations downwind. The runway are flanked by service roads for emergency vehicles intervention and are connected to the parking area of the airplanes using the taxiway lighting.

The airports are identified by one code called ICAO of four letters, in the technical aspect and a IATA code of unambiguous three letters for the commercial aspect.

1.1.2 Employees

We're going to use three types of employees on our application, flight controller (tower controller), land controller (ground controller) and operator. First we're going to talk about the controllers.

The Air Traffic Controller is the person who has the task of conduct the airplanes in the airspace and the airports, in a safely, orderly and quickly mode, providing the pilots with information and instructions needed, inside the airspace in his jurisdiction, to prevent crashes, overcoat between airplanes and obstacles in the maneuver area.

Inside the ATC are other specific types of controllers. One of it is the tower controller, his purpose is to control the airplanes into the airport airspace. Particularly he must assign the landing runaway and follow the airplane into the relative operations, while in the take-off phase is responsible of guiding the airplane in the runway(which has provided to assign for the operation) to the end of the airport airspace.

Another type is the ground controller, he must guide the airplanes into the airport structure. Concretely must guide the airplane from the runway to the parking zone, for that purpose he uses the taxiways, streets for guide the airplanes.

Finally we have the operators who are in charge of assign the supporting means to the arriving airplanes and has the responsibility of the controlling of the operating state of
this means and of other airport structures.

1.1.3 Runways and other roads

A runway (RWY) is a strip of land at an airport on which aircraft can take off and land and forms part of the maneuvering area. Runways may be a man-made surface (often asphalt, concrete, or a mixture of both) or a natural surface (grass, dirt, gravel, ice, or salt). The identifier of this element of the structure can be created based in the magnetic orientation of the road: for example a road orientated in 185° can be called Runway 18. If there are more runaways can use a letter that indicates the absolute position: for example 18L, 18C, 18R... His length is variable between 240m to 4800m and his widths is variable between 8m to 80m

Also exists the taxiway that links the runway with the squares of the parkings.

1.1.4 Airplanes

The airplanes have an ICAO abbreviation, a four letters code to identify the model of the airplane. For example the Boeing 747 will be B747. Also has a transponder that is a device that can identify the airplane during the operations of aerial traffic control sending a unique code assigned to the mean.

The ICAO airline designator is a code assigned by the International Civil Aviation Organization (ICAO) to aircraft operating agencies, aeronautical authorities and services. The codes are unique by airline which is not true for the IATA airline designator codes.

Each aircraft operating agency, aeronautical authority and services related to international aviation is allocated both a three-letter designator and a telephony designator. The designators are listed in ICAO Document 8585: Designators for Aircraft Operating Agencies, Aeronautical Authorities and Services.

An example is:

* Three-letter designator – AAL
* Telephony designator – AMERICAN
* Operator – American Airlines

Certain combinations of letters are not allocated to avoid confusion with other systems (for example SOS). Other designators particularly those starting with Y and Z are reserved for government organizations.

Designator YYY is used for operators that have a code allocated.

IATA airline designators, sometimes called IATA reservation codes, are two-character codes assigned by the International Air Transport Association (IATA) to the world's airlines. The standard is described in IATA's twice-annual publication, the Standard Schedules Information Manual. Airline designator codes are of the format "xx(a)",
i.e. two alpha-numeric characters followed by an optional alpha character. Although
the IATA standard provides for three-character airline designators, IATA has not used
the optional third character in any assigned code. This is because some legacy
computer systems, especially the "central reservations systems", have failed to
comply with the standard, notwithstanding the fact that it has been in place for 20
years. The codes issued to date comply with IATA Resolution 762, which provides
for only two characters. These codes thus comply with the airline designator standard
but use only a limited subset of its possible range.

A flight designator comprises the airline designator, "xx(a)" plus the numeric flight
number, "n(n)(n)(n)" plus an optional, so-called, "operational suffix" being one alpha
character, "(a)". The full format of a flight designator is thus: "xx(a)n(n)(n)(n)(a)".

Designators are used to identify an airline for commercial purposes, including
reservations, timetables, tickets, tariffs, air waybills and in airline interline
telecommunications. There are three types of designator: unique, numeric/alpha and
controlled duplicate.

Since 1967, ICAO has also issued airline identification codes. ICAO codes are three
characters in length. For many years, IATA has let it be known that they are "about"
to adopt the ICAO codes as IATA airline designators. This has never occurred and
IATA has made no statement about if they intend to do so.

After an airline is delisted, IATA can make the code available for reuse after six
months and can issue "controlled duplicates". Controlled duplicates are issued to
regional airlines whose destinations are not likely to overlap, so that the same code is
shared by two airlines. The controlled duplicate is denoted here with an asterisk (*)
following the code and in IATA literature as well.

IATA also issues an accounting or prefix code. This number is used on tickets as the
first three characters of the ticket number.
2. Requirements specifications. General definition

We want to implement an information system that takes care of management of the controlling services of aerial traffic in the airport field: this management will be focused to the land services and will be used by the competent authority (ENAV, ENAC and the airport manager) for the movement of the means in compliance to “Decreto Legge n.237” of 08/09/2004 issued by the Italian “Ministero delle infrastrutture e dei trasporti”.

2.1 Product perspective

In this section we will comment the relationship between the application with other components forming part of the system. The system hasn't external relationships with other applications or servers. The only one external interface is with the DBMS, and Hibernate takes care of this for us. We only need a computer with Mysql and a Java platform to run this application.

2.2 Product features

The information system may be able to provide to the actors the features that we're going to describe.

2.2.1 Functionalities

Runaway of landing assignment

DESCRIPTION

At the moment of the approach of an aeroplane in the airport airspace, the flight controller communicates to the aeroplane's pilot the runaway to do the landing. The flight controller must make a ticket with the information of the flight plan and the information about the runaway designed for the landing. These informations may be introduced into the information system that may be provided to the other actors.
INPUT
The user must select a company associated to the flight. The user must choose an existing aeroplane or introduce an new one by ID and acronym. The user introduces the number of the flight, the origin and the type. The user chooses a runaway from the available ones.

OUTPUT
The system must introduce into the database the runaway assignment.

RESTRICTIONS
1. The ID can't be repeated and must be an integer.
2. The origin must be an ICAO code of 4 letters.
3. The type must be choose between 'passengers', 'freight plane' or 'special'.
4. The runaway must be free.

PROCEDURE
The system shows a the companies to select the correct one. The user select a existing aeroplane or introduces an existing one. If the data is correct the user introduces the flight information. The system writes all in the database. The user selects a runaway between the free ones. The system shows a report from the operation and insert all the information in the database. If any operation fails the system must inform the user.

Route of landing assignment

DESCRIPTION
The function is used by land controller to take charge a flight in landing phase and define a route, to communicate to the flight pilot, from the runaway used in the landing to the parking area asigned.

INPUT
The user must choose between the different arriving flights. After this the user must choose a parking to the aeroplane.

OUTPUT
The system must introduce into the database the route of landing assignment.

RESTRICTIONS
1. The route and the parking must be free to use it.
procedure
The system searches all the flights with a runaway of landing assignment but with no route of landing assignment. The land controller selects a flight from the available ones. The system searches all the free parking areas. The land controller selects a parking area from the available ones and selects a route from the runaway to the parking. The system must check the information and save into the database the assignment.

Support means assignment

DESCRIPTION
This function is used by the airport operator to assign means of support to the plane in the airport structure.

INPUT
The user must choose between the different arriving flights. After this the user must choose the support means to assign to the flight.

OUTPUT
The system must introduce into the database the assignment of the means

RESTRICTIONS
1. The means must be free to use it.

procedure
The system searches the arriving flights with a landing route assigned but without a support means assignment. The airport operator choose an arriving fly. The system searches free means of support. The airport operator chooses the means between the available. The system save the information into the database.

Take off runaway assignment

DESCRIPTION
Is the function that uses the flight controller at the moment of allowing the plane at the initiation of the operations and finish the first operations. The flight controller takes the information about the flight and assigns a runaway to start the take off operations.

INPUT
The user chooses the arriving flight and the free runaway.
OUTPUT
The system saves the runaway assignment into the database.

RESTRICTIONS
1. The runaway must be free to use it

procedure
The system researches every arriving flight with an assignment of means. The flight controller selects the plane. The system makes the flight into done state. The system inserts a departure flight. The system searches all the free runaways. The flight controller introduces the information about the flight and selects a runaway. The system saves the assignment into the database.

Take off route assignment

DESCRIPTION
The function is used by the land controller to define a route to a plane in take off phase, from the parking area to the runway designed for the take off. This is communicated to the pilot of the plane.

INPUT
The user may select the plane to the assignment and the runaway.

OUTPUT
The system saves the route assignment into the database.

RESTRICTIONS
1. The route must be free to use it.

procedure
The system searches all the flights in departure with an assigned runaway but without a route assigned. The land controller selects a plane. The system selects a route from the runaway to the parking zone. The system saves the take off route.

Insert a new support means

DESCRIPTION
The function is used by the airport operator to insert a new support mean.

INPUT
The user may introduce all the information about the support mean.
OUTPUT
The system saves the information into the database.

RESTRICTIONS
1. The ID must be an integer and can't be repeated.
2. The year must be an integer.
3. The type must be choose between 'parking' or 'support'.
4. If the type is parking the subtype can be 'bus', 'baggage', 'charge', 'landing', 'supplying' and 'maintenance'.
5. If the type is support the subtype can be 'security', 'SOS' or 'fire-guard'

procedure
The system presents a form where the airport operator can insert the information about the support mean. The airport operator inserts the data. The system check the information and insert in into the database.

Modify support means

DESCRIPTION
Has been used by the airport operator to modify the data of a support mean or for change the operative state.

INPUT
The user must insert the old information for search the old register and the new information for change it.

OUTPUT
The system updates the database with the new information.

RESTRICTIONS
1. The ID can't be changed.
2. The year must be an integer.
3. The type must be choose between 'parking' or 'support'.
4. If the type is parking the subtype can be 'bus', 'baggage', 'charge', 'landing', 'supplying' and 'maintenance'.
5. If the type is support the subtype can be 'security', 'SOS' or 'fire-guard'.

procedure
The airport operator complete a form to stablish the search criteria. The system
searches the mean if exists else shows a error message. The operator selects the wanted mean. The system shows all the data of the mean. The operator modifies what he want. The system checks all and update the mean into the database.

**Delete support means**

**DESCRIPTION**
Is used by the airport operator to delete a support mean that is not in service in to the airport structure. The delete is not physical but is logical to maintain the information about the activities done by the mean. The logic delete have been done updating the operative state into disabled, that can't select it another time for the support operations.

**INPUT**
The operator must search and select the support mean.

**OUTPUT**
The system deletes the support mean from the database.

**RESTRICTIONS**
1. The mean must be enabled.

**procedure**
The airport operator complete a form to research the support mean. The system shows a list or a error message. The operator selects the mean. The system provides a information about the mean. The operator confirms the delete. The systems delete it from the database.

**Modify Structure**

**DESCRIPTION**
This function is used by the airport operator to modify the data of a structure element of to modify his operative state.

**INPUT**
The user must insert the old information for search the old register and the new information for change it.

**OUTPUT**
The system updates the database with the new information.
RESTRICTIONS

1. The type of the surface can be MACADAM or cement.

procedure

The airport operator complete a form in which the user can specify the research criteria. The system do the research of the desired element if exists, else shows a error message. The airport operator selects the wanted element. The system shows a page with the structure element. The airport operator modifies the wanted data. The system updates the database.

Airport structure report

DESCRIPTION

Informative function that provides the information of the state of the airport structure with special importance of the operative runaway.

INPUT

None.

OUTPUT

The system shows all the information of the structures in the airport

RESTRICTIONS

None.

procedure

The system researches all the runaways and all the structure elements not operative. The system shows a report of the found data.

Support means report

DESCRIPTION

Informative function that provides the information of the means actually in service in the airport structure.

INPUT

None.

OUTPUT

The system shows all the information of the means in the airport.
RESTRICTIONS
None.

procedure
The system researches all the support means operative and non operative. The system shows a list of the found data.

Airport movement report

DESCRIPTION
Informative function that provides a report from the registered movements into the airport structure.

INPUT
None.

OUTPUT
The system shows all the information about the arrivals and departures into the structure.

RESTRICTIONS
None.

procedure
The system researches all the movements in the airport done in the present day, present month and present year. The system shows a list of the found data.

2.2.2 Other requirements
In this part a brief description will be included of some hardware and software requirements. Also some details on the graphical user interface are provided.

2.2.2.1 Graphical user interface
The interfaces are realized with the Swing library, and are composed of list, text fields, combo boxes, labels, etc... Which provide a common window interface in Java.

There's not limit for the design of the interfaces then we use a clean interface which can obtain a better usability.

2.2.2.2 Hardware interfaces
The system only needs a complete computer, no special hardware is needed.
2.2.2.3 Software interfaces

It's needed to install the database (recommended Mysql), and have installed a Java Platform to run the program.

2.2.3 Non functional requirements

There are some qualities of the software that will be token into account. We are going to talk about reliability, maintainability and portability.

2.2.3.1 Reliability

At the data level we may check correctness the insertions, updates and deletes into the database and show confirmation messages towards assurance of the reliability of the program.

2.2.3.2 Application-DB connection maintenance

In this application we don't need to care about to make changes in the database cause hibernate makes this for us. The update of the Java application is easy but the mapping files are some difficult to maintain and make changes.

2.2.3.3 Portability

MySql and Java can run in almost all OS, who guarantee us that we can run the application in the most important systems.
3. Analysis

3.1 Introduction

Into this phase, we're going to transform the requirement specification into a clear, realistic and verifiable system.

The analysis process is the previous step to the software design, that may give us the structure of the system, because of this is very important make a good study of the requirements and interpret them correctly to later do his implementation. In this point we've to verify if our communication with the client was correct and we've done well all his requests. If we've any problem we must redo the requirement specification which are not correct, before continuing with the development.

In this process we're going to show in a graphical way how we transform the requirements into the object model that will be the static description of the model. Then we're going to describe case uses, identify and describe the objects and the interactions between them.

To do this job we're going to use the object oriented model language UML tailored to the analysis phase. Between the various diagrams that offers, in this section we're going this types of diagrams:

- Use case diagram: shows the interactions between the system and the outside, by the user or by other subsystems. In this way are clear the communications between the functionalities.
- Class diagram: is a schema where we show the static part of the classes, there we show the classes of the system and the relationships between them.

3.2 Use cases

The use cases are methods to know the goals of the software. In them we show the interaction between the application and the external components, that can be users or other subsystems to have and specific objective.

Ultimately, in a use case we represent the communications between a principal actor and the system between the actor and the subsystems of the application. We use them to show the requirements of the system when a event occurs.

Now we show some of the elements that we use in this diagrams
This graphic represents the actors who interact with the systems. Due to them we establish the communication that will determine the interaction between the external entities with the application.

The use cases are represented by this icons. In it we have a functionality with an defined objective, that satisfies some requirement established by the client.

We use the links to specify the connection between to elements. We can indicate the sense of the communication. Can be directed or undirected.

Amplifies the functionality of other use case.

If needs another use case to do his job.
3.3 Class diagram

With using this diagram, we can show in a static way, the conceptual design of the system information, and the working components with the relationship between them. Will appear all the structure of the system divided in classes, we also can include his most important methods and attributes and the existing links between them.
4. Design

4.1 Introduction

The design of our system is based on the approach of the mentioned database project. But his schema is not completely true cause for the final design we have let hibernate to do this moment. Now we show the generated SQL tables by mySQL. For a clarification we've joined together the entities of the land means to simplify the schema and added the attributes type and subtype. Also some child tables are joined in one big father one to simplify the use of the inheritance. Hibernate does this automatically and makes an attribute called discriminator type to select between the different entities.

This is the standardized schema.
### 4.2 Table design

**TABLE: AEROPORTO**

**DESCRIPTION**

In this table we're going to store the information about the airports we're using in our database. The airports are been identified by the ICAO primary key of four letters (each airport has a different one), also we include the IATA code. The difference between this two codes is that the ICAO code is focused mainly to the technical purpose (for example the flight planning) and the IATA code is used into the commercial matters (like ticket emission). The other attributes we include are the name, city and country of the airport.

**DESCRIPTION OF THE ATTRIBUTES**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICAO</td>
<td>String</td>
<td>Four letters unique code for identify an airport</td>
</tr>
<tr>
<td>NOME</td>
<td>String</td>
<td>Name of the airport</td>
</tr>
<tr>
<td>CITTA</td>
<td>String</td>
<td>City of the airport</td>
</tr>
<tr>
<td>NAZIONE</td>
<td>String</td>
<td>Country of the airport</td>
</tr>
<tr>
<td>IATA</td>
<td>String</td>
<td>Three letters code for the airport</td>
</tr>
</tbody>
</table>
**TABLE: PERSONALE**

**DESCRIPTION**

We include in this table all the types of personal that can be used in the database, we joined into this table all the types of personal. All the types have common attributes, they are the name, surname, document number, the registration number and the date of birth. In this table there are attributes that aren't useful in all types of personal but hibernate manages it. The personal can be divided in three types:

- Airport operator, they are responsible of the management of the supporting means for the air plane, they have an attribute for the recruitment date.
- Ground controller, he's in charge of guide the air planes into the airport structure. He has a particular attribute called qualifica to write the level of qualification (controller, supervisor, boss, director...)
- Tower controller, he manages the runaways of landing and take off procedures. He uses also the qualifica attribute and have another one particular attribute called abilitazione we're we indicate the type of radar that the controller is allowed to use.(approach radar or regional radar).

**DESCRIPTION OF THE ATTRIBUTES**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>COD_FIS</td>
<td>String</td>
<td>The Italian identification document code.</td>
</tr>
<tr>
<td>DISCRIMINATOR</td>
<td>String</td>
<td>A internal attribute for Hibernate that is used to distinguish between the different types of personal</td>
</tr>
<tr>
<td>MATRICOLA</td>
<td>String</td>
<td>Identifies the personal of the airport</td>
</tr>
<tr>
<td>NOME</td>
<td>String</td>
<td>The name of the employee</td>
</tr>
<tr>
<td>COGNOME</td>
<td>String</td>
<td>The surname of the employee</td>
</tr>
<tr>
<td>DATANASC</td>
<td>String</td>
<td>Date of birth of the employee</td>
</tr>
<tr>
<td>DATASUN</td>
<td>String</td>
<td>Recruitment date (only for airport operator)</td>
</tr>
<tr>
<td>QUALIFICA</td>
<td>String</td>
<td>Level of qualification (only for controllers)</td>
</tr>
<tr>
<td>ABILITAZIONE</td>
<td>String</td>
<td>Type of radar used (only for tower controllers)</td>
</tr>
<tr>
<td>PERS_ID</td>
<td>String</td>
<td>Identifier of the airport</td>
</tr>
</tbody>
</table>
**TABLE: COMPAGNIA**

**DESCRIPTION**
In this table we include the information about the aerial company his official code, his name and his country.

**DESCRIPTION OF THE ATTRIBUTES**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODICE</td>
<td>String</td>
<td>The ICAO code of the company composed by three letters</td>
</tr>
<tr>
<td>NAZIONALITA</td>
<td>String</td>
<td>Country of the company</td>
</tr>
<tr>
<td>NOME</td>
<td>String</td>
<td>Name of the company</td>
</tr>
</tbody>
</table>

**TABLE: AEROMOBILE**

**DESCRIPTION**
This table stores the information about the air planes. The air planes are identified unequivocally by three fields together. The first one is ICAO, represents the ICAO code of four letters that represents the air plane model (Ex. Boeing 747 is B747). The other one is ID, this field represents the identifier of the plane saved in the transponder (the transponder is a device in the planes used in the aerial handling). The final one is ID_COMPAGNIA a foreign key that matches with the primary key of the company.

**DESCRIPTION OF THE ATTRIBUTES**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Integer</td>
<td>The transponder identifier</td>
</tr>
<tr>
<td>ICAO</td>
<td>String</td>
<td>The four letter code for represent the model of plane</td>
</tr>
<tr>
<td>ID_COMPAGNIA</td>
<td>String</td>
<td>The ICAO code of the company composed by three letters</td>
</tr>
</tbody>
</table>
**TABLE: VOLOARRIBO**

**DESCRIPTION**

In this table we have all the information about the arriving flights into the airport area, and the flights who have arrived recently and are going to be guided into the airport structure. We have a composed primary key that represents the flights. We have a number of flight and a date of the flight and together define a unique flight. We can define the type of flight in the tipo attribute: passengers flight, flight of goods, etc... In the field origine we can write the name of the origin city of the flight. Finally ID_FK and ICAO_FK are keys generated by hibernate to do the relationship between the flight and the plane that does this flight.

**DESCRIPTION OF THE ATTRIBUTES**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>numero</td>
<td>Integer</td>
<td>The number of the flight (depends of the day)</td>
</tr>
<tr>
<td>data</td>
<td>String</td>
<td>Day of the flight</td>
</tr>
<tr>
<td>tipo</td>
<td>String</td>
<td>Describes the type of the flight</td>
</tr>
<tr>
<td>stato</td>
<td>String</td>
<td>Describes if the flight is finished or not</td>
</tr>
<tr>
<td>origine</td>
<td>String</td>
<td>The city of origin of the flight</td>
</tr>
<tr>
<td>ID_FK</td>
<td>Integer</td>
<td>The air-plane's transponder identifier</td>
</tr>
<tr>
<td>ICAO_FK</td>
<td>String</td>
<td>The four letter code for represent the model of plane</td>
</tr>
</tbody>
</table>
TABLE: VOLOPARTENZA

DESCRIPTION
In this table we've the information about the planes that are going to take off out of the airport area, and the flights who are in the internal structures on the airport like parkings, hangars, etc. and have finished all the process of the landing and are prepared to take off. We have a composed primary key that represents the flights. We have a number of flight and a date of the flight and together define a unique flight. We can define the type of flight in the tipo attribute: passengers flight, flight of goods, etc... In the field partenza we can write the name of the destination city of the flight. Finally ID_FK and ICAO_FK are keys generated by hibernate to do the relationship between the flight and the plane which does this flight.

DESCRIPTION OF THE ATTRIBUTES

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>numero</td>
<td>Integer</td>
<td>The number of the flight (depends of the day)</td>
</tr>
<tr>
<td>data</td>
<td>String</td>
<td>Day of the flight</td>
</tr>
<tr>
<td>tipo</td>
<td>String</td>
<td>Describes the type of the flight</td>
</tr>
<tr>
<td>stato</td>
<td>String</td>
<td>Describes if the flight is finished or not</td>
</tr>
<tr>
<td>origine</td>
<td>String</td>
<td>The city of origin of the flight</td>
</tr>
<tr>
<td>ID_FK</td>
<td>Integer</td>
<td>The airplane's transponder identifier</td>
</tr>
<tr>
<td>ICAO_FK</td>
<td>String</td>
<td>The four letter code for represent the model of plane</td>
</tr>
</tbody>
</table>
TABLE: M_TERRA

DESCRIPTION

In the next table we put the information about the land and rescue meanings that are present in the airport. We will store information about the working state of this type of helping devices. We've made a change to the original design of the table and instead of doing different tables for the different types of land meanings we've added two attributes called type and subtype which can do the same function and simplify our design. The attribute type can be a parking or a rescue meaning and into this two types we can be more specific and describe the subtypes. On the side of parking we've six different types: bus, baggage, charging, landing, replenishment and maintenance. On the other side we've three different types: security, rescue and anti fire. We've an unique identifier for our meanings. We've also the model, the year of construction, the year in which starts working and the operating state which indicate to us if the meaning can work or cannot.

DESCRIPTION OF THE ATTRIBUTES

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Integer</td>
<td>The id attribute is a number which identifies our meaning</td>
</tr>
<tr>
<td>S_OPERATIVO</td>
<td>String</td>
<td>Indicates us if the mean is ready or not to work</td>
</tr>
<tr>
<td>MODELLO</td>
<td>String</td>
<td>The model of the device</td>
</tr>
<tr>
<td>ANNO_C</td>
<td>Integer</td>
<td>The year of construction of the mean</td>
</tr>
<tr>
<td>ANNO_I</td>
<td>Integer</td>
<td>The year in which the device starts to provide services</td>
</tr>
<tr>
<td>TIPO</td>
<td>String</td>
<td>The type of the mean</td>
</tr>
<tr>
<td>SUBTIPO</td>
<td>String</td>
<td>The subtype of the mean</td>
</tr>
<tr>
<td>DATA_FK</td>
<td>String</td>
<td>Is a foreign key indicating the date of the flight in which is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>assigned the meaning</td>
</tr>
<tr>
<td>HORA_FK</td>
<td>String</td>
<td>Is a foreign key indicating the hour of the flight in which is</td>
</tr>
<tr>
<td></td>
<td></td>
<td>assigned the meaning</td>
</tr>
<tr>
<td>ID_FK</td>
<td>String</td>
<td>Is a foreign key indicating the identifier of the flight in which</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is assigned the meaning</td>
</tr>
</tbody>
</table>
In this category we include all the structures that are used by the planes to drive along the airport. There are different types of structures and we could have done different tables by is easier use hibernate in this way so we have done one table for all the subtypes of structures. First let's see the common attributes. An element is unambiguously identified by an integer ID. The attribute DataCostruzione indicates the date of building of the structure, TipoSuperficie indicates the material of the structure (cement or MACADAM), StatoSuperficie indicates the good or bad state of the structure and finally Operavilita is used to know if the structure is in correctness of working and have no problem to drive on it. Now we're going to do a description of the hierarchy of the structures. On one hand we've Area_di_manovra with the attributes Segnaletica which indicates if the road sign is available or not, and LuciDelimitazione indicates if the lights are available, there are two subtypes of Area_di_manovra, one type is Raccordo that represents the links between the runway (has not extra attributes) and the other one is Pista which includes the attributes Lungheza and Largheza to indicate the long and width. Pista is divided in two, the first one is Runaway and the other is Taxiway to link the Runway with the parkings (has not extra attributes). The Runaways to landing and taking off have extra attributes; Orientazione to indicate the orientation angle from north, Latitudine and Longitudine define the coordinates, Elevazione define the altitude from the sea, DistanzaDecollo indicates the useful distance for the taking off and finally DistanzaAtterraggio indicate the useful distance for landing operations. On the other hand of the structure we've Linea_di_volo wich can be divided in two. The first one is Piazzale (without extra attributes) who represent the squares where are the parkings, the other one Parcheggio represent the parkings of the planes, his attributes are Dotazione to represent the equipment of the parking and Stato to know if is free or busy.

DESCRIPTION OF THE ATTRIBUTES

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Integer</td>
<td>A number for identify the structure</td>
</tr>
<tr>
<td>DISCRIMINATOR TYPE</td>
<td>String</td>
<td>A code to help hibernate to distinguish between the different entities</td>
</tr>
<tr>
<td>DATA_CONS</td>
<td>String</td>
<td>The date of building of the structure</td>
</tr>
<tr>
<td>TIPO_SUP</td>
<td>String</td>
<td>Material of the surface</td>
</tr>
<tr>
<td>STATO_SUP</td>
<td>String</td>
<td>The state of the surface (from very bad to very good)</td>
</tr>
<tr>
<td>OPERAVILITA</td>
<td>String</td>
<td>The operability of the runway</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DOTAZIONE</td>
<td>String</td>
<td>The equipment of the structure</td>
</tr>
<tr>
<td>STATO</td>
<td>Bool</td>
<td>If the runway is free or not</td>
</tr>
<tr>
<td>AM_SENGALETICA</td>
<td>String</td>
<td>If the road sign is available</td>
</tr>
<tr>
<td>AM_LUCI</td>
<td>String</td>
<td>If the lights are available</td>
</tr>
<tr>
<td>Strut1Id</td>
<td>Integer</td>
<td>The foreign key for indicate the first structure linked by Raccordo</td>
</tr>
<tr>
<td>Strut2Id</td>
<td>Integer</td>
<td>The foreign key for indicate the second structure linked by Raccordo</td>
</tr>
<tr>
<td>LUNGHEZA</td>
<td>Float</td>
<td>The length of the runway</td>
</tr>
<tr>
<td>LARGHEZA</td>
<td>Float</td>
<td>The width of the runway</td>
</tr>
<tr>
<td>ORIENTAZIONE</td>
<td>Float</td>
<td>The orientation of the runway</td>
</tr>
<tr>
<td>LATITUDINE</td>
<td>Float</td>
<td>The latitude of the runway</td>
</tr>
<tr>
<td>LONGITUDINE</td>
<td>Float</td>
<td>The longitude of the runway</td>
</tr>
<tr>
<td>ELEVAZIONE</td>
<td>Float</td>
<td>The elevation to the sea</td>
</tr>
<tr>
<td>PENDENZA</td>
<td>Float</td>
<td>The pendent of the runway</td>
</tr>
<tr>
<td>DIST_DEC</td>
<td>Float</td>
<td>The useful distance for take off</td>
</tr>
<tr>
<td>DIST_ATT</td>
<td>Float</td>
<td>The useful distance for landing</td>
</tr>
<tr>
<td>S_LUCI</td>
<td>String</td>
<td>The light system</td>
</tr>
<tr>
<td>STRU_ID</td>
<td>String</td>
<td>Foreign key to the airport of the structure</td>
</tr>
</tbody>
</table>
TABLE: PERCORSO

DESCRIPTION
Percorso is a list of structures used in one landing or take off process but in this table we only have the ID of the route.

DESCRIPTION OF THE ATTRIBUTES

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDperc</td>
<td>Integer</td>
<td>The number of the route</td>
</tr>
</tbody>
</table>

TABLE: PERC_STRUTT

DESCRIPTION
This table is used for the relation between Percorso and Struttura. We've one foreign key for the Percorso, other for Struttura and the third attribute is for know the order in the route.

DESCRIPTION OF THE ATTRIBUTES

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRI_ID</td>
<td>Integer</td>
<td>Foreign key to the table struttura</td>
</tr>
<tr>
<td>elt</td>
<td>Integer</td>
<td>Foreign key to the table Percorso</td>
</tr>
<tr>
<td>ordine</td>
<td>Integer</td>
<td>Order in the route</td>
</tr>
</tbody>
</table>
TABLE: A_MEOZZI

DESCRIPTION
When a plane is on the airport it needs to assign some means of support. The table A_mezzi represent the ternary relationship between one flight, the means assigned to it and the operator who makes the assignment. First we've the date and the hour of the assignment and together with the ID of the mean which we're assigning make the primary key of this entity. The attributes numero_fk and data_fk represent the foreign key of the flight that we're going to assign. Finally the attribute OP is the foreign key to design the operator who have made the operation.

DESCRIPTION OF THE ATTRIBUTES

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>String</td>
<td>Date of the assignment</td>
</tr>
<tr>
<td>Hora</td>
<td>String</td>
<td>Hour of the assignment</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>ID of the assignment</td>
</tr>
<tr>
<td>numero_FK</td>
<td>Integer</td>
<td>Number of flight</td>
</tr>
<tr>
<td>data_FK</td>
<td>String</td>
<td>Date of the flight</td>
</tr>
<tr>
<td>OP</td>
<td>String</td>
<td>Codice Fiscale of the operator</td>
</tr>
</tbody>
</table>
**TABLE:A_RUNAWAY**

**DESCRIPTION**

When a plane is prepared to land or to take off in the airport we make an assignation of the runaway in which it can be done. It have a ternary primary key with the date, hour and identifier of the assignment. We've the number and the date of the flight that we're going to assign. If is a arriving flight we use the attributes numero_FK and data_FK otherwise if is a departure we use the attributes numero_FK2 and data_FK2. We save the Flight Controller in the attribute miCVolo and the runaway assigned in miRunaway.

**DESCRIPTION OF THE ATTRIBUTES**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>String</td>
<td>Date of the assignment</td>
</tr>
<tr>
<td>hora</td>
<td>String</td>
<td>Hour of the assignment</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>Identifier of the assignment</td>
</tr>
<tr>
<td>numero_FK</td>
<td>Integer</td>
<td>Number of the arriving flight</td>
</tr>
<tr>
<td>data_FK</td>
<td>String</td>
<td>Date of the arriving flight</td>
</tr>
<tr>
<td>numero_FK2</td>
<td>Integer</td>
<td>Number of the outgoing flight</td>
</tr>
<tr>
<td>data_FK2</td>
<td>String</td>
<td>Date of the outgoing flight</td>
</tr>
<tr>
<td>miCVolo</td>
<td>String</td>
<td>Codice Fiscale of the Flight Controller</td>
</tr>
<tr>
<td>miRunaway</td>
<td>Integer</td>
<td>Runaway assigned to the flight</td>
</tr>
</tbody>
</table>
**TABLE:A_PERCORSO**

**DESCRIPTION**
When a plane is arrived to the runway needs an assignment of a route for go from the runway to a free parking. The assignment have an hour, a date and a identifier. The attribute miCTerra represents the land controller who have made the assignment. The attribute miPercorso represents a foreign key to the route assigned to the plane. Finally we've the attributes numero_FK and data_FK which represent the number and the date of the arriving flight, otherwise if it's an outgoing flight we use numero_FK2 and data_FK2.

**DESCRIPTION OF THE ATTRIBUTES**

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>TYPE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>String</td>
<td>Date of the assignment</td>
</tr>
<tr>
<td>hora</td>
<td>String</td>
<td>Hour of the assignment</td>
</tr>
<tr>
<td>ID</td>
<td>String</td>
<td>Identifier of the assignment</td>
</tr>
<tr>
<td>miCTerra</td>
<td>String</td>
<td>Reference to the land controller</td>
</tr>
<tr>
<td>miPercorso</td>
<td>Integer</td>
<td>Reference to the assigned route</td>
</tr>
<tr>
<td>numero_FK</td>
<td>Integer</td>
<td>Number of the arriving flight</td>
</tr>
<tr>
<td>data_FK</td>
<td>String</td>
<td>Date of the arriving flight</td>
</tr>
<tr>
<td>numero_FK2</td>
<td>Integer</td>
<td>Number of the outgoing flight</td>
</tr>
<tr>
<td>data_FK2</td>
<td>String</td>
<td>Date of the outgoing flight</td>
</tr>
</tbody>
</table>
5. Implementation: Realizing the data layer

5.1 Layer architecture

In this application we've used a three layer architecture for the software design. This means that we've separate procedures doing different works, then we've different modules developed and maintained independently, so is easy to maintain because of their independence. We divide the application into three different parts:

- **Interface layer**: the user interface is the top level layer in an application. The function of the interface is to provide the user results and tasks that they can understand easily, and also provide the computer the inputs that they need, and if it's possible check the correctness of them. In our application we've used Swing for this purpose.

- **Logic layer**: this layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. Its main purpose is move and process the data between the other two layers. Our application is build in Java. For the traducing between this layer and the database layer we use the library Hibernate.

- **Database layer**: the information there is saved and read from a database or file system. The information comes and goes from this layer to the logic layer. We use mySql for this purpose.
5.3.1 Conclusion

The use of a data layers into this project can be useful. This type of development is better because we can change or modify parts of the application without changing anything. For example, change the interface, the structure of the database, put the different layers in different computers, etc. For example our application doesn't use all the information of the database. Also some information may be introduced directly on the database. Maybe we can create a new application to manage all this information, or make a new interface for a different use of the application.
5.2 MySQL

5.2.1 Introduction

The software Mysql is a Database Management System of the structured language SQL. His main benefits are speed, robustness and multi-threaded and multi-user capability. Due to their characteristics is used for critical application environments, with a high workload, and to integrate with software.

Mysql uses the standard for relational databases SQL. This programming language was commercialized first for IBM in 1981. Since 1986, the standard SQL has been different versions like SQL:92, SQL:99 and SQK:2003. Mysql starts in the open-source company MySQL AB initially established in Switzerland in 1995, its founders were David Axmark, Allan Larsson and Michael Widenius. The objective of this company consist in MySQL satisfies the standard SQL, but without sacrificing speed, reliability or usability.

The software have double license. In this way, is allowed to choose the option Open Source, under the terms of the GNU (General Public License), or the standard commercial license that includes support and assistance. This is possible due to is property and is sponsored by a private company, that have the copyright of almost all source code.

We face a software very used, is supported by lots of platforms due tu his variety of API's wrote in some of the most popular languages and for debuggers like Bugzila.

5.2.2 Characteristics

Mysql is a very fast database in reading when uses the non transactional motor, but may cause problems in environments of high integrity concurrence in the amendment.

These are some characteristics that can be useful to our application:

- Internals and portability
  - Write in C and C++
  - Tested in different compilers
  - Works in different platforms
  - Uses GNU portability
  - API's for a lot of programming languages
  - Allows multi-threaded development
  - Use a very fast disk tables with index compression
  - The server is available as a separate program for applications
client-server

• Type of columns
  ◦ Allows different type of columns: Integer, Float, Double, Char, Varchar, Text, Blob, Date, Time, etc...
  ◦ Records of fixed length and variable length

• Security
  ◦ Privilege and password system very flexible and secure.
  ◦ Treatment of traffic or passwords using encryption

• Scalability and limits
  ◦ Allows the support of big databases

• Location
  ◦ Sample error messages in multiple languages
  ◦ Support for Unicode

• Clients and tools
  ◦ Mysql has support for SQL commands to check, optimize and repair tables
  ◦ All programs allow the on line help

5.2.3 Conclusion

Mysql is a used Database Management system. It provides speed, integration with Java via Hibernate, consistency, portability and robustness. Mysql is a good choose for us purposes.

In our application we haven't made the database directly, because the Hibernate library have made it for us, using the mapping files we've described how we want the database.
5.3 Realizing the application: the Hibernate framework

5.3.1 Introduction

In the design of a Java application a very important part is the way in which we access to our data in the database. To know this part will be a critical point for the future development. The traditional way to access to database is with JDBC directly connected to Database with SQL sentences, is useful but the code is difficult to maintain. Another way more advanced is DAO (Data Access Object). The problems of this implementation are also the maintain and his portability.

Hibernate is the better bridge between the application and the database. Historically, Hibernate facilitated the storage and retrieval of Java domain objects via Object/Relational Mapping. Today, Hibernate is a collection of related projects enabling developers to utilize POJO-style domain models in their applications in ways extending well beyond Object/Relational Mapping. Hibernate is a persistence object/relational layer and a sql sentences generator. Allows the design of persistence objects who can include polymorphism, relationship, collections, and a lots of data types. In a quickly and optimized way we can generate Database in all the supported environments: Oracle, DB2, MySql, etc... And the most important of all, is open source, and we don't need to pay for it.

Gavin King led a team of Java software developers from around the world in the development of Hibernate. JBoss, Inc. (now part of Red Hat) later hired the lead Hibernate developers and worked with them in supporting Hibernate.

As of 2010 the current version of Hibernate is Version 3.x. This version introduced new features like a new Interceptor/Callback architecture, user defined filters, and JDK 5.0 Annotations (Java's meta-data feature). As of 2010[update] Hibernate 3 is a certified implementation of the Java Persistence API 1.0 specification via a wrapper for the Core module which provides conformity with the JSR 220 JPA standard.

Hibernate allows to main ways to develop top-down and bottom-up. In this project we're using the top-down development process. First we do a the Java classes, after this we do the XML mapping and hibernate will create the database for us. Then go into more details of the ways to develop.

- **Top down**, In top-down development, you start with an existing domain model, its implementation in Java, and (ideally) complete freedom with respect to the database schema. You must create mapping meta-data -either with XML files or by annotating the Java source- and then optionally let Hibernate's hbm2dll tool generate the database schema. In the absence of an existing database schema, this is the most comfortable development style for most Java developers. You may even use the Hibernate Tools to automatically refresh the
database schema on every application restart in development.

- **Bottom up** - Conversely, bottom-up development begins with an existing database schema and data model. In this case, the easiest way to proceed is to use the reverse-engineering tools to extract meta-data from the database. This meta-data can be used to generate XML mapping files, with hbm2hbmxml for example. With hbm2java, the Hibernate mapping meta-data is used to generate Java persistent classes, and even data access objects—in other words, a skeleton for a Java persistence layer. Or, instead of writing to XML mapping files, annotated Java source code (EJB 3.0 entity classes) can be produced directly by the tools. However, not all class association details and Java-specific meta-information can be automatically generated from an SQL database schema with this strategy, so expect some manual work.

- **Middle out** - The Hibernate XML mapping meta-data provides sufficient information to completely deduce the database schema and to generate the Java source code for the persistence layer of the application. Furthermore, the XML mapping document isn't too verbose. Hence, some architects and developers prefer middle-out development, where they begin with handwritten Hibernate XML mapping files, and then generate the database schema using hbm2ddl and classes using hbm2java. The hibernate XML mapping files are constantly updated during development, and other artefacts are generated from this master definition. Additional business logic or database objects are added through subclassing and auxiliary DDL. This development style can be recommended only for the seasoned Hibernate expert.

- **Meet in the middle** - The most difficult scenario is combining existing Java classes and an existing database schema. In this case, there is a little that the Hibernate tool set can do to help. It is, of course, not possible to map arbitrary Java domain models to a given schema, so this scenario usually requires at least some re-factoring of the Java classes, database schema or both. The mapping meta-data will almost certainly need to be written by hand and in XML files (though it might be possible to use annotations if there is a close match). This can be an incredibly painful scenario, and it is, fortunately exceedingly rare.

### 5.3.2 Configuration

To configure hibernate first we must take a look on the `hibernate.cfg.xml` file. This file must stay at the class path of our application. Is the way to configure the connection and the properties. This XML file starts with a document type declaration used by the XML parser to validate this document against the Hibernate configuration DTD.
First in the configuration file are the database connection settings. You need to tell Hibernate which database JDBC driver you're using and how to connect to the database with a URL, a user-name, and a password. You set a Dialect, so that hibernate knows which SQL variation it has to generate to talk to your database; dozens of dialects are packaged with Hibernate.

<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost/progettoaereo</property>
<property name="connection.username">root</property>
<property name="connection.password">xxxx</property>

There are two useful properties for show the SQL instructions executed by hibernate to the console. All the executed queries and other commands in SQL that hibernate makes automatically will be showed to us. This is very important in development process because we can detect important bugs at runtime.

<property name="show_sql">true</property>
<property name="format_sql">true</property>

Another property showed in this file is used to allow Hibernate to make the database. Automatically validates or exports schema DDL to the database when the SessionFactory is created. We can use another options but the most common are create and update. The create option creates a new database, and the update option updates an existing one. In the option create-drop the database schema will be dropped when the SessionFactory is closed explicitly.

<property name="hbm2ddl.auto">create</property>

Finally we can see the mapping files at the end of the configuration file. The mapping files let us specify how we want to map the different classes and attributes into the database. We explain later how to make and use this files.

<mapping resource="/Struttura.hbm.xml"/>
Another configuration files are hibernate.properties and log4j.properties. In the first one we can also specify class path and another properties but in default mode Hibernate uses the hibernate.hbm.xml file to set this parameters.

The other one log4j.properties is the configuration of the Java library Log4j. We use this library to print in the command line the logging on executing time. Therefore we can see all the process of Hibernate and this will help use in the development of our application.

5.3.3 The mapping files

5.3.3.1 Basic mapping

The files hbm are mapping files and are written in XML. These files relate a class with a database table. Lets see with and example the different elements of these files. The example is the file Personale.hbm.xml from our project.

Firstly we can see the header. This header is a typical XML header common to all of our files and we're not going to describe it.

```xml
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

After this we can see the firs label hibernate mapping his element delimits our mapping file and allows us to specify some properties. Some of these options are the name of the database schema, the package and some options to set default value of cascade, lazy and access. We don't often use these options.

The next element we see is the most important, the class label. In this element we link the class to his table in the database with the name attribute, and also we can specify the table we want to link in the table attribute. However we haven't used this property because this class is abstract and don't have a direct relationship into the database (look at the abstract property). We explain this more in detail later when we talk about inheritance.

The other property we can see at the example file is lazy. The default value is true but in some classes we need to put them on false. If lazy is true Hibernate don't access to the database until he need the object. This allows hibernate to save resources, however he can make to us some problems.
Another option than we can see in this project is *discriminator-value* this property is important from some types of inheritance and we explain it later. We can see more options in the Hibernate documentation.

```xml
<class
    name="progettoaereo.Personale"
    abstract="true"
    lazy="false">"}

After link the object with the table, we have to link the properties of the object with the columns of the table. In our example, we can see that to describe this link we use to different XML elements: *id* and *property*.

The first one, *id*, links one (later we explain how to link more than one) of the properties with the primary key of the table. We identify the property that we link with the primary key using the XML attribute *name*. In our case the property we're going to link is *codicefiscale*. This property is linked with the table column *COD_FIS*. If the name of the column and the property are the same we don't need to specify the attribute *column*.

With the attribute *type* we have the option of specify the type of data which we are working. Hibernate can do this automatically so we don't need to specify this.

Between the open and close *id* labels we put the *generator* element. This element let us define how do the primary key values be generated. Hibernate offers a lot of methods to generate values to the primary keys. In our example we use the method *assigned* who let to the application assign an identifier before calling to save(). Another interesting method is *identity* who generate automatically the primary key. We can also use *native* method. This method let hibernate choose between the methods *identity, sequence* or *hilo*. This methods are similar but hibernate selects one of them according to the database characteristics.

```xml
<id
    name="codicefiscale" column="COD_FIS">
    <generator class="assigned"></generator>
</id>
```

When we have a composite primary key we must use *composite-id*, into this label we put the different keys that we're going to use into the *key-property* label. Very important: the persistent class must overwrite the methods *equals()* y *hashCode()* to implement the composite identificator. Also must implements *Serializable*.

```xml
<composite-id>
    <key-property name="ID"></key-property>
    <key-property name="ICAO"></key-property>
</composite-id>
```
Then, we have mapped the first attribute but to map the others we use the element `property`. It serves for link the properties that are not part of the primary key with the columns of one or more tables.

Like we have seen in the `id` label the attribute name indicate the name of the property, the attribute column, the name of the column of the table `Personale` and the attribute type show us the Hibernate type.

We can also include the attribute `unique`, who let us specify if we want duplicated values or not. Also is possible to set the column to be not-null setting to true the `not-null` option.

```
<property name="matricola" column="MATRICOLA"></property>
```

Now in the next points we explain more difficult concepts like inheritance and relations between the tables.

### 5.3.3.2 Inheritance mapping

Mapping a hierarchy of classes to tables can be a complex issue, and we're going to talk about the different ways to do this job. The most important think we have to remember is that SQL-based models provide only as a relationships between entities, SQL database management systems don't support type inheritance. In general, we have chosen the table per class hierarchy strategy, that strategy is the simplest one to work like in a polymorphic way. In this part, we're going to explain different ways to do this mapping and the benefits from the different strategies.

**Table per class hierarchy**

An entire class hierarchy can be mapped to a single table. This table includes columns for all properties of all classes in the hierarchy. The concrete subclass represented by a particular row is identified by the value of a type discriminator column.

This mapping strategy is a winner in terms of both performance and simplicity. It's the best-performing way to represent polymorphism -both polymorphic and non polymorphic queries perform well- and it's even easy to implement by hand. Ad-hoc reporting is possible without complex joins or unions. Schema evolution is straightforward.

There is one major problem: Columns for properties declared by subclasses must be declared to be nullable. If the subclass define several non nullable properties, the loss of NOT NULL constraints may be a serious problem from the point of view of data integrity. Another important issue is normalization. We've created functional dependencies between non key columns, violating the third normal form. As always, denormalization for performance can be misleading, because it sacrifices long-term stability, maintainability, and the integrity of data for immediate gains that may be
also achieved by proper optimization of the SQL execution plans.

To define this type of inheritance we use the label `subclass`, and inside of them we can use the same attributes and options that we use at `class` label. Also we add a new option in the label that is called `discriminator-value`. In the top class we have specified a discriminator to let the database select between the different subclasses. In the subclasses we can specify the value that this type of subclass may have in the database table.

This strategy is good for polymorphic associations or queries, and subclasses declare relatively few properties. Is recommended if we use this strategy to minimize the number of nullable columns and be sure that the denormalized schema won't create problems in the future.

**Table per concrete class with unions**

We can use exactly one table for each non abstract class. All properties of the class, including inherited properties, can be mapped to columns of this table. We firstly map the shared properties of the main class and after the different ones. Let's describe the mapping file and finally we'll describe the other types of inheritance mapping that we haven't used.

An abstract superclass or an interface has to be declared as `abstract="true"`: otherwise a separate table for instances of superclass is needed. In this approach we usually have a common identifier that is shared for the concrete classes of the hierarchy. At the beginning of the file we declare de properties of the superclass and these properties are inherited for all the concrete classes. This avoids duplication of the same mapping (In other strategies like table per concrete class with implicit polymorphism we duplicate the code).

If you don't require polymorphic associations or queries, the table per concrete class is the best. There are to types: implicit polymorphism and union-based (the one that we've used). Implicit polymorphism is most useful for queries utilizing non-persistence-related interfaces. An explicit union-based mapping should be preferred, because (optimized) polymorphic queries and associations will then be possible later.

**Table per subclass**

If you do require polymorphic associations or queries, and subclasses declare many properties we can use table-per-subclass. The main idea of this approach is that we have different tables for the superclass and the concrete classes and the inheritance is represented by foreign keys between the class and the subclasses. This approach is not used in our project.

**5.3.3.3 Mapping of collections and entity associations**

Another important part of the mapping is to map the relationships between the tables
of our database but if we want to talk about this firstly we must talk about the mapping of collections.

Java has a rich collection API, and we can choose the collection interface and implementation that best fits our domain model design. We're going to see the collections that we've used in the project. Also we've to consider that the collection we're going to use has a parallel hibernate class that allows us to use it.

Out the box, Hibernate supports the most important JDK collections, maps, and arrays in a persistent fashion. Each interface has a matching implementation supported by Hibernate, and it's important that you use the right combination. Hibernate only wraps the collection object you've already initialized on declaration of the field (or sometimes replaces it, if it's not the right one). We've used two collections Set and List.

A java.util.Set is mapped with a `<set>` element. We initialize the collection with a java.util.HashSet. The order of its elements isn't preserved, and duplicate elements aren't allowed. This is the most common persistent collection in a typical Hibernate application. We add to the set label the attributes name (the name of the set) and table (the name of the related table on the database). Is possible to set the lazy initialize. This initialize allows the objects to be charged on request and don't be charged all together. It's utile to optimization of searches. The inverse attribute make the collection as the end of a bidirectional association. Is usually used in the many-to-many relationships. We can allow the cascade operations setting cascade at true. Inside this label we've to add the key label to define the key property to make the association.

A java.util.List can be mapped with `<list>`, preserving the position of each element with an additional index column in the collection table. Initialize with a java.util.ArrayList. In the lists we've used the same properties as in the set but we add a label list-index to select the column that will be the reference for sort the list.

Now we're going to explain the associations.

**Many-to-many**

In this association we've two classes A and B. An element in A have a set of child elements in B, and a element in B have another set different or equal of elements in A.

This structure can be designed making a intermediate tablet that links the codes of the elements in A with the elements in B. It is clear therefore that a collection many to many may be mapped in a separated table with the keys of the other tables as a foreign keys. This can be mapped as follows:

```xml
<set role="setOfB" table="A_By_B">
  <key column="A_id"/>
  <many-to-many column="B_id" class="elementOfB"/>
</set>
```
In this point we don't have an extra column in B that says the elements in B that correspond at a element in A. Instead we've a new table A_By_B that have the pairs of related keys from A to B and from B to A. To make it bidirectional must be declared in the mapping of the class B as follows, and we declare it as the final of the relation between the two tables. Whatever parameter, like in the collection can be used there (lazy, cascade...)

<set role="setOfA" table="A_By_B" inverse="true">
  <key column="B_id"/>
  <many-to-many column="A_id" class="elementOfA"/>
</set>

One-to-many
This relation loses some semantic concepts of the collections in have:

- No null value can be contained in a map or set.
- An instance of container can't belong to more than one instance of a collection.
- An instance of the contained entity can't appear in more than once in the index of a collection.

Like in the first case if we wasn't have a relation one-to-many between two tables, we must map properly the two classes. In one we must create a relationship one-to-many and in the other a many-to-one. An association one-to-many from A to B requires a new field in B with the index value of A which is associated. We don't need a new field in the A table, let's see an example:

<set name="setOfB" table="B">
  <key column="A_id"/>
  <one-to-many class="B"/>
</set>

One-to-one
We can use the attribute unique in the one-to-many label to convert the one-to-many association into one-to-one association.

Common considerations
It should be noted certain characteristics of the collections in Hibernate:

- Lazy initialize: When we define a collection as lazy we get the data load from the DB is in demand for them. For example, if an 50000 objects collections we only need to search in the first ten elements it has no sense charge all. Because of this the objects that are returned at the iteration of the collection are charged when we access to him. It can make some troubles to us (for example when we
go to another class).

- **Ordered collections:** Hibernate supports the implementation of ordered collections through the interfaces java.util.SortedMap and java.util.SortedSet. If we want we can define a comparator in the collection definition. The allowed values are natural, unsorted and the name of the class that implements java.util.Comparator.

- **The garbage collector of the collection:** The collections are automatically persisted when they're referenced by a persistent object and also are cleared automatically when they stop being.

Finally if a collection is moved from a persistent object to another one, his elements are moved from one table to other. The good thing of Hibernate is that we don't need to worry about this, we must use the collections like we usually did.

### 5.3.4 How to use hibernate

In every application that uses Hibernate appear four basic objects:

- **Configuration:** this object contains the needed information for connect to the database. It must read the file Hibernate.properties. Also process the information for mappings. It reads and verifies the mapping files.
  
  ```java
  Configuration conf = new Configuration();
  ```

- **SessionFactory:** creates the Sessions. An object Configuration is able to create a SessionFactory due to he has every needed information. Normally an application only has a SessionFactory.

  ```java
  SessionFactory sessionFactory = conf.buildSessionFactory();
  ```

- **Session:** The main interface between the Java application and Hibernate. Is who maintain the conversation between the application and the database. Allow add, modify and delete objects from database.

  ```java
  Session session = sessionFactory.openSession();
  ```

- **Transaction:** As its name indicates, is responsible for the transactions. It allows to define units of working.

  ```java
  Transaction tx = session.beginTransaction();
  [...]
  session.save(object);
  tx.commit();
  [...]
  tx.rollback();
  ```

Let's look better how we do this. First of all we create and open the session as follows:

```java
Session session = HibernateUtil.getSessionFactory().openSession();
```

The class HibernateUtil is a class that we have made previously it creates the
configuration and makes simple the working of create a session. The method getSessionFactory return a session factory with the corresponding configuration. Finally we open a new session. Now we can start to work with the session. We can make queries, insertions, updates, deletes... but is better to create different units of work called transactions.

We start our transaction with the command Transaction tx = session.beginTransaction() into this block we can execute all the instructions that will be executed at the final of the transaction indicated by the instruction tx.commit(); after this if there's no error the instructions will be executed.

When we've finished with all of our transactions and instructions we must close the session with the command session.close();

To insert in the database a new object we use the instruction session.save(object); but is better to use the method session.saveOrUpdate(object); lets see.

Difference between hibernate's save, update and saveOrUpdate() methods.

Hibernate has set of methods for saving and updating the values in the database. The methods look like same and difficult to differentiate between them if you are not understanding them clearly.

- **save** - save method stores an object into the database. That means it insert an entry if the identifier doesn't exist, else it will throw error. If the primary key already present in the table, it cannot be inserted.

- **update** - update method in the hibernate is used for updating the object using identifier. If the identifier is missing or doesn't exist, it will throw exception.

- **saveOrUpdate** - This method calls save() or update() based on the operation. If the identifier exists, it will call update method else the save method will be called.

For make queries we can do it in different modes. Now we're going to show an instruction that makes a query directly in SQL

List <AssegnazioneMezzi> l = (List <AssegnazioneMezzi>) session.createQuery("select * from a_mezzi").addEntity(AssegnazioneMezzi.class).list();

This instruction first executes the query select * from a_mezzi and takes a list of results and converts all of them into the class specified in .addEntity(). After that we can do two things, the first one is to take a list like in the example with .list() the other option is to have only one result with changing .list() for .uniqueResult(). For the assignment to the class is better do a cast (List
this ensures us that the method is correct.

5.3.5 Considerations in the classes

To let our classes work in hibernate we have to satisfy the unique condition of Java Beans, the properties of the persistent object must implement the getters and the setters. Also is strongly recommended to use a void constructor, because hibernate uses it in the queries.

5.3.6 Conclusion

Hibernate can be a useful tool to help us in the communications between the Java application and the database but the design of all parts have to take into account that we're going to use Hibernate because are some limits in the explanation of the mapping.

It has been some difficult to do this from an existing project because the project was formally correct but in fact it wasn't so useful.
5.4 Making an interface: Swing

Swing is the primary Java GUI widget toolkit. It is part of Sun Microsystems' Java Foundation Classes (JFC) — an API for providing a graphical user interface (GUI) for Java programs.

Swing was developed to provide a more sophisticated set of GUI components than the earlier Abstract Window Toolkit. Contained in the package javax.swing. Formed "java.awt." Allows an interface tailored to each OS without changing code. Is easy to use.

5.4.1 Classes of the Swing toolkit

Swing uses so many classes to do the interfaces. The schema to understand the main ones is the next.

![Swing Classes Diagram](image)

The most used components inherit from java.awt.JComponent. JFrame will be the base for the main application. JDialog builds the dialog windows. The classes
inherited from Jcomponent will be components. The Sub-windows are used for a better organization of the components in the program. We won't use the part of JApplet. All components allow you to set tooltips. Let's take a look to different classes that can be useful for our application.

- **Jbutton**, is a button that can contain text, images or both. The text will be putted on the centre, and the images can be at left or under the text. Let's see the most important methods of this component
  - `setText("Texto");` sets the text on the button.
  - `setTooltipText("Tooltip");` sets the help messages.
  - `setBackground(new Color(R, G, B));` sets the background colour.
  - `setForeground(Color.color);` sets the foreground colour
  - `setIcon(new ImageIcon("ruta");` puts a image on the button.
  - `setFont(new Font("tipo", estilo, tamaño));` sets the font of the text in the button.
  - `setBounds(new Rectangle(posX,posY,tamX,tamY));` set the size of the button.

We can also use the same methods changing set to get, and we obtain the value that we need.

- **JToggleButton**, this button have the same characteristics than JButton but it have two states on and off. It can be used in two ways, like checkboxes in independent mode or like radiobutton in exclusive mode. We add two new methods to the existing in JButton:
  - `isSelected();` for know if is selected or not.
  - `setSelected(boolean);` for set its state as selected.

We can use the groups for set the exclusivity.

- **JCheckBox**, has the same properties and methods than the JToggleButton but
looks some different.

- **JRadioButton**, allows to select an unique option between a group of options. Only can be selected one option once. Have the same function than JtoggleButtons but it's better to use radio button on dialogues and the other on toolbars. The methods are similar to the other ones. This components like the check boxes usually appear into a group with an identifier legend. Swing uses a panel for this purpose called ButtonGroup.

![Radio button example](image)

- **JComboBox**, this component allows us to select between a list of different options which are exclusive between them. Can be editable or not editable. We've three interesting methods on this component.
  - *SetEditable(boolean)*; allows or not the option of make editable the options of the combo box.
  - *addItem(object)*; add an item to the combo box.
  - *GetSelectedItem()*; returns the selected item.

![ComboBox example](image)

- **JList**; shows a set of items of text, graphics or both. Allows three types of selection: single item, simple range or multiple range. For this purpose we use
the method `setSelectionMode(ListSelectionModel.SINGLE_SELECTION)` we can substitute `SINGLE_SELECTION` by `SINGLE_INTERVAL_SELECTION` or `MULTIPLE_INTERVAL_SELECTION`.

- `Jlabel`, show text, images or both, and it's only for reading. Must be disabled if the corresponding component is disabled.

- `JtextField`, shows an editable line of text. For edit this line of text we can use the method `setText("Texto")`.

- `JpasswordField`, is like the text field but hiding the characters introduced by the user. We've new methods for this component. We can change the character for hide the letters using the method `setEchoChar('char')`. For read the introduced password we can use the method `getPassword()`.
- *JTextArea*, rectangular space to look and edit multiple lines of text. We can put it into a JScrollPane to get the scroll bars.

- The menus go in the main application, is possible to assign to a graphic. There are different types, one type is Drop-Down is the typical menu that we see in the windows, for example Edit menu. Other type are the sub-menus which exit form other menus. The jPopupMenu is the last type of menu that we can find and is the typical menu that we can take pressing the right button of the mouse. For this purpose we must use this hierarchy of classes in this order jMenuBar, jMenu and jMenuItem.

- *JToolBar*, this is a type of container, with buttons of commands or commutation, they can use graphics. It use the method setFlotable(boolean) to set if we can move the bar. And addSeparator() to add a separator between the buttons.

- *JPanel*, is the container that group the components into the window. We use layouts to put them where we want.

- *JTabbedPane*, is the same that the panel but can be organized by tabs.

We use layouts to organize the different components into the containers and set their size. First we create the container and set its layout, after that, we can add the
components that we want. Let's see some types of layout:

- Flow Layout, is the most simple, the components are added in one or more files from left to right and from top to down.

- Border Layout, it uses five areas to organize the components: north, south, east, west and centre. If someone isn't busy the other be expanded.

- Grid Layout, we create the container with a determinate number of lines and columns. The size of the cells are the same. The components been placed from left to right and from top to down.

- Grid Bag Layout, the same as grid layout but with no limit of cells.

- Box Layout, it allows to organize the components in vertical or horizontal, without space between the components.
6. Functionalities: How to use the application

In this part we're going to take a look to the application and know how to use it. First we must enter the user and the level of skills that the user have. If it's correct we can enter a window to select the functionality that we want.

We're going to know all the functionalities according to the different users and the functionalities which can be used by all the users. First let's take a look to the common functionalities.

6.1 Common Functionalities

6.1.1 Enter to the application

The first thing we've to do is enter the application putting our name, surname and password (by default is our Codice Fiscale) and select our type of user. Then we enter to the different menus of the three types of users: operator, land controller or flight controller.

After this we can select a functionality from the combo box and then press Next.
6.1.2 Report of airport structure

Now we're going to see the first function. All common functions are used to show the state of the airport, in this case of the structures in the airport. This window can be useful for select the better runaway in a landing or take off operations, for select the best structures in a road, for do the maintenance of the different structures, etc. After clicking on Next the application shows us a screen like this one.

We've two different lists on this screen. The first one shows us the operative runaways in the airport structure. The first number is the identifier of the structure, after them we've the date of building of the runway, and the confirmation of the operability. The fourth column shows the material which we've used to construct the runway. And finally the last one tells us if the runway is in good or in poor condition.

The second list shows other types of structures that are present in the airport. The list have the same fields as the first list. We can see the identifier, the date of building, the operability, the type of surface, the condition of the surface and finally we've defined the type of structure (not runway).
In the example we can see three runways, on the first list, which are built in 24/12/1986, operative, built with MACADAM (a type of cement) and in perfect condition. On the second list we can see a structure of a taxiway, with the identifier 669, built in 8/10/1988, operative, built with MACADAM, and in a good condition.

6.1.3 Report of means of support

The next one functionality shows the user the different supporting means that are present in the airport structure. They are divided in two parts the first one for operative means and the other for not operative means. This functionality is mainly aimed to the operator but can be used by everyone. Is useful for take a look to all the means before doing a mean assignment, to know which means may be repaired, to do inventory, etc.

On the first list we can see the operative means. The first example on the list has the identifier number 6, is free, its model is Seat built in 1945, it starts working at the airport in 1980 and is a Bus. The second one has the identifier 12, is free for use, the model of mean is Alpha, its built in 1977 but starts to work in 1978 and is an anti-fire mean. The last one of the first list have the identifier 18, in this moment is busy, is a Seat built in 1989, started working in 1990 and is a disembark mean.

On the second list are the non operative means, we can see in the second column. The first column is the identifier of the mean we can see 15, the model of the example is Alpha built in 1977 and started working in 1978, is an anti-fire mean.

6.1.4 Report of airport movements

In the last report we can see all the flights that have done any movement in the airport in different periods. There are three parts one for the current day, another for the
current month and finally one for the current year. Is useful to organize all the movements in a airport structure.

On the example we can see the first two lists empty because there aren't any flights introduced on this day or month. In the last list we can see different flights introduced. The first example is the flight number 2 in the 26th of July of 2010, the number of the airplane is 7 and his company is Vueling. The flight have finished. Is an arriving flight and comes from LIRA, that's the code for the airport Roma Ciampino. We can see at the third example that the state of the flight is in process, it means that we haven't introduced in the system the landing of the plane. The airport is also different and is Valencia Manises LEVC.
6.2 Flight Controller Functionalities

The flight controller have a particular functionalities because of his knowledges, is who better knows the process of landing and taking off of the planes, let's see his particularities.

6.2.1 Assignment of landing runway

An airport has an aerial zone under his jurisprudence, when a flight arrives to this area it may communicate if it's going to land into the airport. Then the pilot of the plane inform to the flight controller about the information of the plane, the flight, and his coordinates. In this moment the flight controller starts introducing the data into the system and decide the best option to do the landing of the plane, and finally introduces his decision into the system and send a message to the pilot to know the runaway in which he have to land. Let's see how works the application in this issue.

First show us a window with a combo box to select the company of the plane. We select the company and press Continue.
Now we've two options the first one is select a plane that we've introduced in the database, we can search the model and the number in the database, select them and press Next to continue. The second option is to insert a new aeroplane into the database we press new and go to the window to insert the new plane.

The window we can see up, have two parts the first part for introduce a new aeroplane and the second part to introduce the arriving flight. The second part is common with the insertion of flight with a existing aeroplane so let's explain first the first part and after we see the second part with the other example. In this window we must introduce the ID, it is the unique identifier for the plane. The next field is Sigla, there we introduce the code ICAO for the model of the plane. For example B747 is Boeing 747.
Now let's see how to introduce the data of the flight. We can see the information about the plane on the top of the window and check it. The first field we can see is the number of the flight. The second field is the date of the flight. It automatically puts the date of today, but can be modified. The third field is the origin of the flight, it's recommended to put it on ICAO four letters code. And finally we have a combo box to introduce the type of flight in this case a passengers flight.

After that the controller must select a Runaway between the available ones. The occupied runaways doesn't appear in this list. We click on a element of the list and after that we press next.
Finally we get a confirmation of the insertion. We can check the correctness of the insertion on the reports or using the MySql console. In this case we can use this report to check the insertion.

6.2.2 Assignment of take off runway

When an outgoing flight is programmed a flight controller must select a runway for the take off. He knows the destination of the flight and takes the information about the flight to decide the better runway.
First select the parked flight which is going to take off and press next.

![Image of a window with a table showing ID, Latitude, Longitude, and Orientation. The table entries are:
1 | 40.0 | 50.0 | 60.0
2 | 40.0 | 50.0 | 60.0
3 | 40.0 | 50.0 | 60.0

Next button is visible.]

Then we select a runway for the take off and press next.

![Image of a window showing a report with details such as Compagnia: Air Nostrum, Aeromobile: B747 101, Volo: 12 del 17/8/2016, Runaway: 3 | 40.0 | 50.0 | 60.0, and a Save button.]

Now we've correctly inserted the outgoing flight.
6.3 Land Controller functionalities

The land controller is the responsible of guiding the planes inside of the airport structure. His main purpose is to guide the planes from the runways to the parkings. Let's see his functions into the system.

6.3.1 Landing route

When a plane is arrived to the runway of the airport it needs an assignment of a parking and a route to go from its runway to the parking. The pilot of the plane request a parking and the land controller introduces it into the system and communicates the route to the pilot Let's see how to do this.

First of all we've a list of possible routes from our runaway to other parts of the airport we select one and continue.
Now we can select the parking between the free ones to finish the route.

![Confirmation Message](image)

We get a confirmation message when is inserted in the database.

### 6.3.2 Take-off route

When a take-off runaway is assigned to the plane we must take a route from the parking to the runway. When the pilot knows his runaway it request the route from his parking to this runaway and the land controller introduces the information in the database and send the information to the pilot. The process to do this is the next.
We select the flight we want and then press next.

The system assigns us the best route.
6.4 Operator functionalities

The operators must manage the means to support the airport issues. Let's see the functionalities of the operators.

6.4.1 Assign support mean

We can assign a support mean to a plane when the plane is arrived to the correct hangar. The operator first introduces the information into the database and after starts his work. We're going to explain how to make it.

First of all we select the flight we want.
Then we select one or more means to assign to the flight. We can select more than 1 pressing Ctrl+Click and holding Ctrl button. When we've finished the selection we press next.

We get a confirmation of the insertion.

6.4.2 Insert support mean

We can insert into the database the new support means that we have in our airport. The procedure to do it is the next.
This window is to introduce the data of the new support mean. On the first field we introduce an identifier, if it's repeated we get an error message and we need to insert another one. The second field is the combo box and we can insert the operative state of the mean which can be free, used or non operative. The third field is the model of the mean. The fourth is the construction year and the fifth is the year when started to work, both have to be integers or we get an error message. The next combo box is the type of mean, according to the type we insert in this combo box we will get different types to choose in the subtype combo box.

Finally we get a confirmation window of the insertion.

6.4.3 Modify support mean

Now we're going to know how to modify the operative state of the mean.
The first window we get is a window to search the mean that we want, we must find all the means that at least have an equal field. We complete all the options that we want and then press Search.

We obtain all the support means with all the information, we select one of the list and then press select.

We can check the information and if all is correct we can press continue.
Then we've the window to modify all the fields we want except the ID which is invariable.

![Image of modified window]

We check the information that we've changed and then press Save button.

![Image of loaded information]

6.4.4 'Delete' support mean

When we've a support mean that we're not going to use more, because is broken or obsolete, we can cancel them. We won't delete them from the database, because we want remember all the means that have been in the airport, but in practical effects we are doing this. Let's see how to make it.
First we search the mean that we're going to cancel. We introduce the data we want and the application will search the means that coincide at least in one field.

Then we get a window with all the coinciding fields. We must select one and press on the Select button.

We confirm our selection and then press Continue to cancel the mean.
6.4.5 Modify structure

We can modify the operative state of a structure when it is busy or under maintenance. Let's take a look how to make this and other modifications to our structures.

First of all we must search the structures. We introduce the data we want and the application will search the structures that coincide at least in one field.

We select the structure we want and then press the Select button.

ID: 5
Data Costruzione: 23/10/2006
Operatività: Operativa
Tipo di superficie: Cemento
Stato di superficie: Ottimo
We get a window off confirmation and press Continue.

![Image of confirmation window]

The we will get a window like this where we can modify the fields that we want except the ID. We make all the changes we want and then press Save button.

![Image of modification window]

We will get a confirm window and then press continue to finish the process.
6.5 Conclusion

Then we've finished the explanation of our application and we know how to use it the best as possible. The problem of aerial control traffic is a hard issue, the controllers get lots of information from the planes in the air and in the airport zone, they must make organization into this chaos.

All this information may be managed with correctness to be useful to optimize the resources and also to prevent accidents due to the uncoordinated aerial movements. In a few minutes all the arrived information have to be used into the correct way,

But the controllers need some tools to do easier this job, because without this help it can be a very stressing job. That's what we have developed this application, to simplify all this process and to store the information about all the airport structure.

Finally, I hope this application is the most useful as possible and contribute to help the user, thank you for your attention.
7. Bibliography

Problem statement

Hibernate
Slides from Antonella Poggi
http://www.dis.uniroma1.it/~poggi/didattica/progettoas09/
Hibernate documentation

MySQL
Documentation of MySQL
http://dev.mysql.com/
Slides from Antonella Poggi
http://www.dis.uniroma1.it/~poggi/didattica/progettoas09/