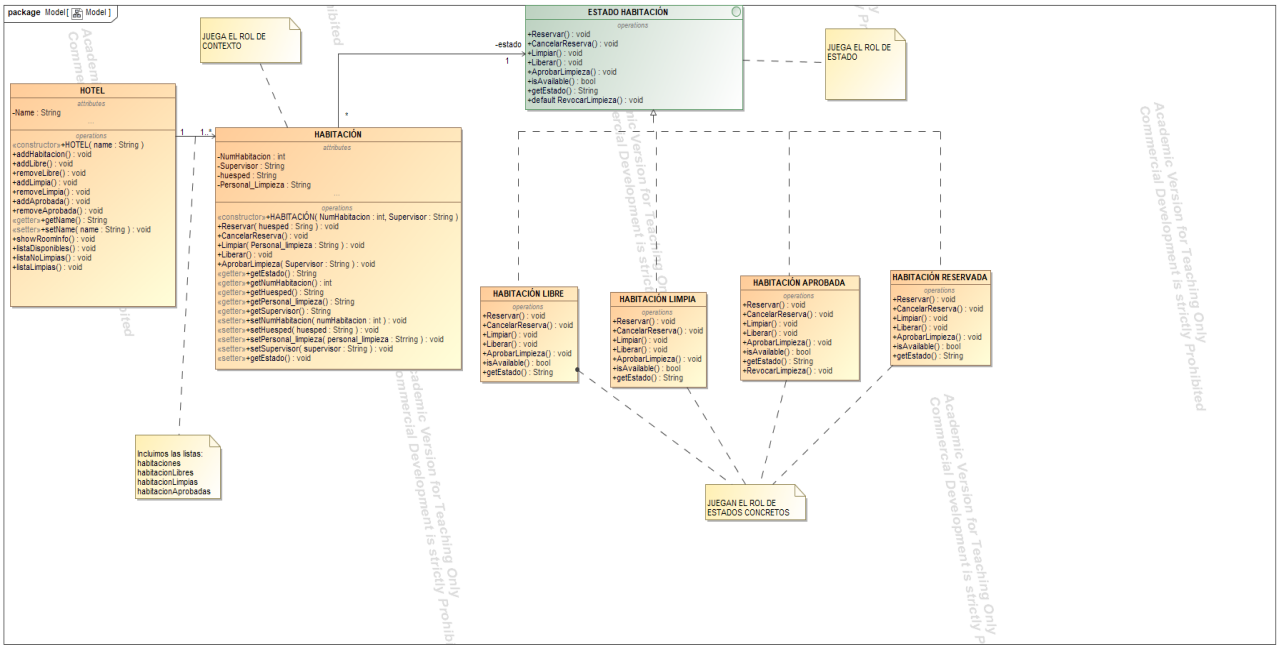
1-Gestión de hotel

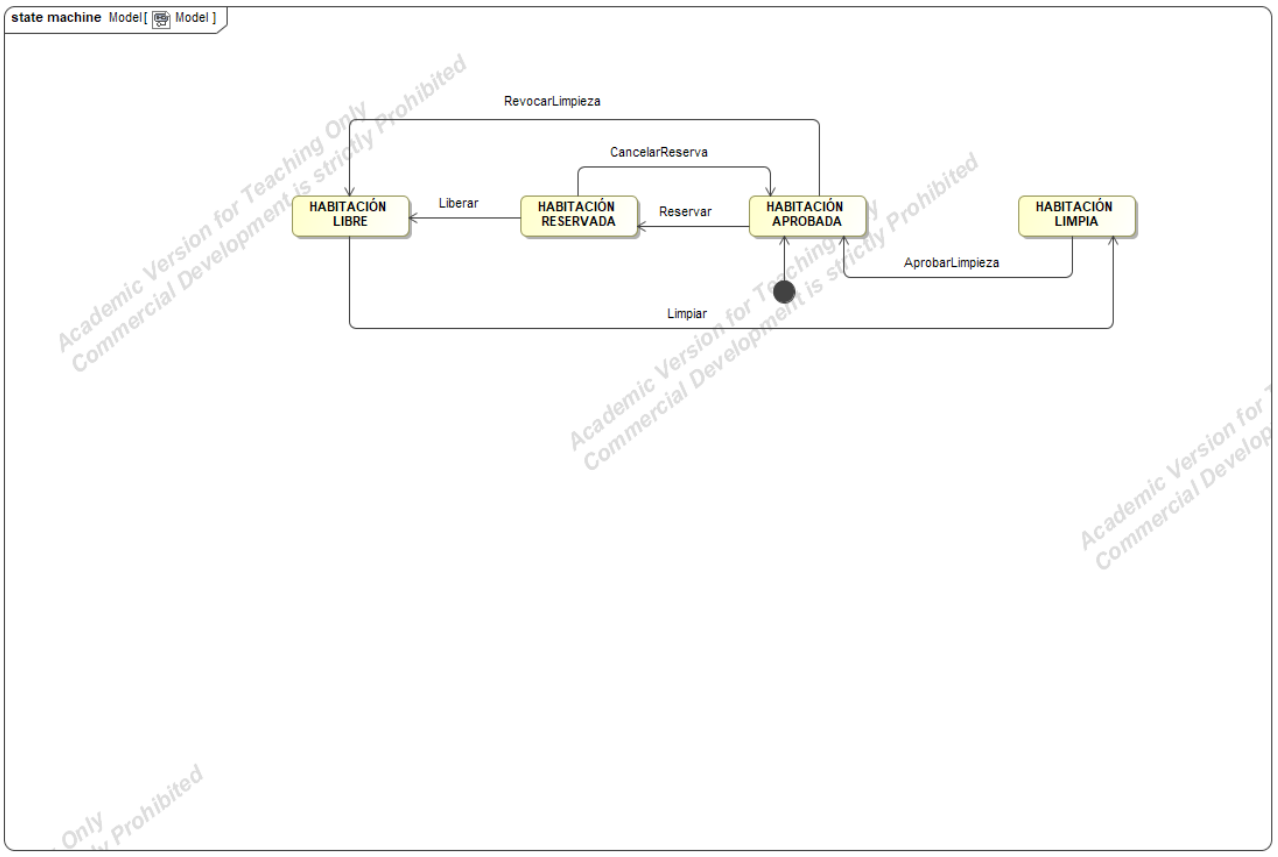
En el primer ejercicio hemos decidido utilizar el patrón de estado ya que vamos a manejar diferentes estados por los que va a pasar una habitación de un hotel,con ello permitimos a una habitación modificar su conducta al cambiar su estado interno,además este patrón nos será útil para evitar el uso de sentencias condicionales que complicaría saber el funcionamiento de un estado concreto,además de ser poco adaptable a los posibles cambios de futuro,cosa que nos interesa que no sea así porque en el futuro podrían aparecer nuevos estados de las habitaciones que nos obligarían a cambiar la clase Habitación ,incumpliendo principios como el Abierto-Cerrado, y que la clase Habitación tendría demasiadas responsabilidades incumpliendo el principio de Responsabilidad Única.Gracias a este patrón podremos añadir nuevos estados simplemente añadiendo nuevas clases pensando en el futuro.Emplearemos una clase Hotel para gestionar las habitaciones y obtener el listado de las diferentes habitaciones en función de su estado.Usaremos una clase Habitación que jugará el rol de Contexto,mantiene una instancia de un estado concreto y delega en ella el funcionamiento de ese estado concreto;Esta clase no conocerá los diferentes estados de la habitación gracias a la interfaz Estado que jugará el rol de Estado,la cual encapsula el comportamiento de los estados del contexto.Por último emplearemos las clases Habitación libre,limpia,aprobada y reservada que jugarán el rol de Estados Concretos los cuales implementan el comportamiento asociado con un estado del contexto.Además este patrón nos será útil gracias a que los estados se conocen entre sí ,esto es importante para establecer transiciones entre ellos y que dado un comportamiento , los estados podrán variar de uno a otro.
Utilizamos además el patrón Instancia Única ya que definiremos los estados como singletons para poderlos compartir entre los distintos tipos de habitaciones.

En cuanto a los Principios empleamos el principio abierto-cerrado en la clase habitación ya que tiene una implementación por defecto ,y luego cada tipo de habitación reside en una clase propia con sus características propias.Empleamos el principio de Responsabilidad Única en todas las clases, ya que todas las clases solo tiene una responsabilidad donde un cambio solo variaría en dicha clase.
Empleamos el principio de la inversión de la dependencia en la interfaz Estado ,ya que se depende de la interfaz y no de clases o funciones concretas.Con esto además conseguimos cumplir el principio de "Bajo Acoplamiento" ya que hacemos que nuestra clase no trabaja sobre objetos concretos ,consiguiendo que sea menos dependiente de una implementación concreta.
Además empleamos un método por defecto en la interfaz Estado que es Revocar Limpieza ya que no se usará en ninguna clase excepto en la clase Habitación Aprobada que si que le dará su propia implementación, en el resto se dará una implementación por defecto.Además todas las clases cumplen con el principio de Mínimo Conocimiento, ya que dentro de los métodos de los objetos solo mandamos mensajes al propio objeto , un parámetro del método, un atributo propio del objeto o un elemento de una colección que es atributo del objeto.

miguel.marcos.trillo@udc.es
gael.garciaa@udc.es

Emplearemos para el diagrama dinámico el diagrama de estado porque con ello podremos mostrar los distintos estados por los que pasa una habitación, y como puede variar su estado en función de los eventos que recibe.Su estado inicial será la habitación aprobada,esto quiere decir una habitación disponible para ser reservada y a partir de ella la habitación podrá ir pasando por diferentes estados en función de los métodos que se apliquen,nunca habrá un estado final ,ya que los distintos estados por los que pasa una habitación se irán cambiando con el paso del tiempo y una habitación nunca dejará de modificar su estado,por lo que nunca termina.





miguel.marcos.trillo@udc.es
gael.garciaa@udc.es

2-Incursiones Navales

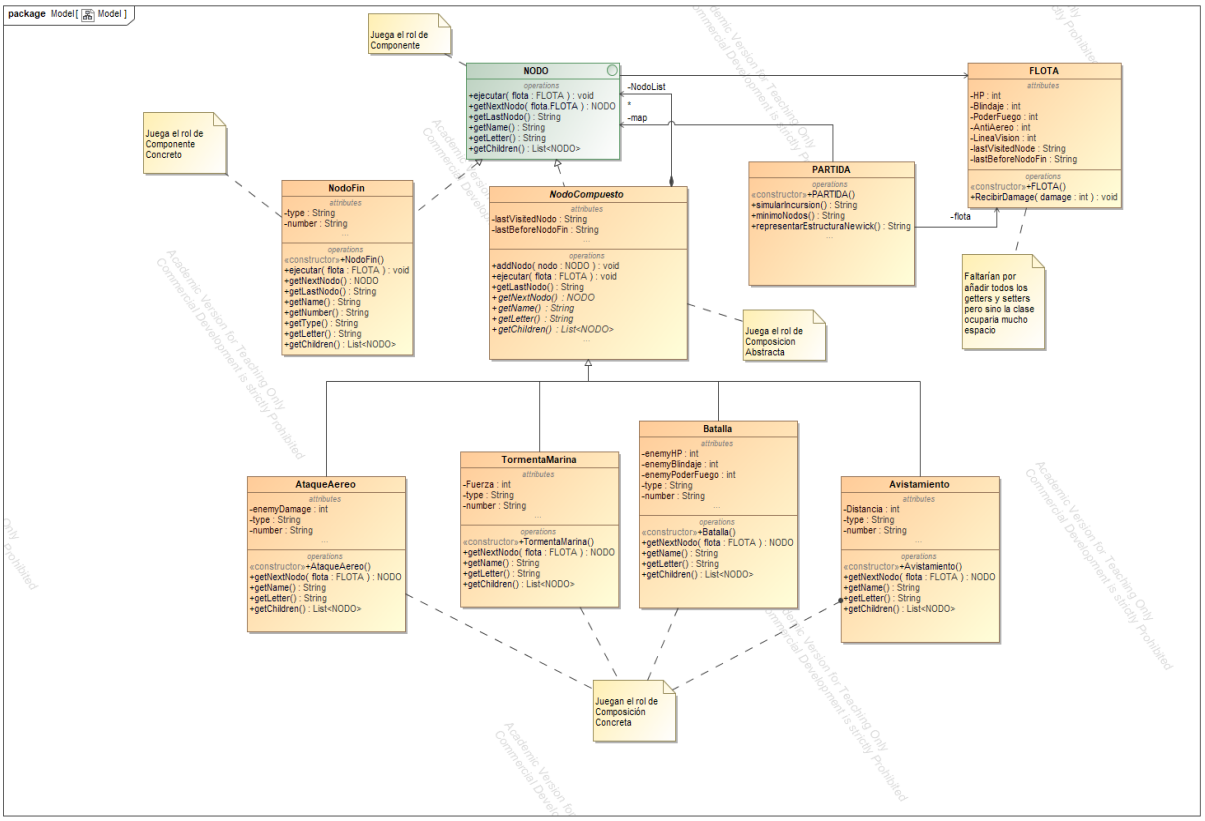
En el segundo ejercicio utilizaremos el patrón Composición, específicamente la Composición compleja. Para estructurar los nodos del mapa en nodos sin hijos y con hijos, usaremos una interfaz Nodo de la cual tanto un nodo fin (sin hijos) como un nodo compuesto (nodo con uno o dos hijos) implementarán los métodos de Nodo para ejecutar todas las operaciones que contengan sus hijos. Por lo tanto, usaremos la composición compleja, empleando una clase abstracta que será Nodo Compuesto, de la cual heredarán los distintos tipos de nodos con hijos, implementando cada uno su distinta operación para pasar al siguiente nodo. Este patrón permite la construcción de una estructura jerárquica flexible y la definición de comportamientos específicos en cada nivel de la jerarquía.Además nos será útil ya que nos permite usar los objetos básicos y los compuestos de manera uniforme y nos facilita la posibilidad de añadir nuevos nodos pensando de cara al futuro por si aparecen nuevos nodos,siendo esto sencillo ya que se simplifica a añadir una nueva clase sin modificar el resto del código.

En nuestro ejercicio la interfaz Nodo jugará el rol de Componente donde sus métodos serán implementados por los distintos tipos de nodos.Nodo Fin juega el rol de Componente Concreto el cual define el comportamiento de los objetos elementales de la composición, es decir, los que no tienen hijos.Nodo Compuesto juega el rol de Composición Abstracta , será una clase abstracta común a todos los nodos que contengan uno o dos hijos,el cual contendrá la lista de nodos y operaciones como añadir nodos. Las clases Ataque Aéreo ,Tormenta Marina,Avistamiento y Batalla juegan el rol de Composición Concreta donde cada uno implementará de forma distinta la operación de obtener el siguiente nodo en función de los requisitos de cada uno.

En cuanto a los principios,emplearemos el principio Abierto-cerrado en la interfaz Nodo y en la clase abstracta Nodo Compuesto y cada tipo de Nodo o de Nodo Compuesto reside en una clase propia con sus características propias que implementarán en el caso de los nodos los métodos del interfaz ,y sobreescribir los métodos de la clase abstracta en el caso de los tipos de nodos compuestos. También empleamos en principio de Inversión de la dependencia en la interfaz Nodo y en la clase abstracta Nodo Compuesto ya que dependeremos de interfaces o clases abstractas en vez de clases y funciones concretas, esto además lo podemos ver con el diagrama de clases donde vemos que los módulos de alto nivel dependen de los módulos de bajo nivel..Además todas las clases cumplen con el principio de Mínimo Conocimiento, ya que dentro de los métodos de los objetos solo mandamos mensajes al propio objeto , un parámetro del método, un atributo propio del objeto o un elemento de una colección que es atributo del objeto.

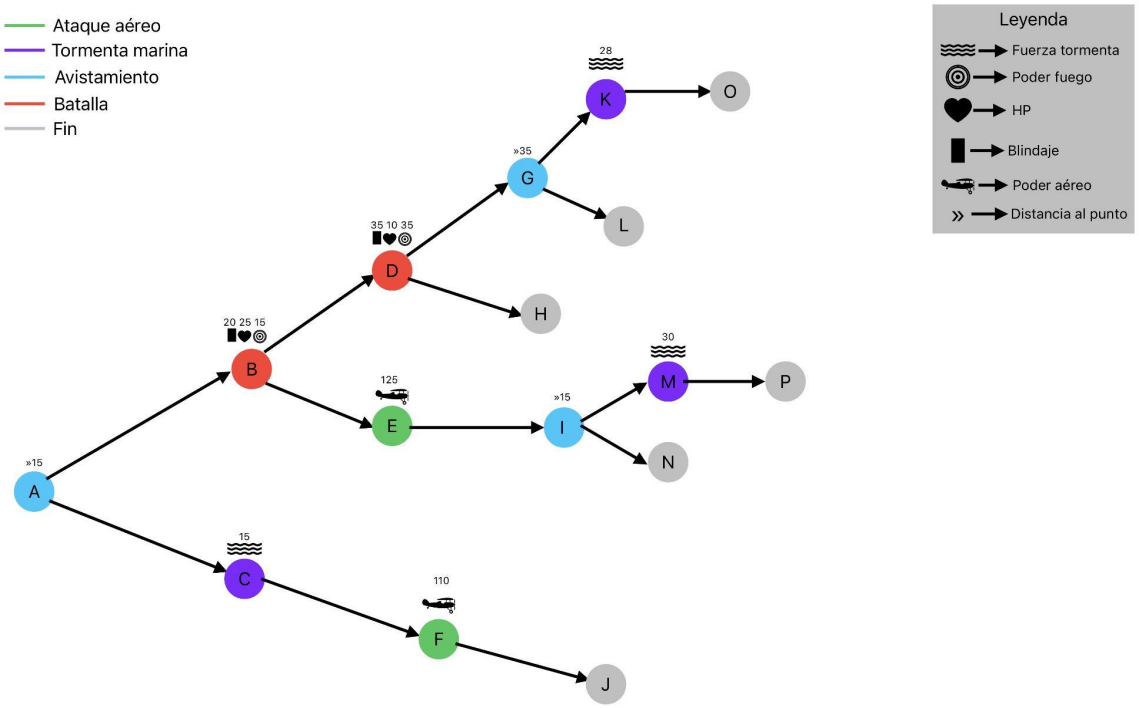
miguel.marcos.trillo@udc.es
gael.garciaa@udc.es

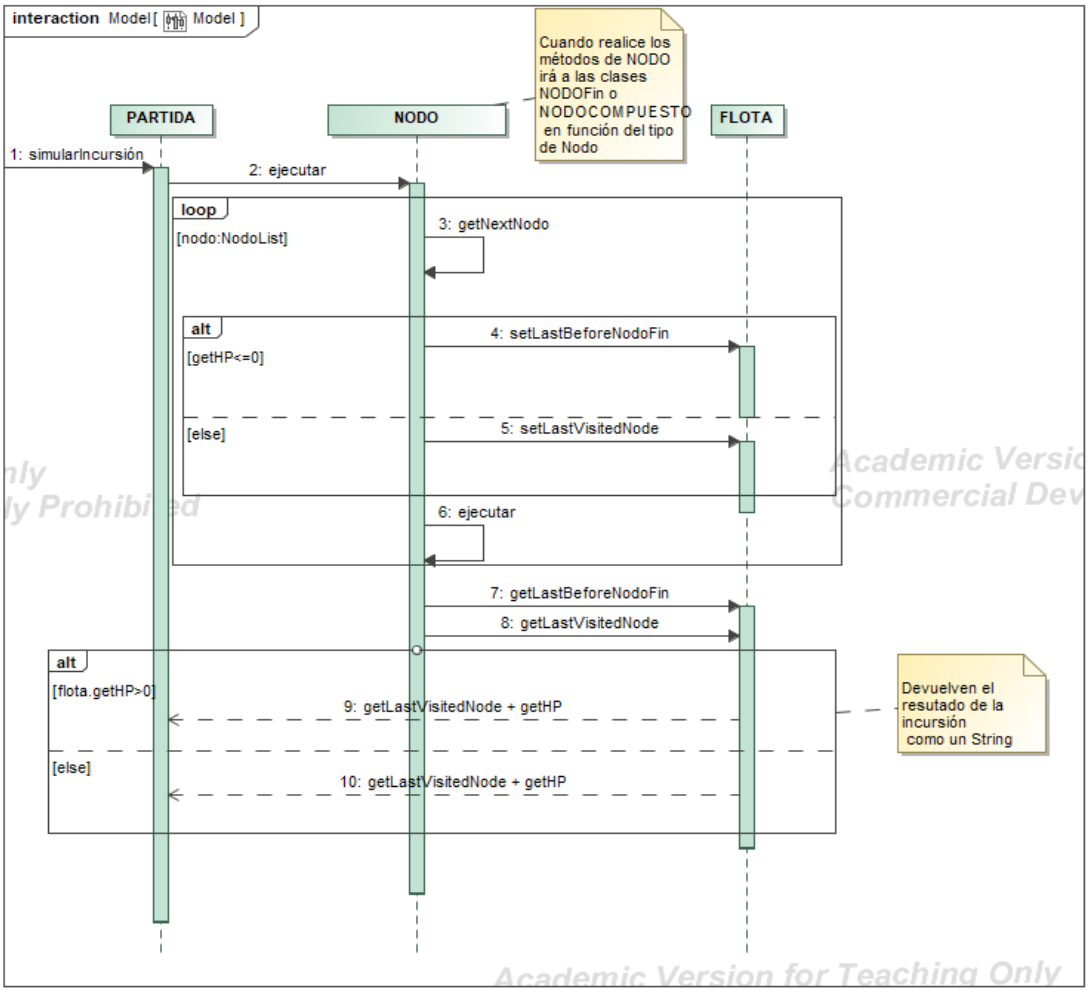
Para el diagrama dinámico,emplearemos el diagrama de secuencia, a pesar de que podría ser interesante el uso del diagrama de comunicación ya que tenemos muchas clases pero decidimos usar el de secuencia para manejar bucles de forma más sencilla.En él representaremos la clase partida para realizar el método de Simular Incursión ,que es lo fundamental del ejercicio y con ello mostramos los distintos recorridos que se realizan mediante la composición de los nodos pasando por sus hijos y realizando las operaciones concretas en función del tipo de nodo hasta obtener el resultado de la incursión.Además representaremos de forma general los distintos tipos de nodos mediante la interfaz Nodo,donde según el tipo de operación ,por ejemplo getNextNodo, acceda a según qué nodo, a sus operaciones concretas y vaya obteniendo su hijo(s) o no en función de si llega a un Nodo Fin o si continúa con un Nodo Compuesto.



La representación que se muestra a continuación corresponde al segundo test de la práctica 2, es decir, a simularIncursion2, creado por nosotros. La última foto corresponde al diagrama dinámico de secuencia.

miguel.marcos.trillo@udc.es
gael.garciaa@udc.es





miguel.marcos.trillo@udc.es
gael.garciaa@udc.es