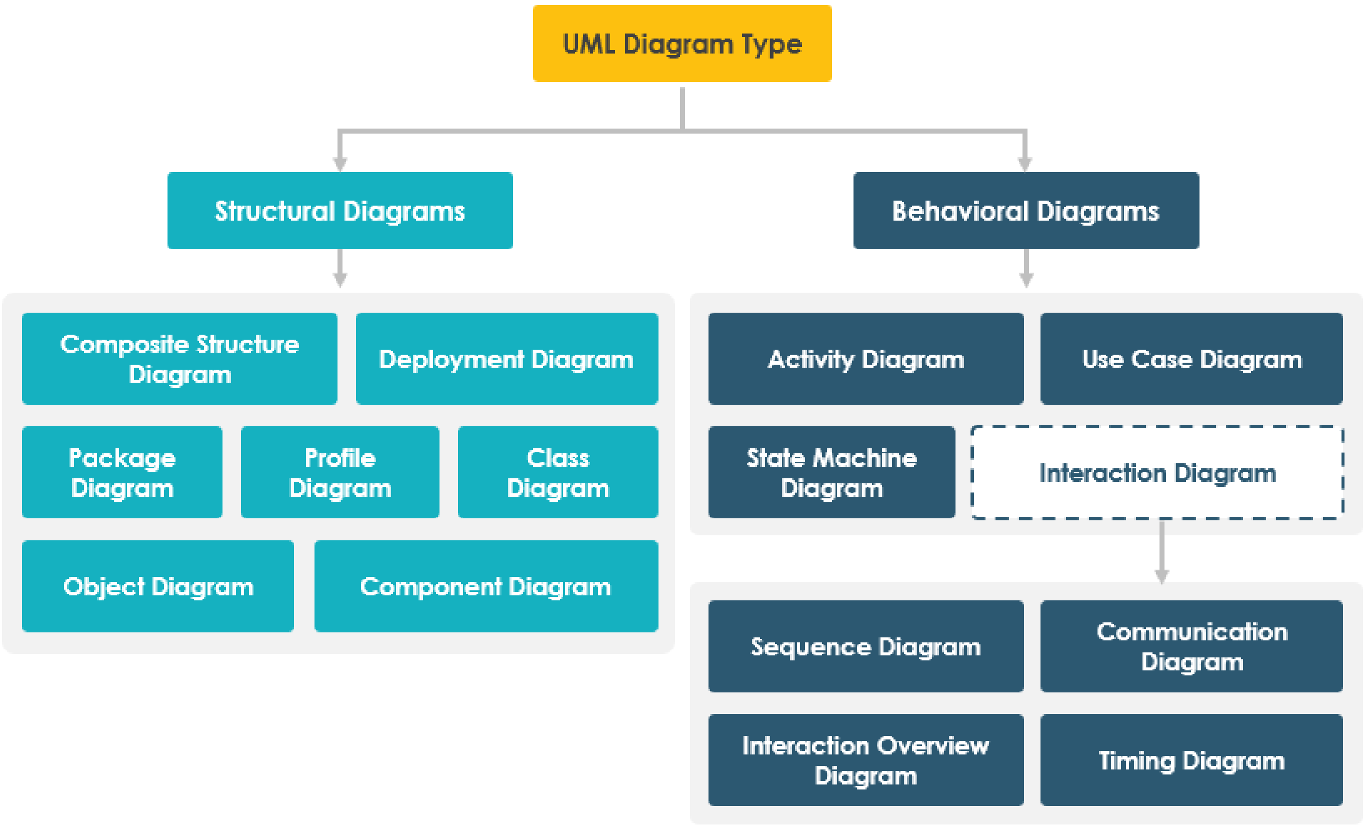
by Javier Carrasco

Development environments
Block 1
Unit 5: Application analysis. Use case diagrams

1.5.1. Introduction to UML

UML (*Unified Modeling Language*) is a standard modeling language that helps developers specify, visualize, build and document software applications. Besides, UML lets us scale these applications, and make them more secure and robust. It was originally conceived to apply the object oriented paradigm to the software analysis and design, grouping some other existing modeling languages. This way, UML 1.1 was firstly published in 1997. Its current version from 2015 is UML 2.5.

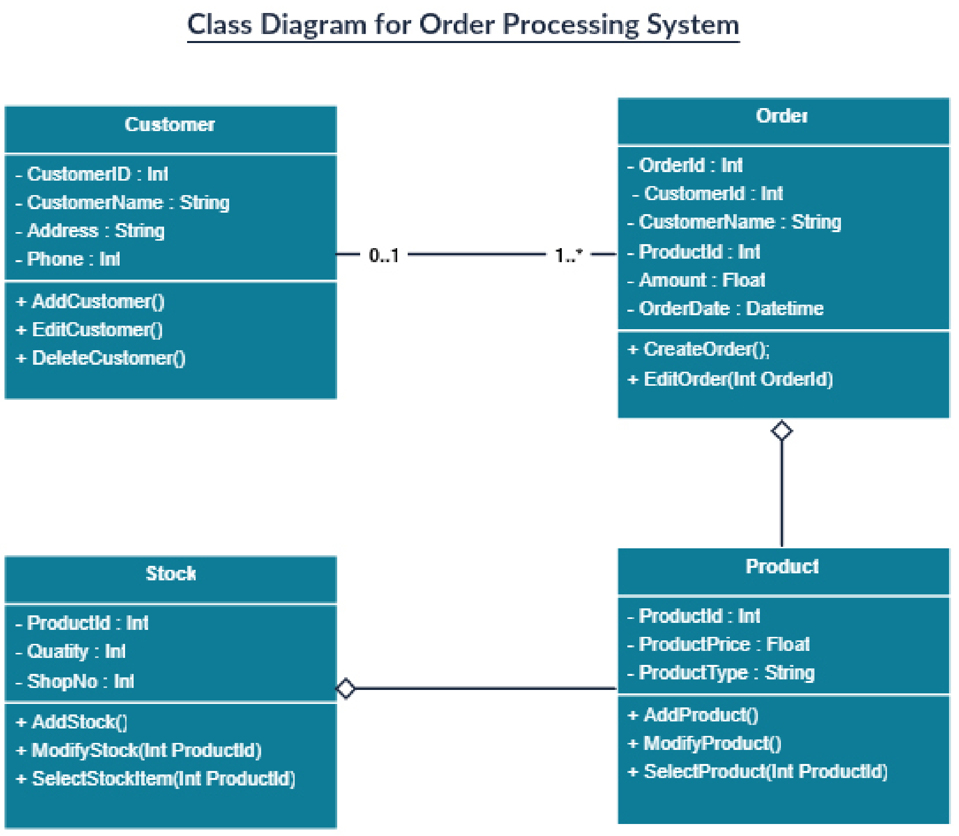
This modeling language has two main types of diagrams: **structural** diagrams and **behavioral** diagrams. Each one contains its own subset of diagrams.



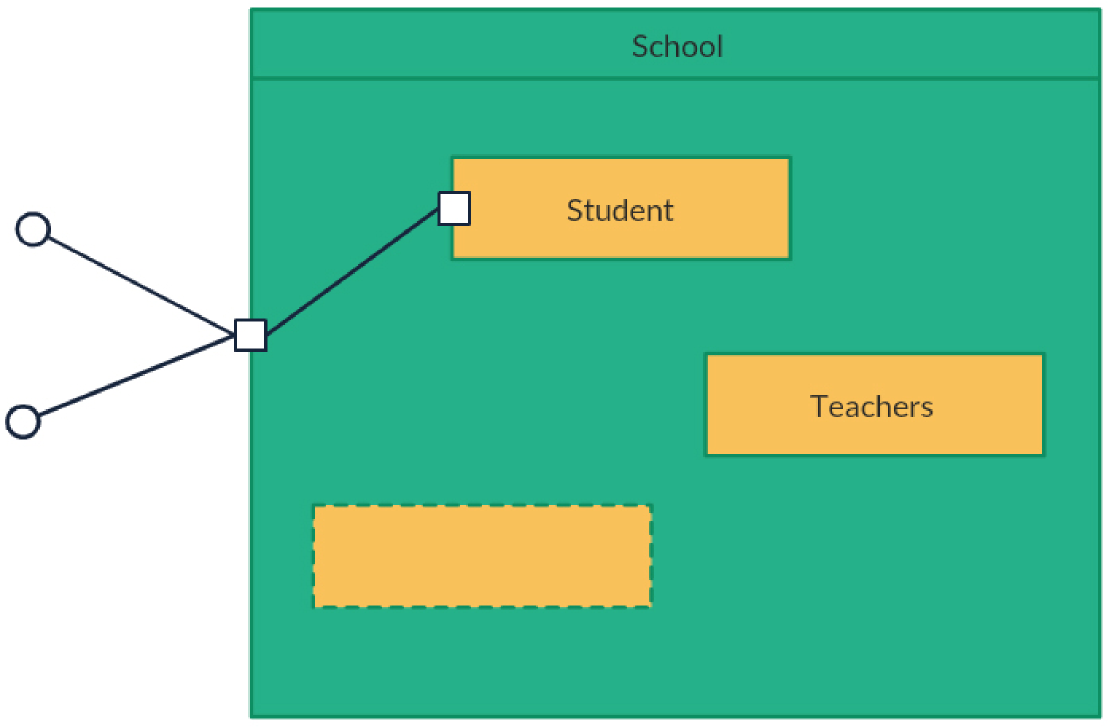
1.5.1.1 Structural diagrams

They show the static structure of the system.

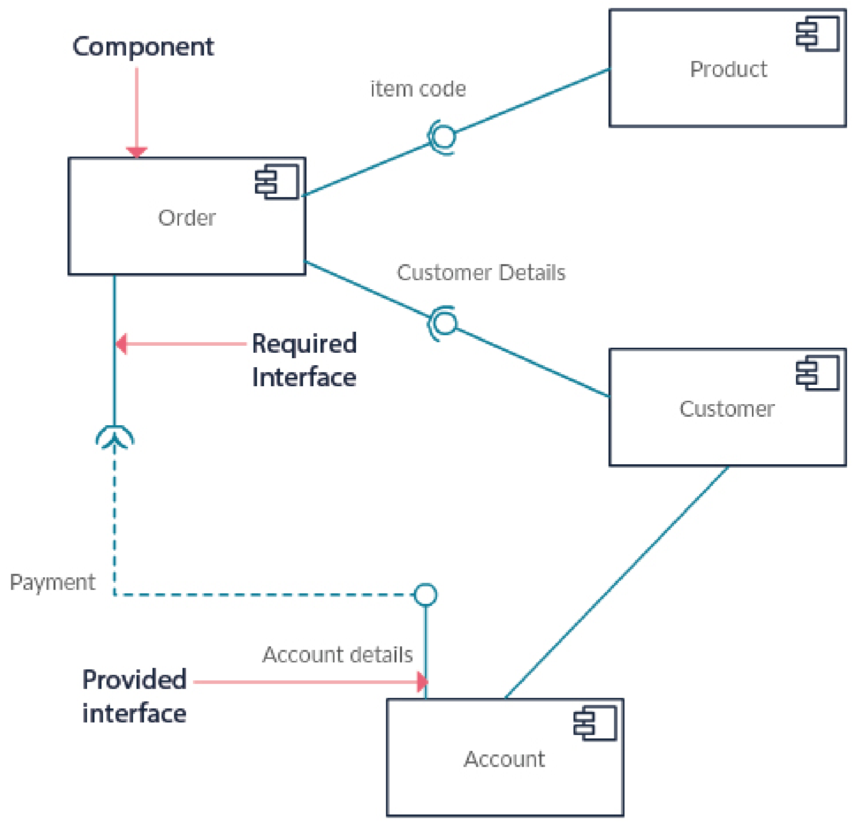
- **Class diagram**: it is the main diagram in terms of application building, for every object oriented application. It shows every class of the application with its attributes, methods and relationships.



- **Composite structure diagram**: it shows the internal structure of a class.

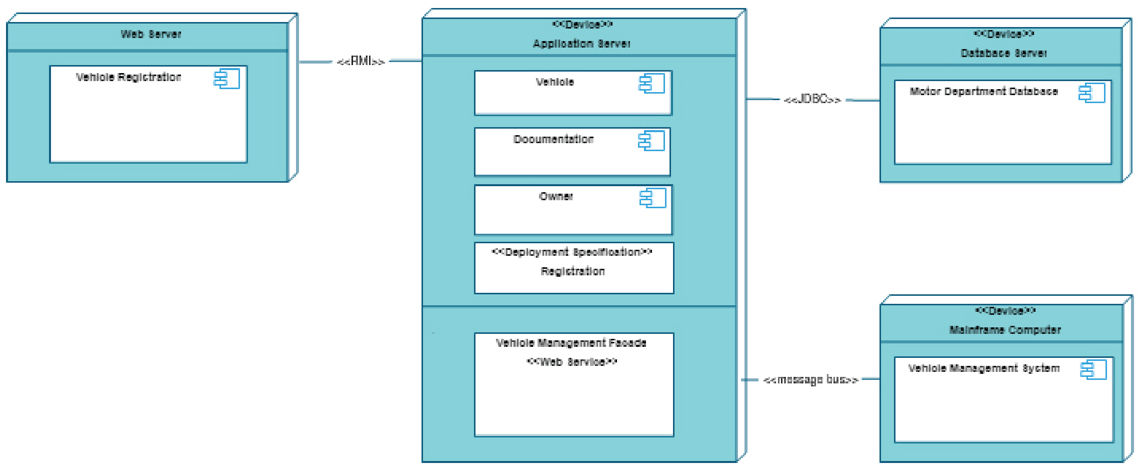


- **Component diagram**: it shows the structural relationship among the system components. It is used in complex systems with many components.

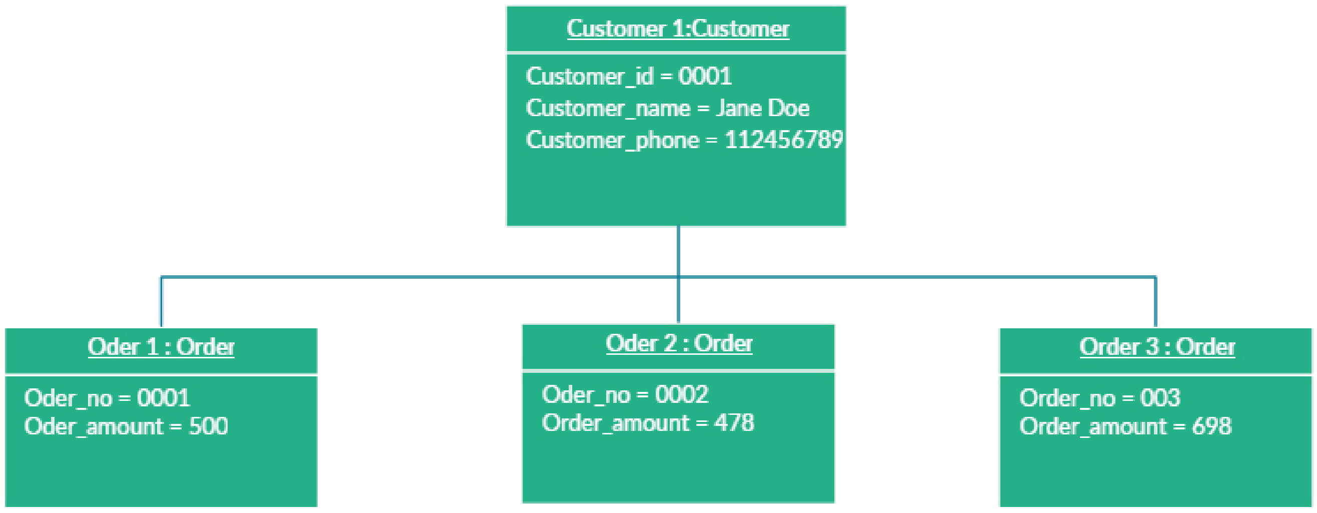


- **Deployment diagram**: it shows the hardware and software of the system, and how the software is distributed in the hardware. It can be employed in systems where the software is distributed in many different hardware components.

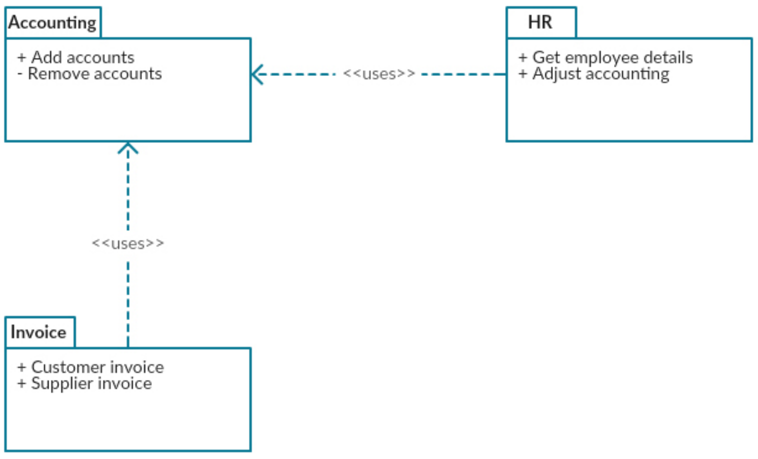
Deployment Diagram For a Vehicle Registration System



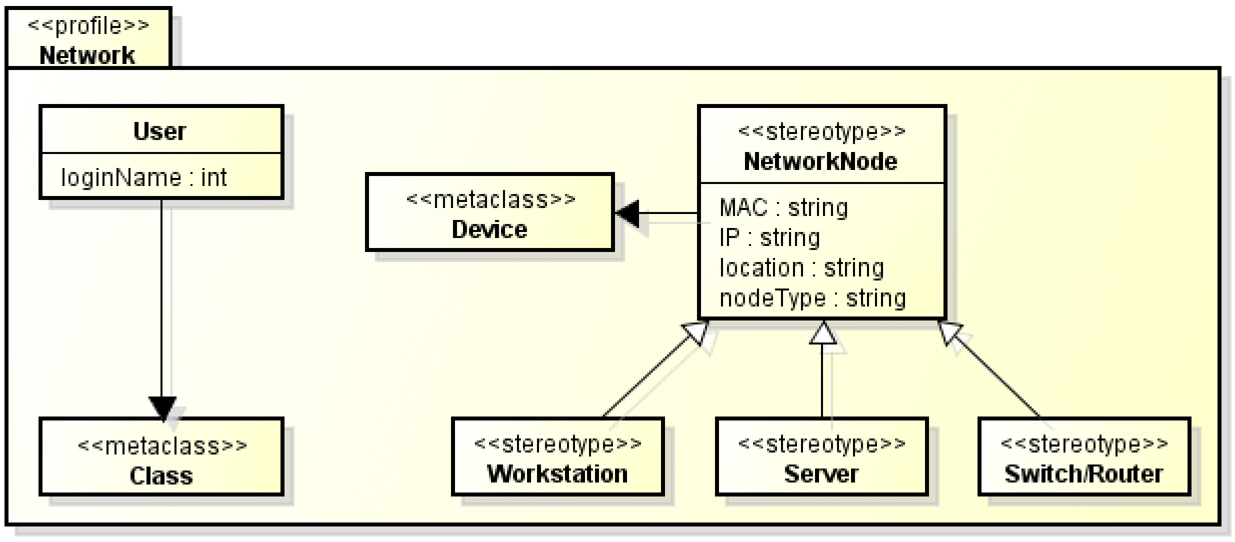
- **Object diagram**: it is similar to the class diagram, but it represent objects with example data on them.



- **Package diagram**: it shows the dependencies among the different packages of the system.



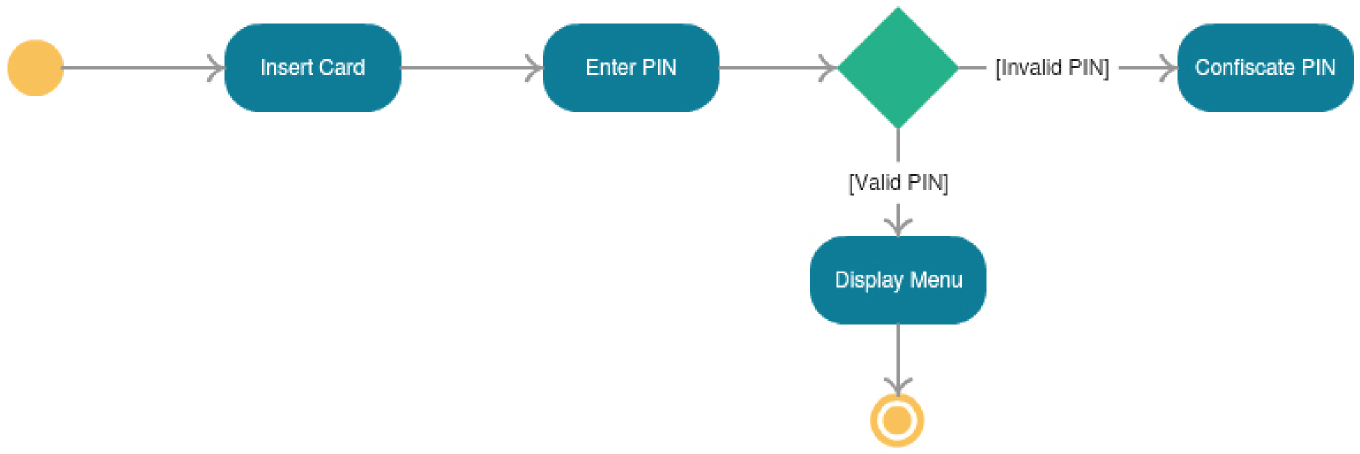
- **Profile diagram**: it lets us specify different stereotypes and tagged values to adapt the general system to specific platforms (for instance, how it should be in Java, or in .NET framework), or domains (for instance, a medical application).



1.5.1.2 Behavioral diagrams

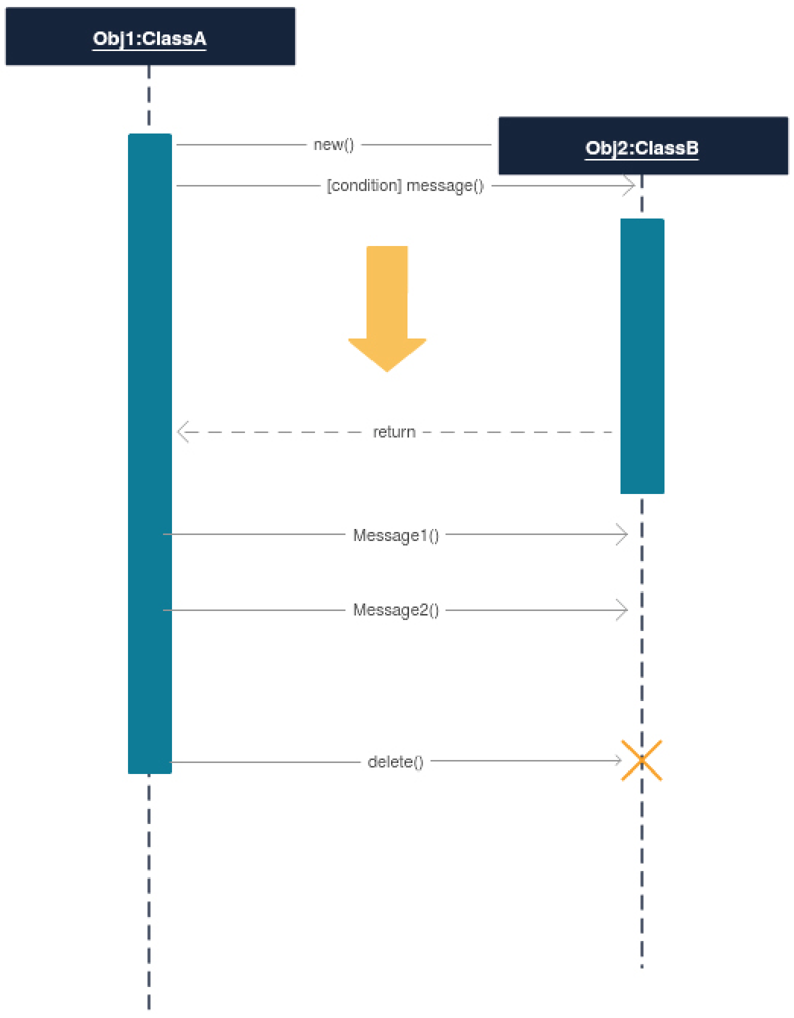
They show the dynamic behavior of the system objects.

- **Activity diagram**: it shows the control flow of a given method or component.

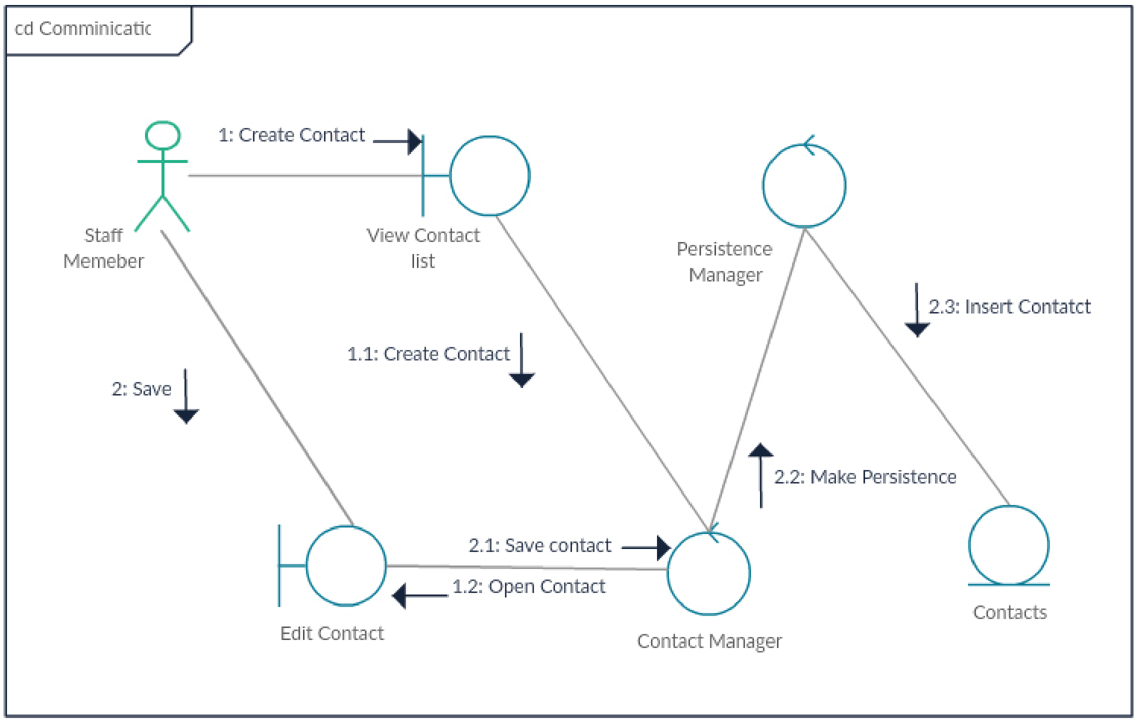


- **Interaction diagrams**: some diagrams are included here:

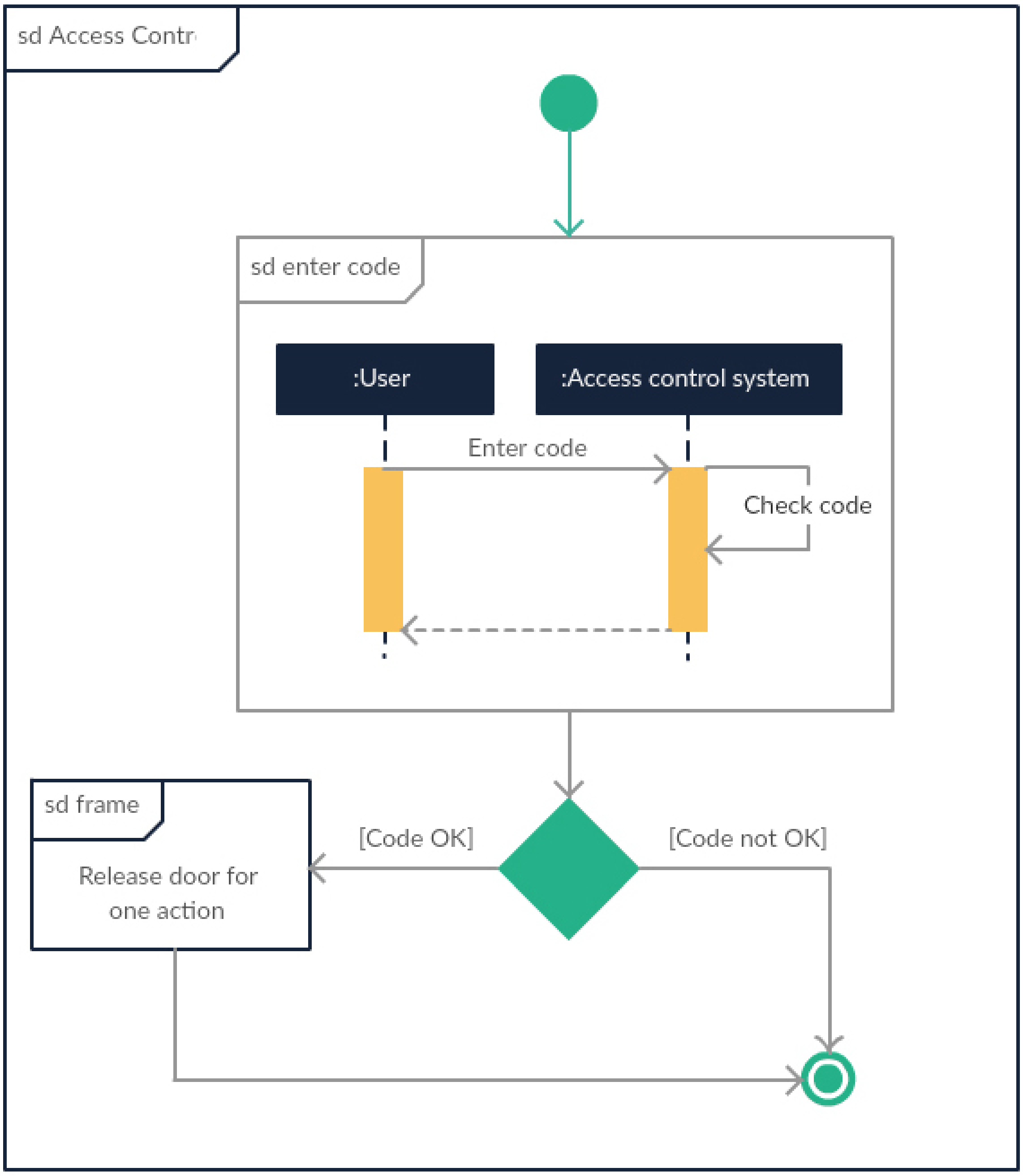
  Sequence diagram: it shows the interaction among the objects of the system, and the order of these interactions.



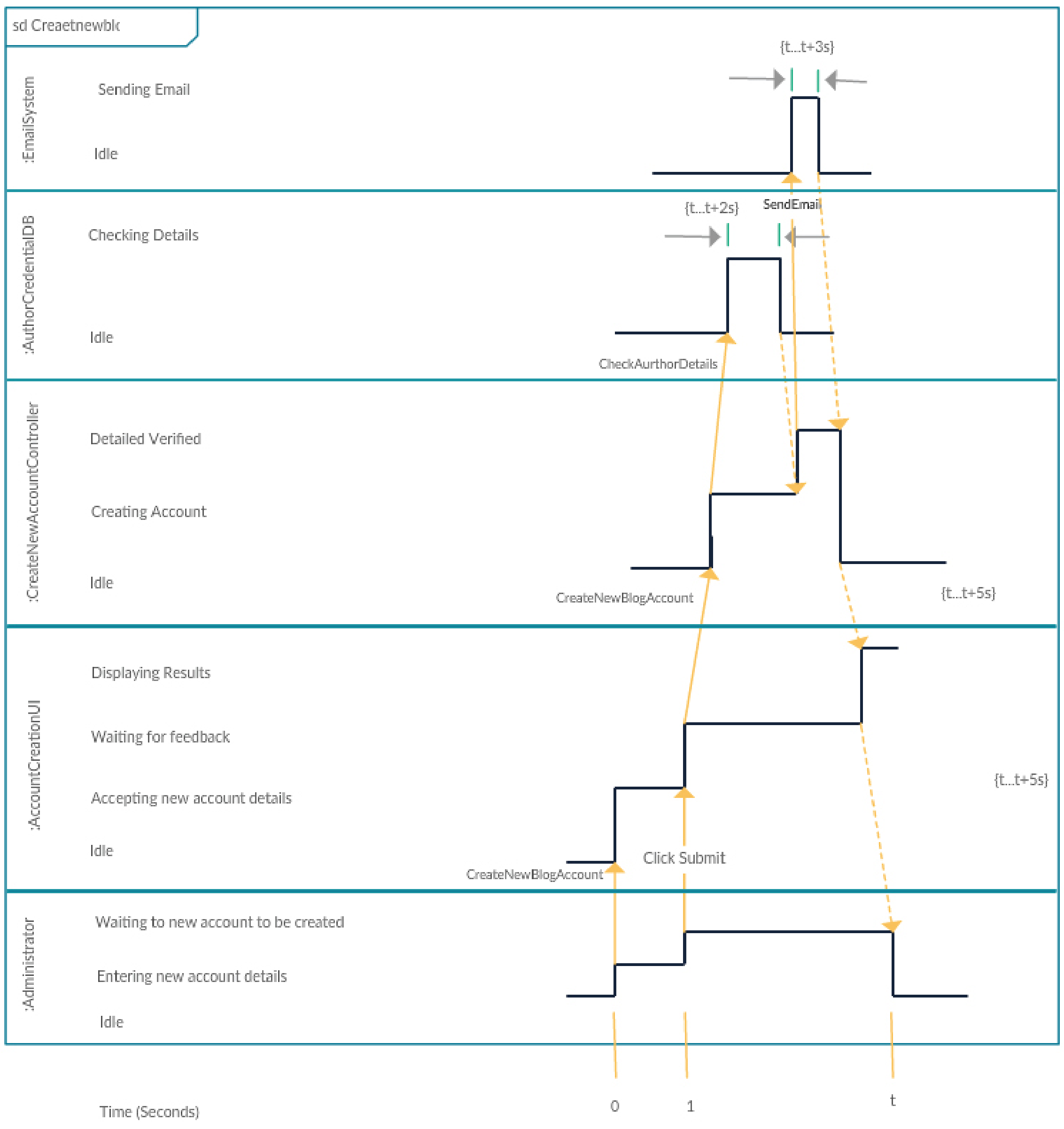
Communication diagram: in UML 1 these diagrams were called *collaboration diagrams*. It is similar to the sequence diagram, but it is focused on the messages that are passed between objects.



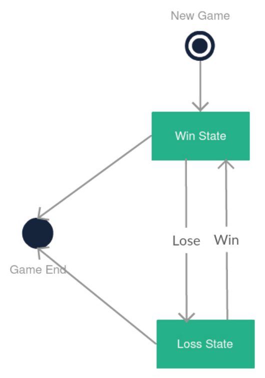
Interaction overview diagram: it is similar to the activity diagram, but it shows the sequence of the interactions.



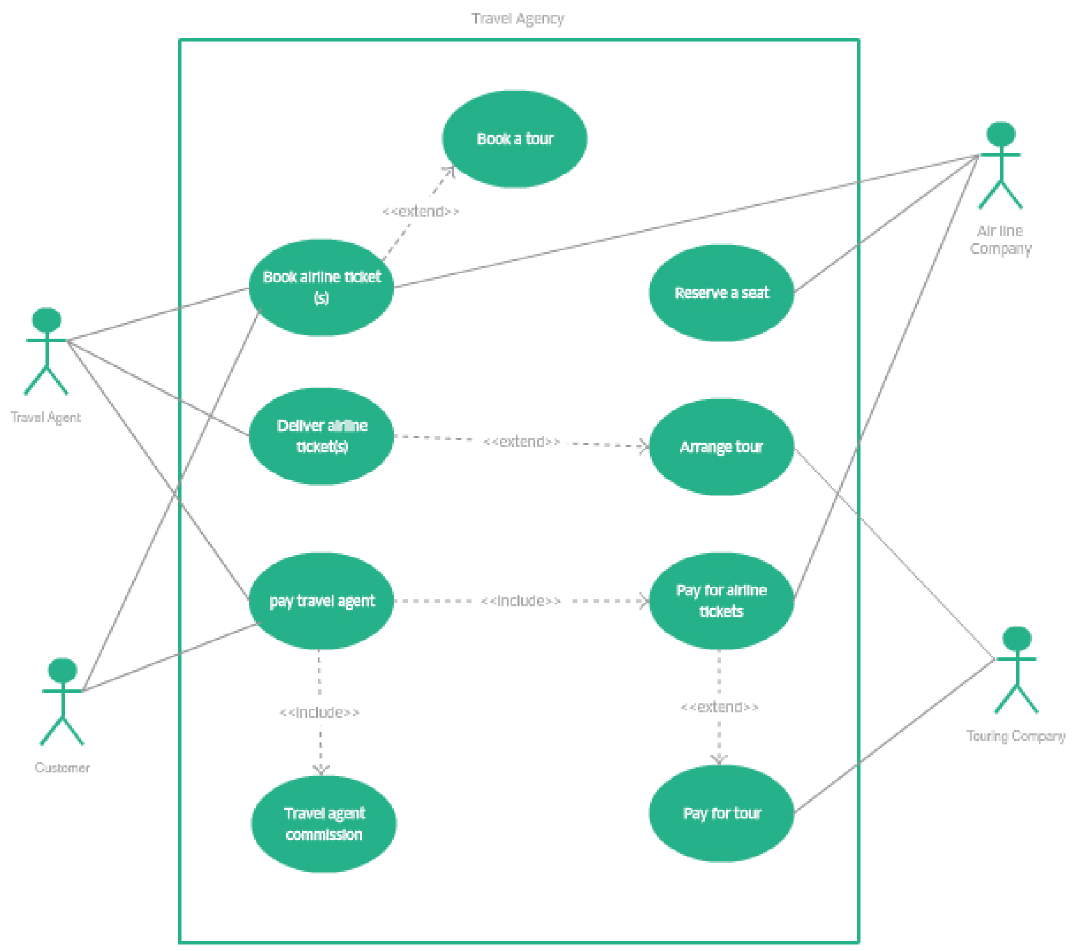
Timing diagram: it is similar to the sequence diagram, but it shows the behavior of the components in a given moment.



- **State machine diagram**: it is similar to the activity diagram, with a different notation and usage (it is often used to describe the behavior of objects which act in a different way depending on their own current state).



- **Use case diagram**: it is the most popular behavioral diagram. It provides a general overview of the actors involved in the system, and the tasks that each one can do on it.



1.5.2. Use case diagrams

Use case diagrams are a subtype of the UML behavioral diagrams, as we have seen before. The main purpose of these diagrams is to represent the possible interactions between the user(s) and the system, and also between the different parts of the system itself, and with other systems. This way, we get a general overview of the application, and a graphical representation of the functional requirements, from the point of view of the final user. It is a good guide for the development process that will come later.

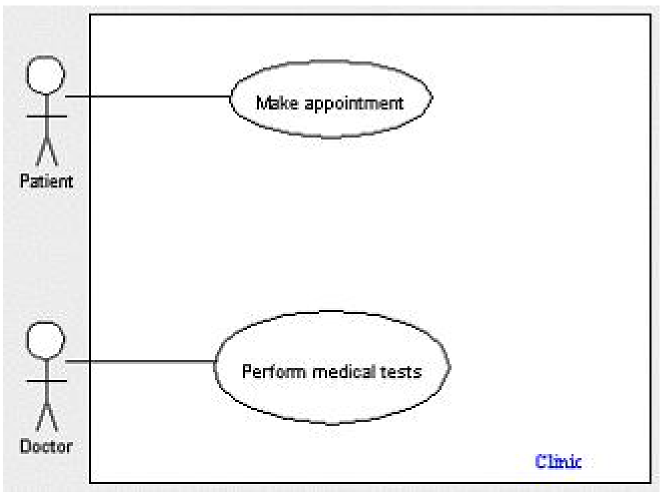
Use case diagrams will use actors and use cases to represent the use of the different functionalities or services of the system. We can use many different actors to represent different system roles, so we can see them and their permissions.

1.5.2.1. Elements of a use case diagram

This type of diagram is one of the easiest to represent, since it has just a few different elements, and it is relatively easy to place and connect them.

System

It is the rectangle that defines the system boundaries. Every use case must be placed within this rectangle, and every actor must be placed out of it.



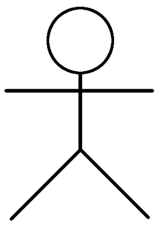
Use case

Use cases represent what an actor wants the system to do. Use cases are a set of actions with a final result. They must always be started by an actor, and they can also launch other use cases. The full set of use cases represents the whole functionality of the system.

Use cases are represented with an oval or ellipse, and they have a name inside, with a verb that indicates the action that will be run (for instance, "Print report", "Pay the bill"...). In previous example, we can see two use cases, named "Make appointment" and "Perform medical tests". Use cases can also be named following a given sequence, such as CU1, CU2, CU1.2. Then, later, this sequence is indexed in a table or something similar.

Actor

Actors are users that interact with the system, and start the different actions represented in the use cases.



Actors can also represent the different system roles, or other systems that interact with our system.

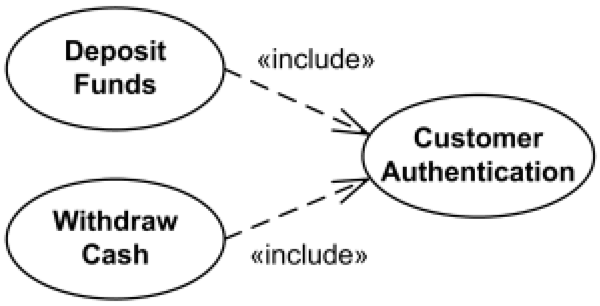
Relationships

- The relationships between actors and use cases are represented by a continuous line, and they mean the beginning of an action.
- The relationships between use cases can be of four different types, according to the following table.

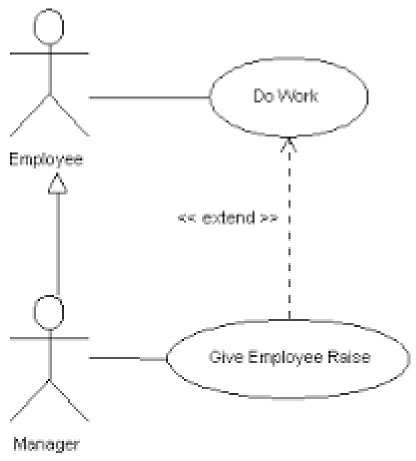
Relationship	**Representation**	**Meaning**
Communicates	———	It communicates an actor with a use case.
Include	— — <<include>> —→	This relationship is used when a use case includes the behavior of another use case. It is mainly used in common use cases, and the arrow points to that common use case.
Extend	←<<extend>> — — —	This relationship lets us add new functionalities to existing use cases in certain points called *extension points*. The arrow points to the base use case from which we want to extend the functionality.
Generalization	———▷	It is used when a use case or actor is a subtype of another use case or actor. Regarding use cases, is very similar to the *extend* relationship, so it is only recommended for actors (not for use cases).

Let's see some examples of these relationships to reinforce what we have learnt so far.

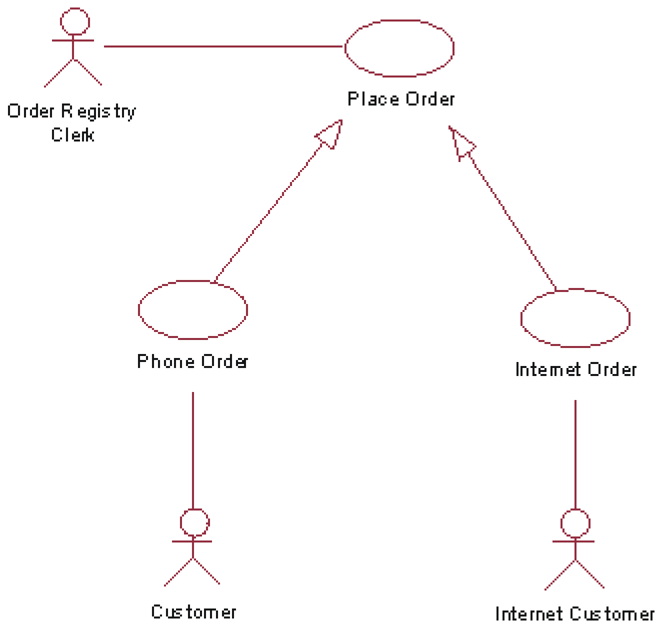
This is an example of an **include** relationship. It represents two basic operations in a bank, with two use cases: one called "Deposit funds" and the other one called "Withdraw money". In both cases, user needs to authenticate in the system, so they both include this use case functionality.



Let's see now an **extend** relationship. In this use case diagram, there are two actors: *Employee* and *Manager* They both do their respective work, but, besides, the manager must pay the employee. So we define the "Give Employee Raise" use case that extends the "Do Work" use case to let the manager pay the employee.



The example above also shows an example of a **generalization**, since the *Manager* actor is a subtype of *Employee*. Below we can see another example of generalization, applied to use cases (although we have seen before that this usage is NOT recommended): we can see a main super-case called *Place order* with two sub-cases that inherit from it: one for phone orders (*Phone order*) and the other one for Internet orders (*Internet order*). Both are different subtypes of the general use case.

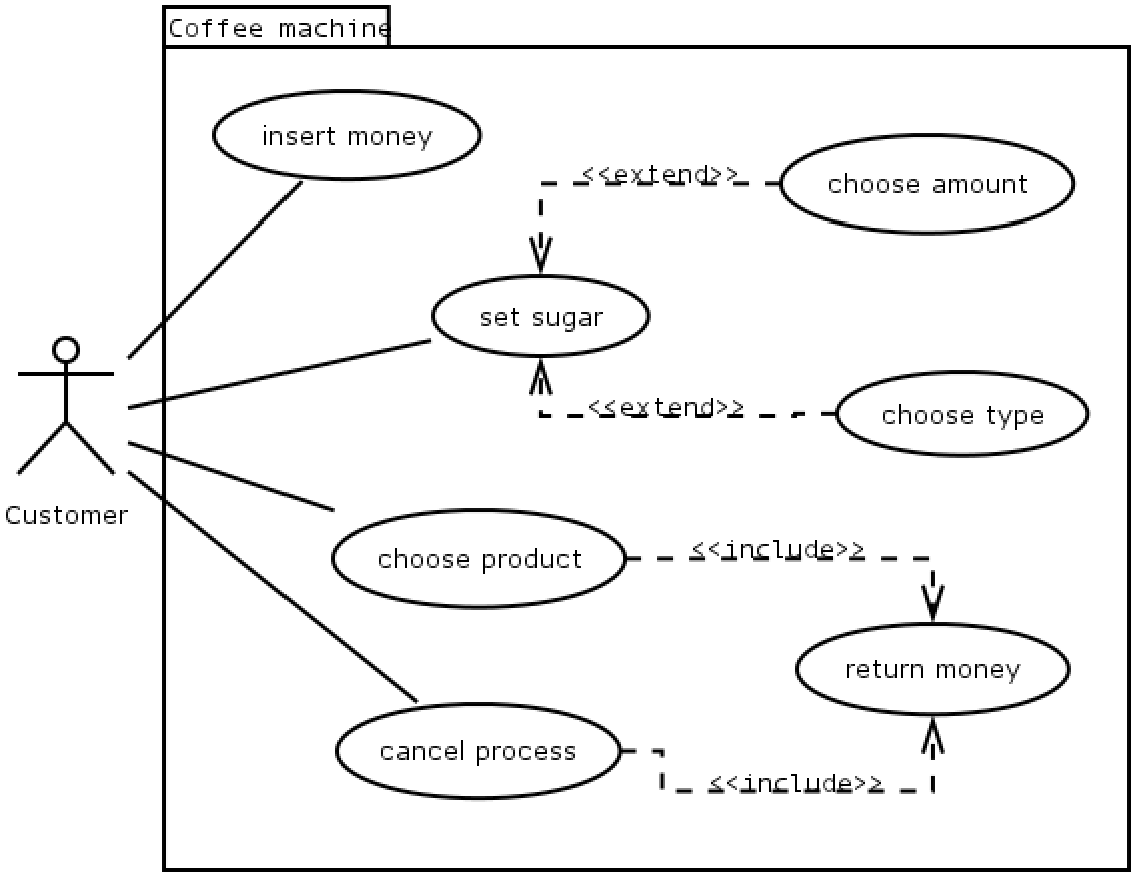


1.5.2.2. A classical example

Let's see how to get a use case diagram from the system requirements. We are in charge of developing a software to control a coffee machine. After talking with the customer, we get the following system requirements specification.

For a user to get coffee from our machine, he must start by introducing enough money. The following step will be to choose the amount of sugar. In our new machine model, users can both choose the amount and type of sugar (white, brown...), and it will be white sugar by default. Once the sugar is set, the user will choose the product (just coffee, coffee with milk, etc.). The machine will return the change once the process has finished and the coffee has been prepared and served. In case the user cancels the operation before choosing the coffee type, the machine will also return his money back.

As we can see, it is a quite detailed specification, but with the use case diagram we can have an overview from the user's point of view, this is, what the coffee machine is expected to do.



1.5.3. Specific software

If we look on the Internet for software to create use case diagrams, or UML diagrams in general, we can find a wide variety of applications. Most of them are proprietary, and some others are open source. We can even find some online tools to design our diagrams. Let's talk about some of them.

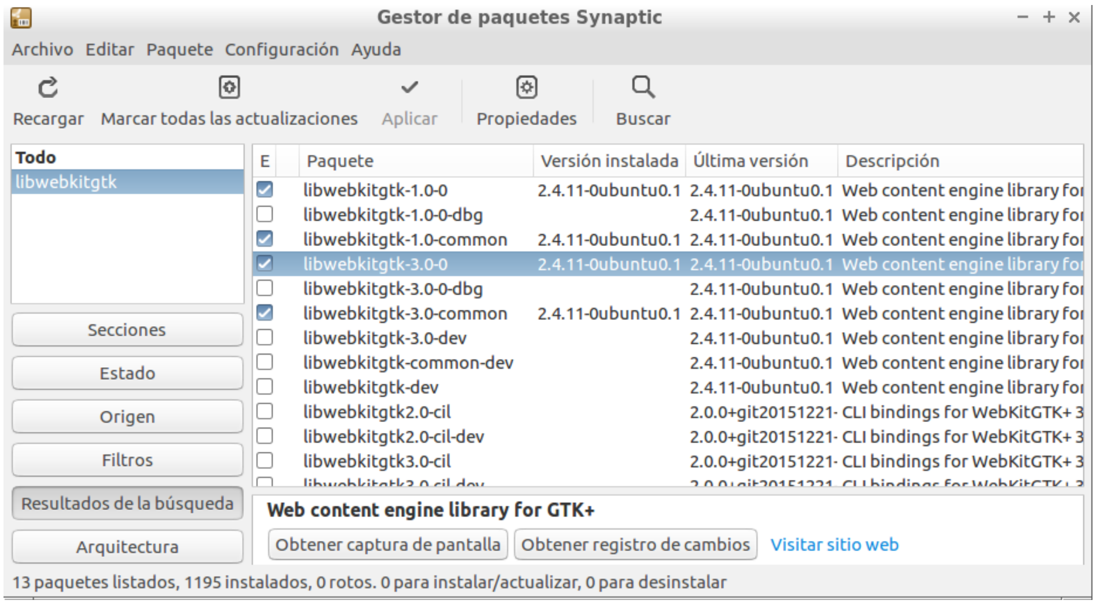
1.5.3.1. Modelio

Modelio is an open source tool to create a wide variety of diagrams, including UML diagrams. We can get it from its official web site for the three main platforms (Windows, Linux and Mac OS X), although there are some problems regarding Mac OS X, as we will see right now.

If we install it under **Windows**, we just need to run the *.exe* file, which is a step-by-step wizard that guides us through the installation process.

Regarding **Linux** (Ubuntu, for instance), we first need to install some dependencies from *Synaptic* package manager (you can find Synaptic in the *System tools* section). You must check if these packages (and all their dependencies) are installed:

- `libwebkitgtk-1.0.0`
- `libwebkitgtk-3.0.0`


Gestor de paquetes Synaptic
Archivo Editar Paquete Configuración Ayuda
Recargar
Marcar todas las actualizaciones
Aplicar
Propiedades
Buscar
Todo
libwebkitgtk
Secciones
Estado
Origen
Filtros
Resultados de la búsqueda
Arquitectura

E	Paquete	Versión instalada	Última versión	Descripción
✓	libwebkitgtk-1.0-0	2.4.11-0ubuntu0.1	2.4.11-0ubuntu0.1	Web content engine library fo
☐	libwebkitgtk-1.0-0-dbg		2.4.11-0ubuntu0.1	Web content engine library fo
✓	libwebkitgtk-1.0-common	2.4.11-0ubuntu0.1	2.4.11-0ubuntu0.1	Web content engine library fo
✓	libwebkitgtk-3.0-0	2.4.11-0ubuntu0.1	2.4.11-0ubuntu0.1	Web content engine library fo
☐	libwebkitgtk-3.0-0-dbg		2.4.11-0ubuntu0.1	Web content engine library fo
✓	libwebkitgtk-3.0-common	2.4.11-0ubuntu0.1	2.4.11-0ubuntu0.1	Web content engine library fo
☐	libwebkitgtk-3.0-dev		2.4.11-0ubuntu0.1	Web content engine library fo
☐	libwebkitgtk-common-dev		2.4.11-0ubuntu0.1	Web content engine library fo
☐	libwebkitgtk-dev		2.4.11-0ubuntu0.1	Web content engine library fo
☐	libwebkitgtk2.0-cil		2.0.0+git20151221-	CLI bindings for WebKitGTK+ 3
☐	libwebkitgtk2.0-cil-dev		2.0.0+git20151221-	CLI bindings for WebKitGTK+ 3
☐	libwebkitgtk3.0-cil		2.0.0+git20151221-	CLI bindings for WebKitGTK+ 3

Web content engine library for GTK+

Obtener captura de pantalla | Obtener registro de cambios | Visitar sitio web

13 paquetes listados, 1195 instalados, 0 rotos. 0 para instalar/actualizar, 0 para desinstalar

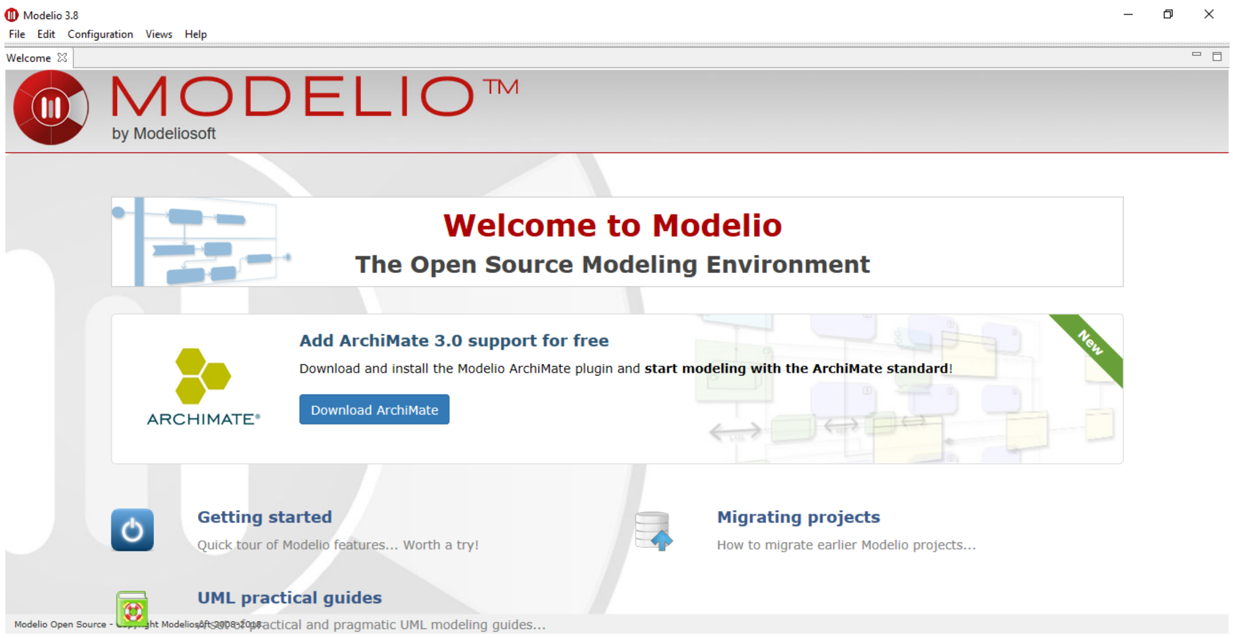
Then, we download the *.deb* file from the official web site, and type this command from the terminal (from the folder where the file has been downloaded):

```
sudo dpkg -i <deb_file_name>
```

where `<deb_file_name>` is the full file name that you have downloaded (for instance, `modelio-open-source3.8_3.8.1_amd64.deb`).

Finally, regarding **Mac OS X** it requires Java JDK 8 to run. Since we are going to use a newer version, and Java 8 is becoming quite obsolete, we are not going to explain how to fix this bug, since it is a little bit tricky. We will use a different tool for Mac OS X instead.

Once the installation is done, we can start Modelio. This is the welcome screen:



Welcome to Modelio
The Open Source Modeling Environment

Add ArchiMate 3.0 support for free
Download and install the Modelio ArchiMate plugin and start modeling with the ArchiMate standard!

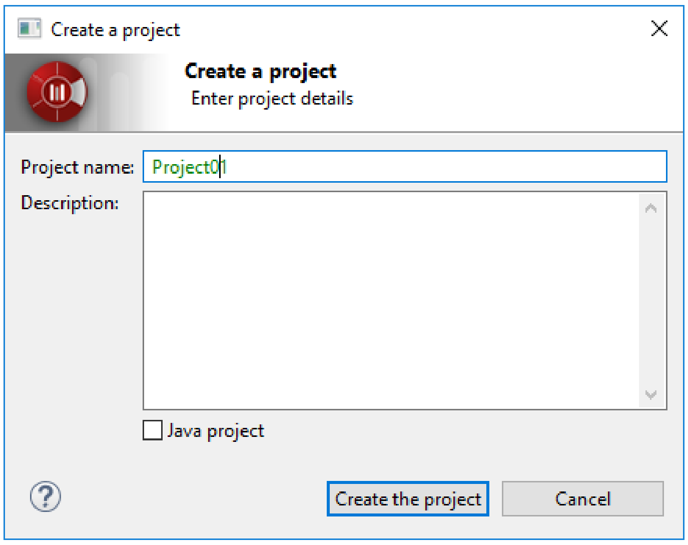
Getting started
Quick tour of Modelio features... Worth a try!

Migrating projects
How to migrate earlier Modelio projects...

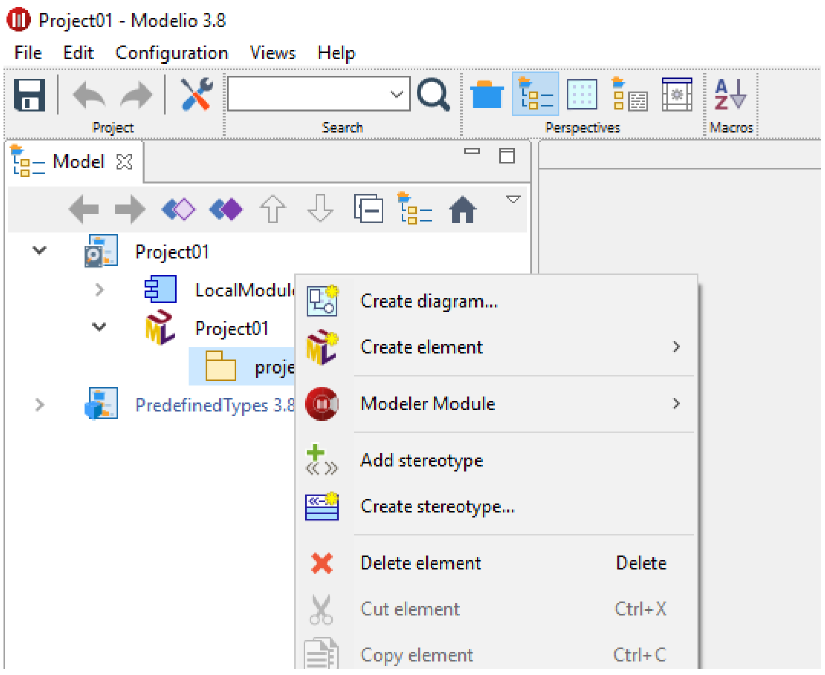
UML practical guides
Practical and pragmatic UML modeling guides...

Modelio Open Source - Copyright Modeliosoft 2009-2018

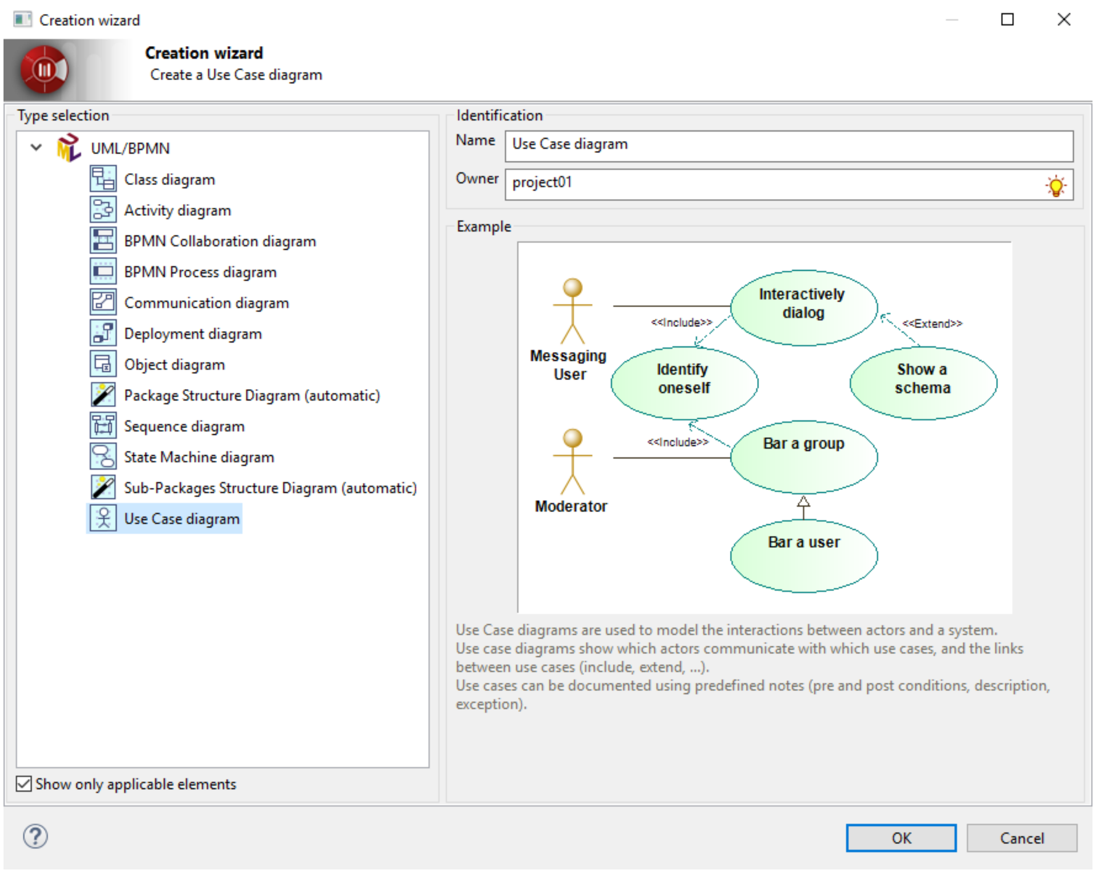
We need to create a new project (from *File* > *Create a project* menu) if we want to start using Modelio. Then, we choose a name for the project



After creating the project, we can see it open at the left panel. Now, we need to unfold it and right click on its inner package to choose *Create diagram*.



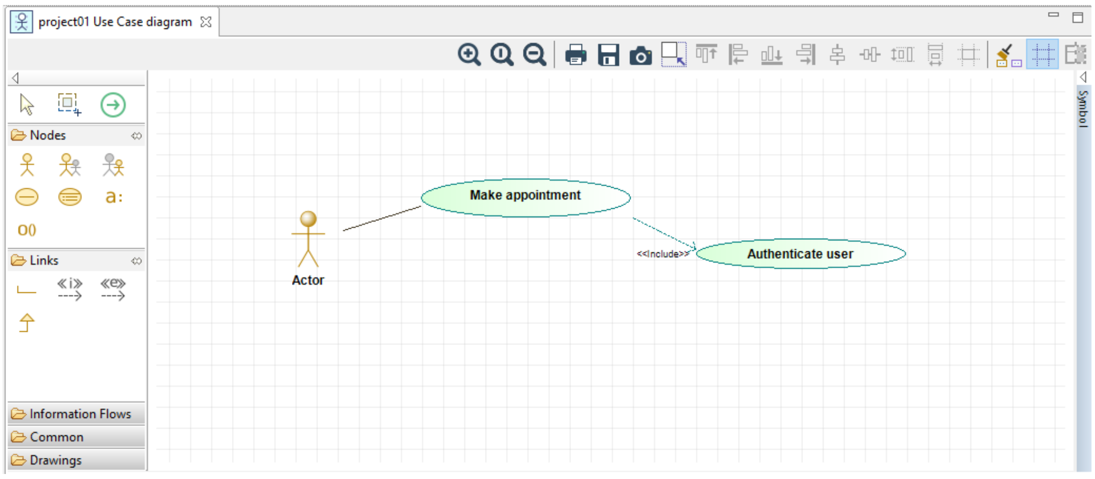
Then, a new dialog with a list of diagrams will appear:



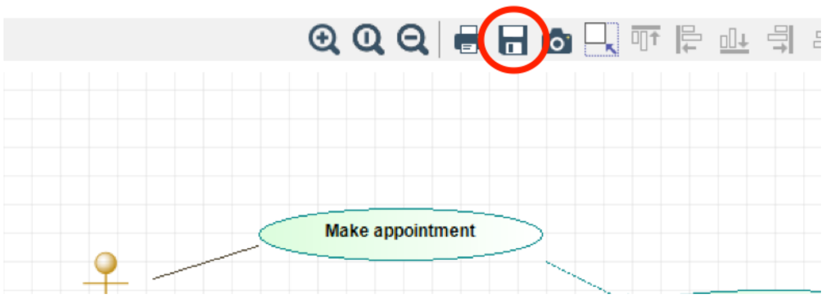
Name: Use Case diagram
Owner: project01

Use Case diagrams are used to model the interactions between actors and a system.
Use case diagrams show which actors communicate with which use cases, and the links between use cases (include, extend, ...).
Use cases can be documented using predefined notes (pre and post conditions, description, exception).

We can choose *Use case diagram* (you should also give it a name in this dialog) and a new, blank diagram will be created. We can select (left click) the elements from the left section (actors, use cases, connections) and place them (left click) in the main drawing area. If we right click on any element, we can change some attributes, such as its name with the *Rename* option.



Once the diagram is created, we can just save it, or export it to an image file. To do this, we need to click on the *Save diagram in a file* button, in the toolbar above the drawing area:



1.5.3.2. Visual Paradigm (online)

Visual Paradigm is a commercial tool with a free community version, and also an online version that can be used with no installation required. We can either download the free community edition from the download page, or try the online version. We are going to focus on this last option, for which we need to sign up before using the tool.

Once we sign up and log in, we can choose between creating a diagram or a form (we choose a diagram, of course):

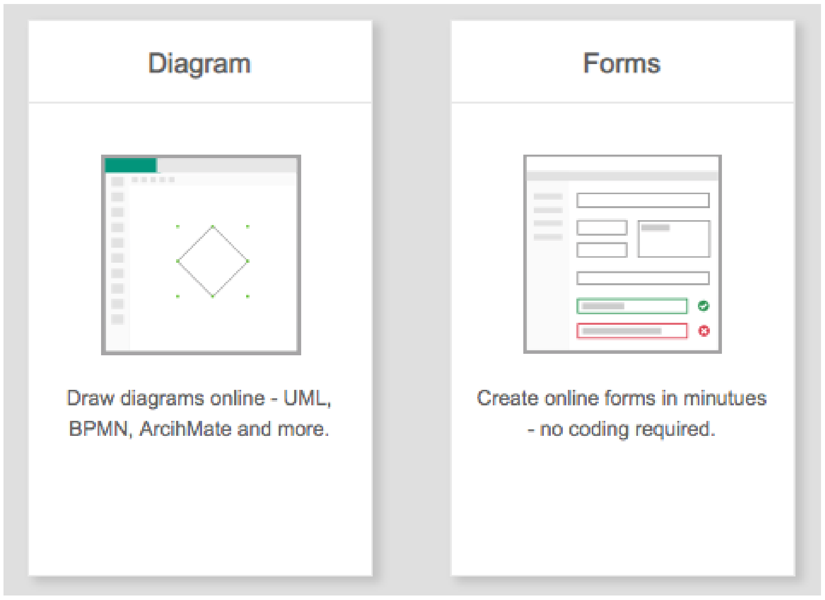
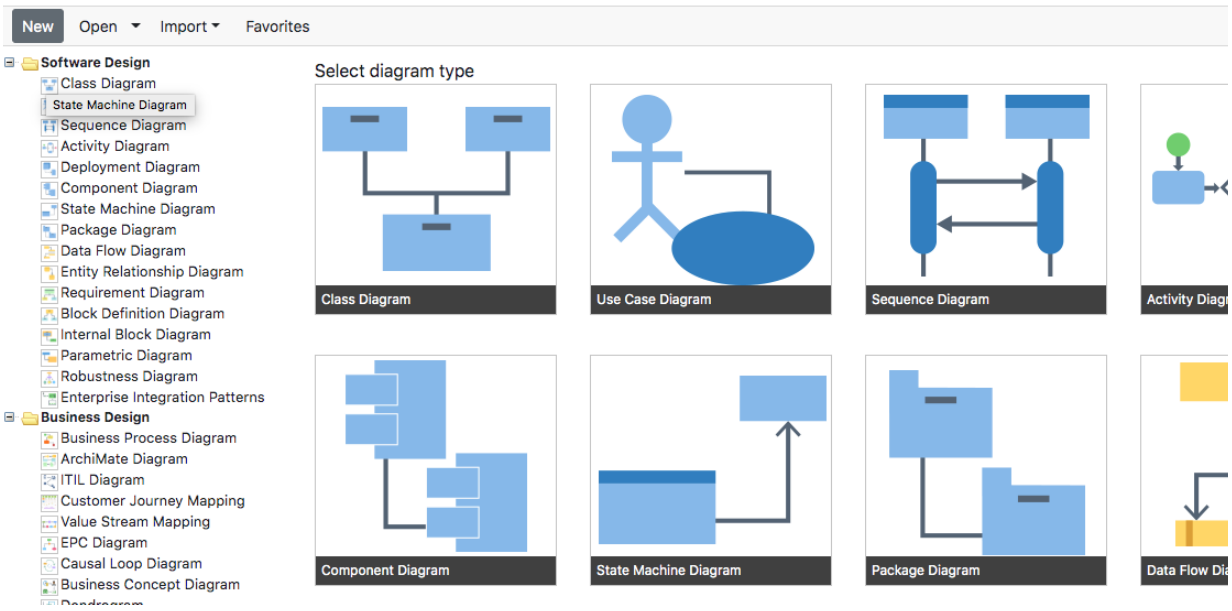
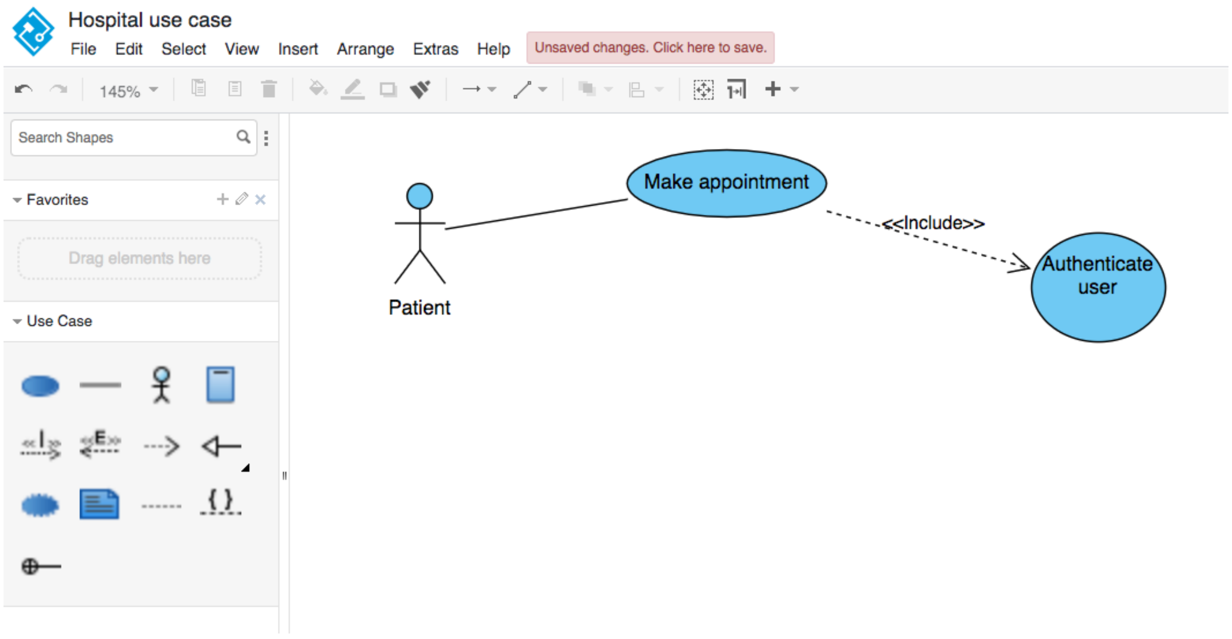


Diagram
Draw diagrams online - UML, BPMN, ArcihMate and more.
Forms
Create online forms in minutues - no coding required.

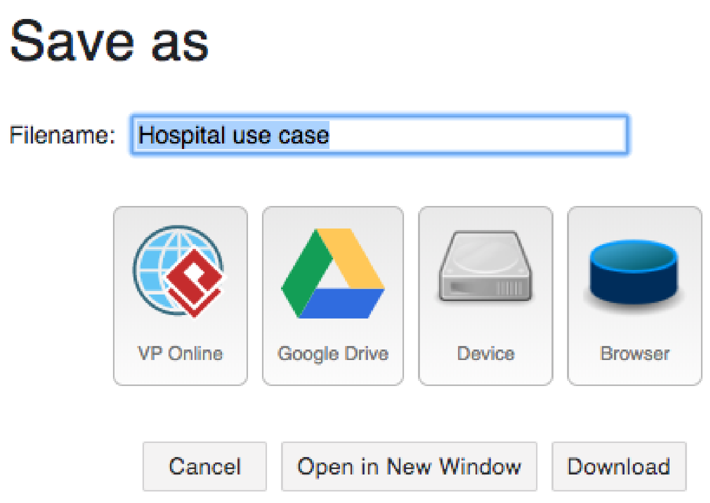
Then, we must select which type of diagram are we going to create. In this case, we are going to choose a *Use case diagram*. We can do this either from the left list of diagrams, or from the large icons in the middle of the page.



Next, we can start drawing the diagram, by clicking on the desired component on the left panel and dragging it on the main area. If we click on any element, we can rename it by just typing the new name.



Finally, we can either export the diagram as an image file from *File* > *Export* menu, or we can save the diagram for later modifications, from the *File* > *Save* menu. In this case, we can save it on many targets, such as a Google Drive account, or a physical device (such as the hard drive).



1.5.3.3. Other tools

There are some other tools that we can use to create UML diagrams, such as:

- **StarUML** a commercial software with a free evaluation version with no time limit. You can get it here.
- **Dia** another tool for general purpose diagrams, including UML. You can get it here.
- **Draw.io** another online editor for many types of diagrams. You can try it here.

Proposed exercises

If you are going to do the following exercises using Modelio, we recommend you to place all of them in a single project called *Block1*.

1.5.3.1. Design the use case diagram for the following system requirements specification.

A customer asks us to develop a software for a snack vending machine. This machine has a two digit code for each product, and it has five shelves with eight products on each shelf. In the first shelf, products are numbered from 11 to 18, in the second shelf numbers go from 21 to 28 and so on. We must take into account that the machine will not accept the money until the user chooses the product first. The user can cancel the operation at any time before pressing the confirmation button, then his money will be returned. After pressing this button, the machine will move a spiral for the chosen product until it falls to the bottom of the machine, so that it can be taken by the user. Then, the user will be able to take his change back, if necessary.

1.5.3.2. Design a use case diagram for this system requirements specification.

We want to implement an App to solve Sudokus. In order to start a game, the user must choose the "Start Game" option, and then he will choose the difficulty level. Once it is chosen, the game starts. Then, the user will iteratively choose an empty cell, and place a number from 1 to 9, until it completes the Sudoku. During the game, the user can press the "Check" button in order to check if he has solved the Sudoku, or if he has made any mistake. The app will show the message "Everything is OK, you have X cells left", or "Error in highlighted cells". In this last case, the cells with wrong numbers will be highlighted. Once the Sudoku is

> *solved, the system will show the message "Well done!", and the user will see his stats and total time employed.*
>
> **1.5.3.3.** Design a use case diagram adapted to the following system requirements specification.
>
> *We want to design an invoice checking module. This module can be accessed either by customers or by business agents, by logging in with their credentials (user and password). When a customer logs in the module, he will be able to search his own invoices, either by date range or by amount. In case a business agent logs in the system, he will be able to search for his customer's invoices, or by customer and date range, or by customer and amount. Once the search is completed, the module will show in the screen the results.*
>
> **1.5.3.4.** Implement the use case diagram from exercise 1.4.3.2 about the cultural organization.

1.5.4. To learn more

If you want to learn more about UML and use case diagrams, you can have a look at the following links:

- UML
- UML 2.5.1.
- Creating UML Use Case Diagrams