BIKERENT

MEMORIA

Armero Guerra, María
Jiménez Soria, Alberto
Martin Martínez, Javier
Muñoz de la Peña Herranz, Laura
de Pablos Blanco, Guillermo
Sánchez Castellanos, Isaac
Sánchez Jiménez, Jose Luis
Sánchez Sánchez, Adrian
Suarez Solorzano, Sergio
Valenciano López, Javier

2012

INGENIERIA DEL SOFTWARE 2

INDICE

1. INTRODUCCION 3

2. MODELO DE DATOS 3

2.1. ANALISIS 3

2.1.1. El modelo del dominio 3

2.1.2. Modelo de negocio 4

2.2. DISEÑO 5

2.2.1. Casos de uso 5

2.2.2. Actividades 6

2.2.3. Transición estados 7

2.2.4. Secuencia 8

2.2.5. Comunicación 9

2.2.6. Componentes 10

2.2.7. Despliegue 10

3. DESARROLLO APLICACION 11

3.1. MVC (Modelo Vista Controlador) 11

3.2. TRANSFER 11

3.3. DAO (Data Access Object) 12

3.4. SINGLETON 12

1. INTRODUCCION

Este documento es una Memoria explicativa del modelo de datos de SGAB y el desarrollo de la aplicación. Pretende completar la parte teórica de la Especificación de Requisitos y el Plan de Proyecto, añadiendo esta explicación de la parte práctica del proyecto.

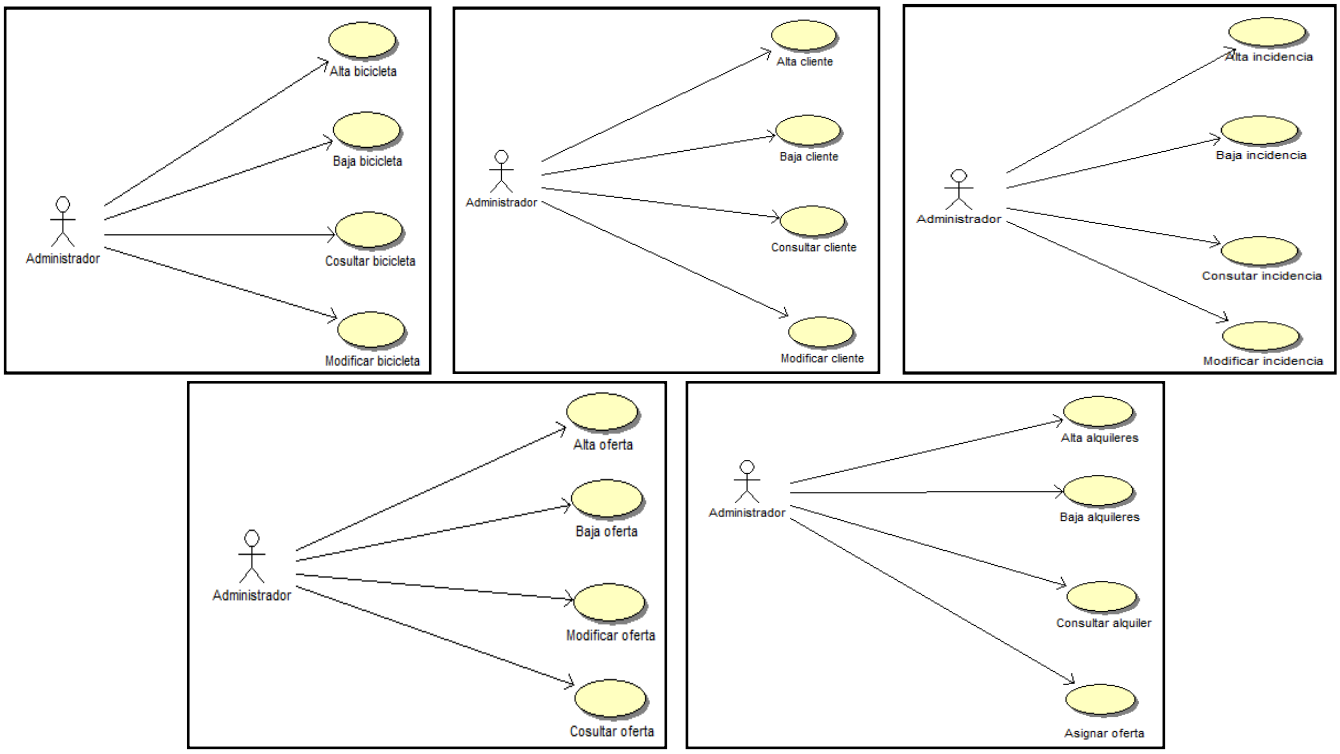
2. MODELO DE DATOS

El modelo de datos de la aplicación tiene como objetivo detallar extensamente como es la estructura de los datos. Este modelado está hecho en la aplicación "BOUML UML"; a continuación se explicará cada paquete siguiendo la estructura y captando algunas imágenes de los diagramas incluidos en el proyecto UML denominado "BIKERENT".

2.1. ANALISIS

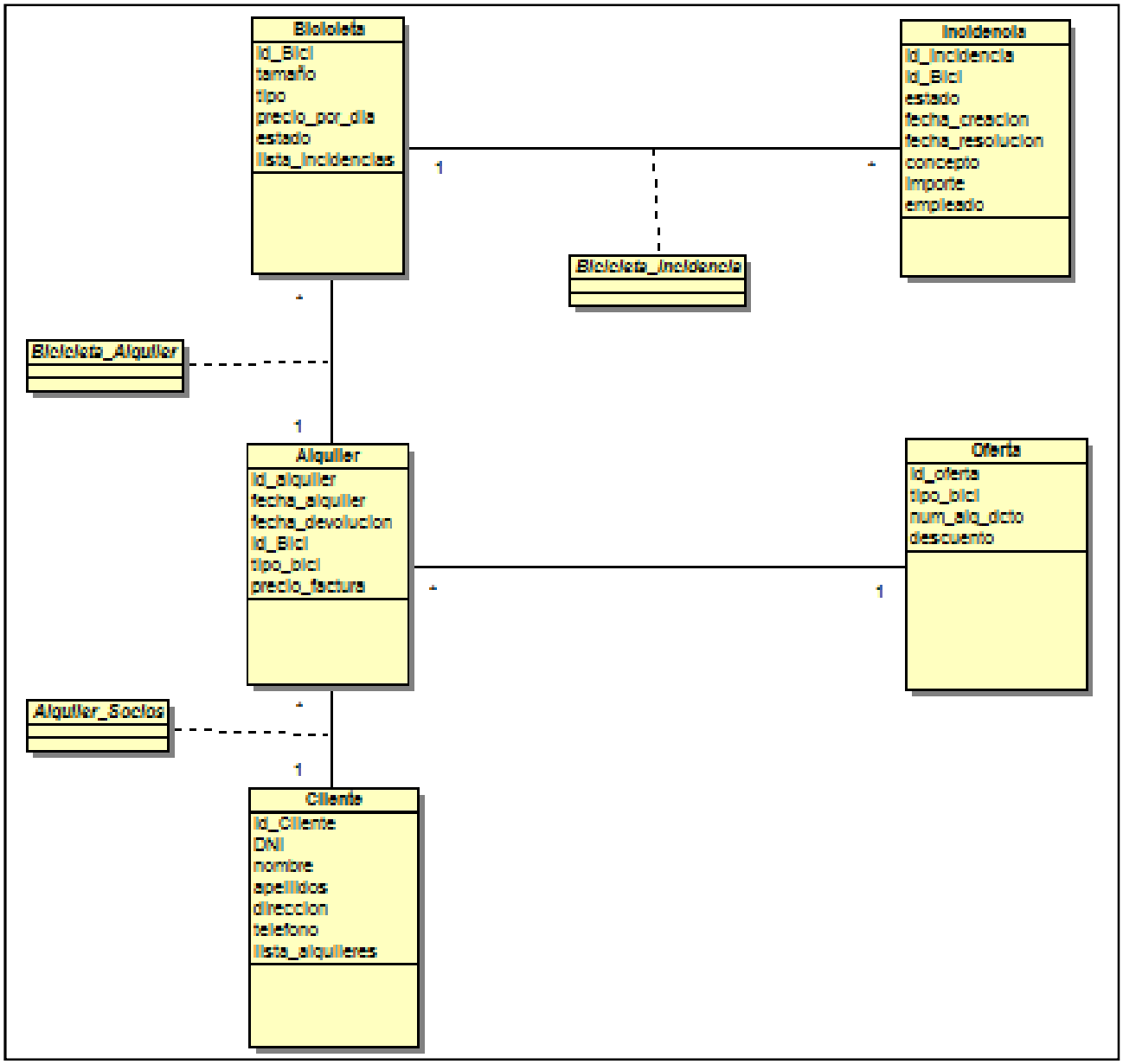
2.1.1. El modelo del dominio

El modelo del dominio muestra la relación entre los diferentes módulos de gestión: Bicicletas, Incidencias, Clientes, Alquileres y Ofertas.



2.1.2. Modelo de negocio

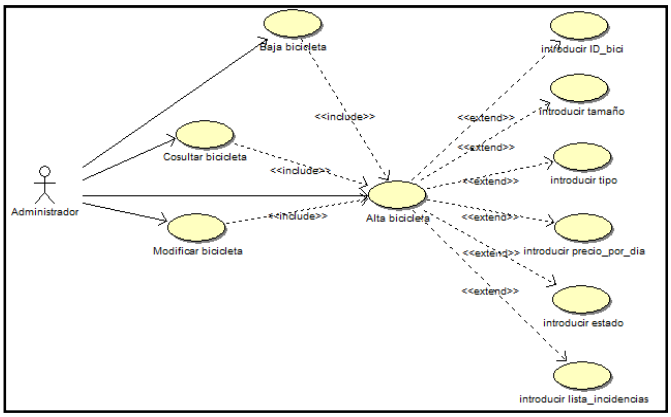
El modelo de negocio contiene los casos de uso generalizados de cada modulo de gestión: Bicicletas, Incidencias, Clientes, Alquileres y Ofertas.

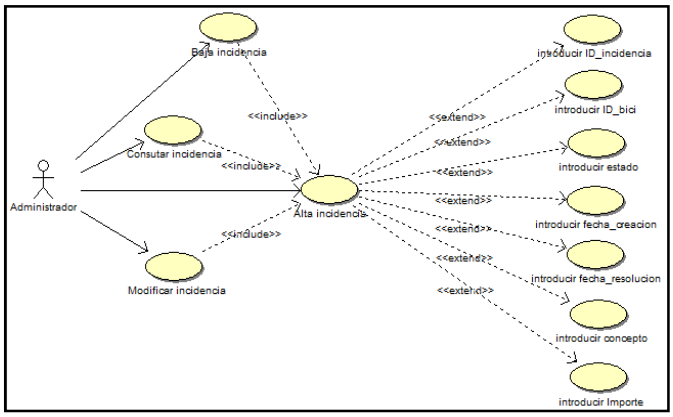


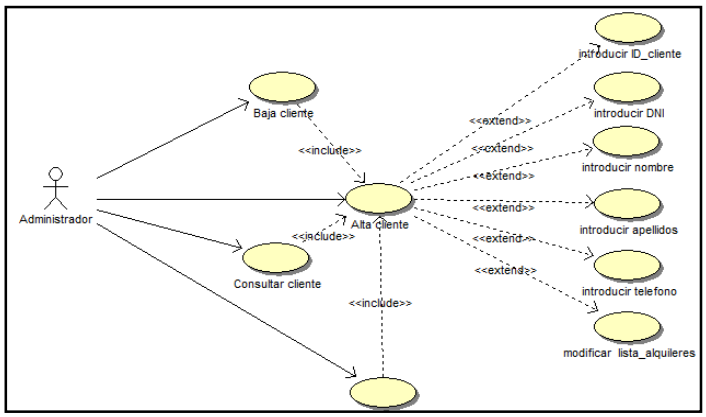
2.2. DISEÑO

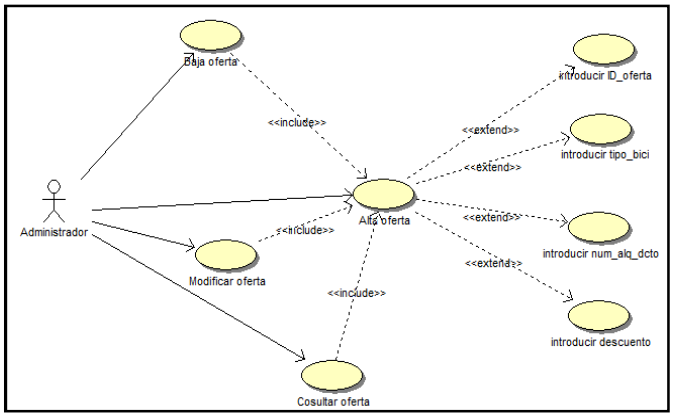
2.2.1. Casos de uso

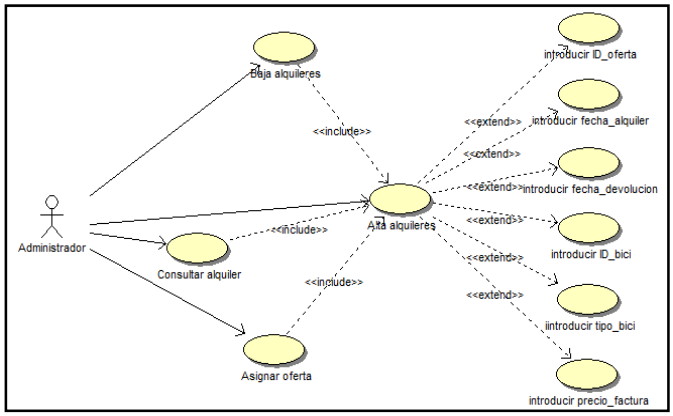
En este paquete se realiza una descripción detallada de los casos de uso de un módulo de gestión con las inclusiones y extensiones de cada uno dentro del mismo. En el proyecto UML están realizados todos pero en la memoria incluiremos Bicicleta como ejemplo orientativo de aquí en adelante con los siguientes diagramas.





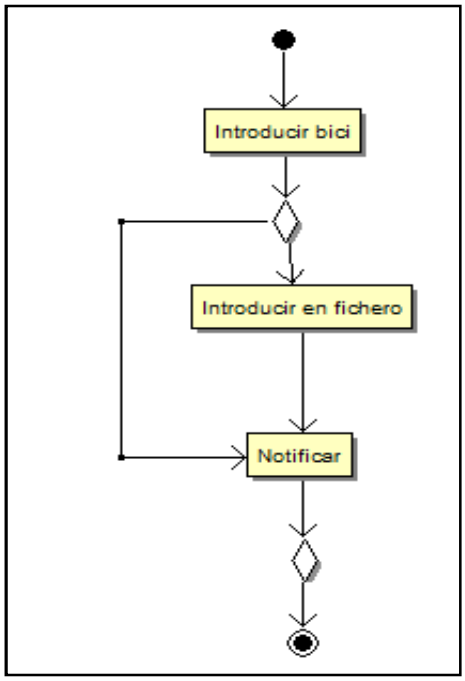


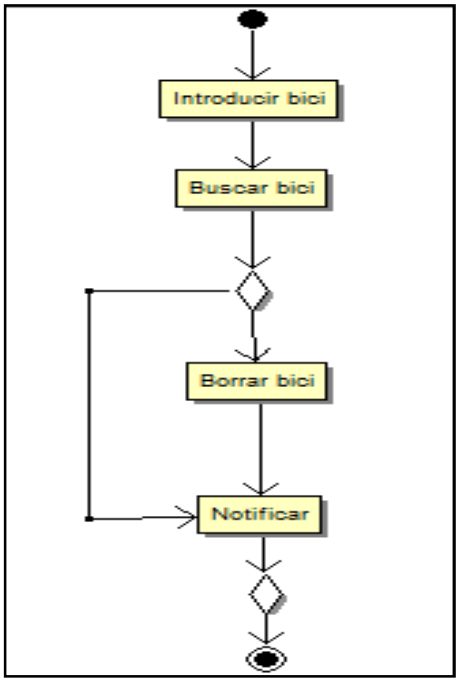


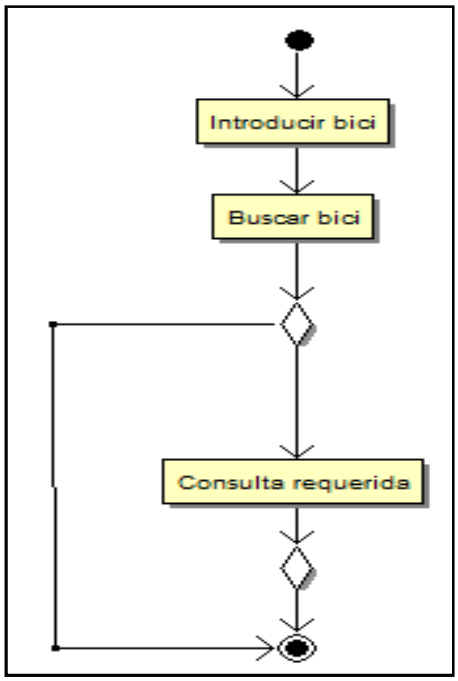


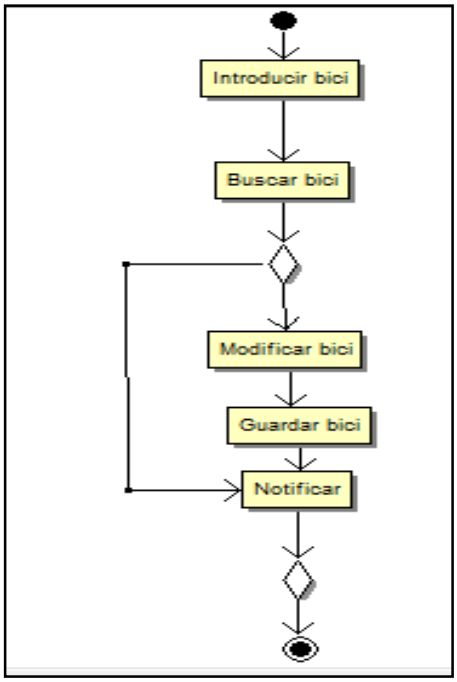
2.2.2. Actividades

Este paquete contiene las actividades que realiza cada caso de uso en un módulo. En este caso se ha realizado solo del modulo de Bicicleta como ejemplo, pues los demás módulos son similares.



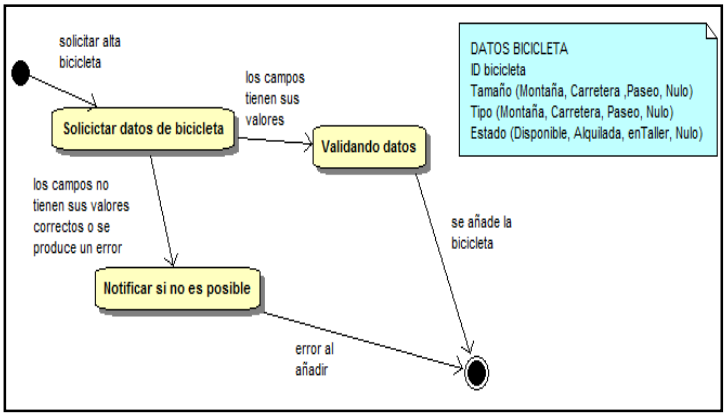


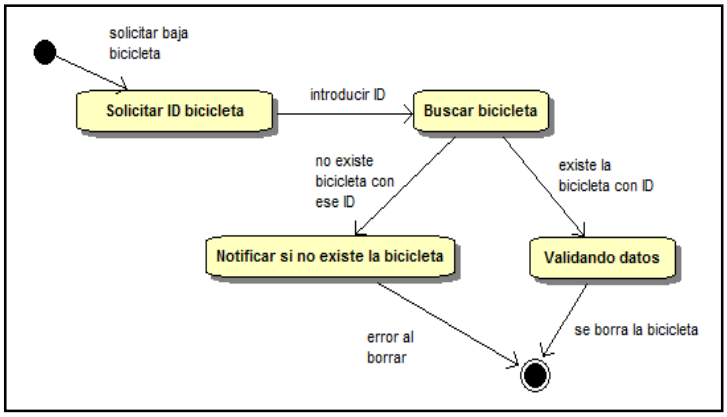


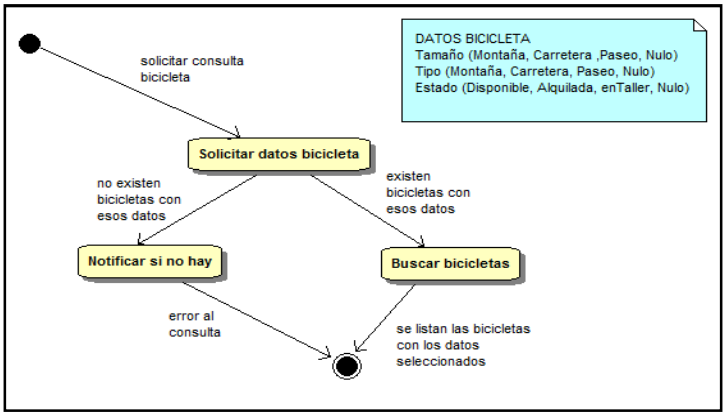


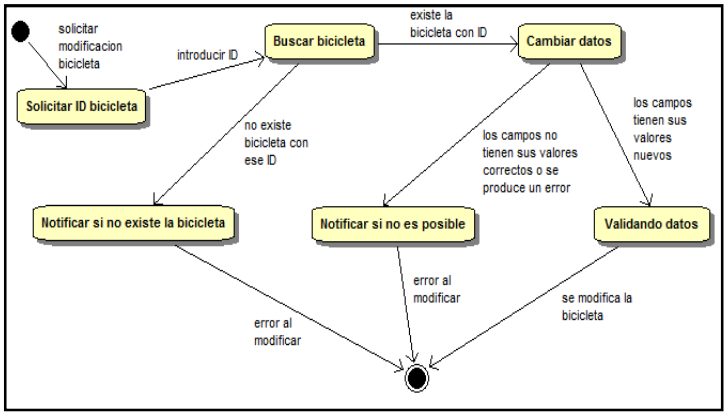
2.2.3. Transición estados

La transición de estados pretende describir los diferentes estados durante la realización de cada caso de uso. En este caso se ha realizado solo del módulo de Bicicleta como ejemplo, pues los demás módulos son similares.



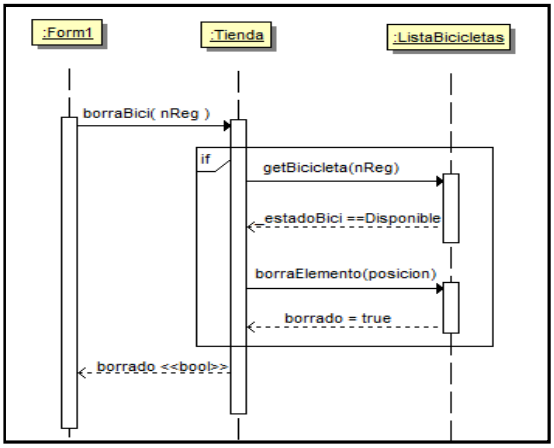


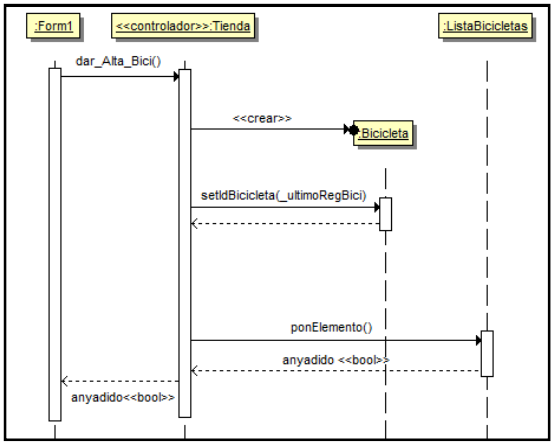


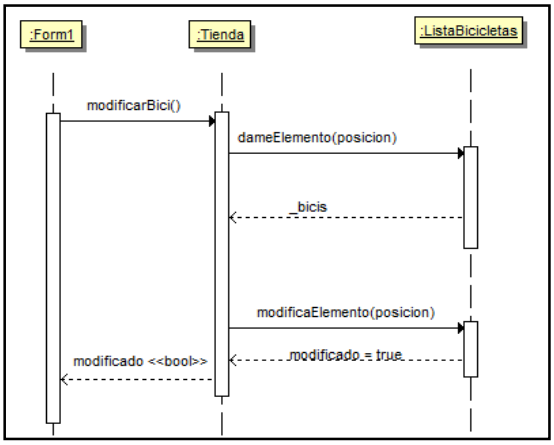


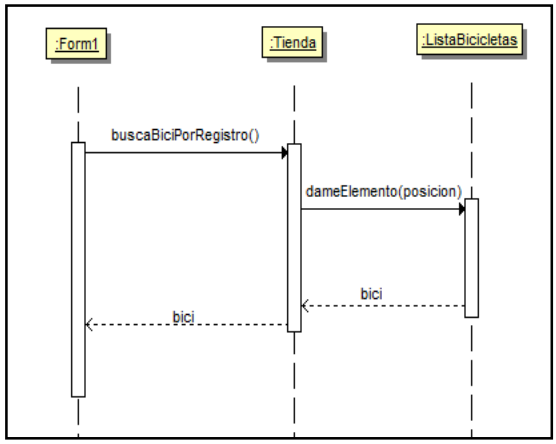
2.2.4. Secuencia

Los diagramas de secuencia muestran la secuencia de los datos durante la ejecución del cada caso de uso. El usuario elige la acción a realizar y la envía al controlador que se encarga de realizar las operaciones oportunas con los módulos y sus listas.

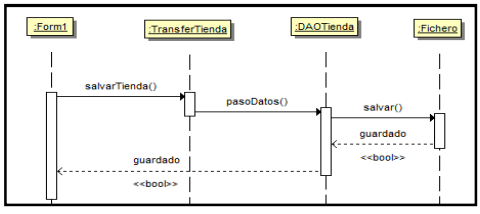


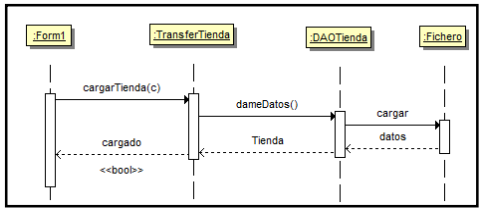






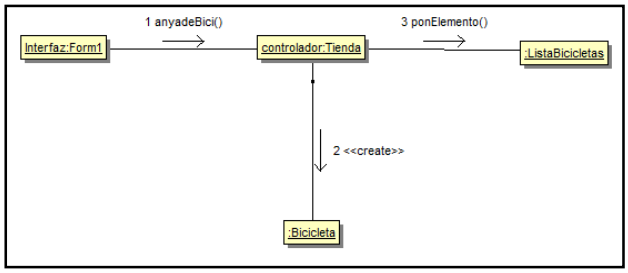
Cabe distinguir la parte encargada de cargar y salvar datos realizada solamente al iniciar y cerrar aplicación respectivamente. La función de cargar del fichero la realiza el DAO, pasando los datos al TRANSFER que crea el atributo "tienda" que al llegar al controlador tiene toda la información para gestionar los módulos. Para salvar se realiza el proceso contrario, el controlador le pasa el atributo "tienda" con toda la información al TRANSFER que se encarga de mandárselo al DAO para que realice la función de salvar al fichero.

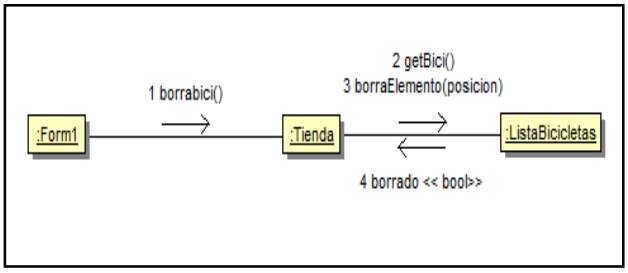


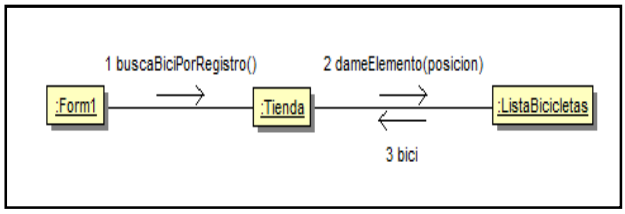


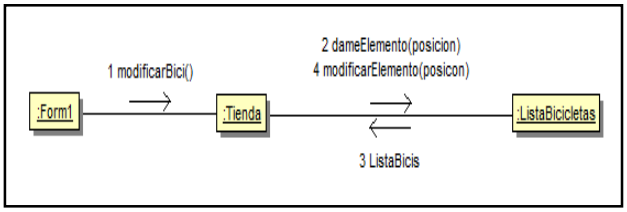
2.2.5. Comunicación

Los diagramas de comunicación detallan el paso y dirección de mensajes entre las clases durante la secuencia de ejecución de cada caso de uso.

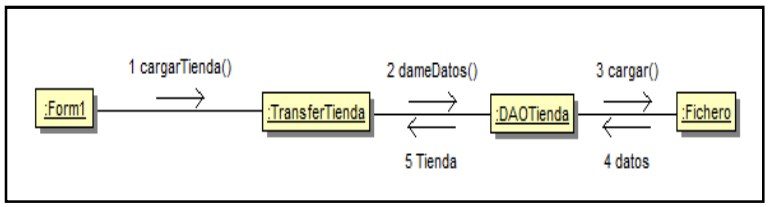


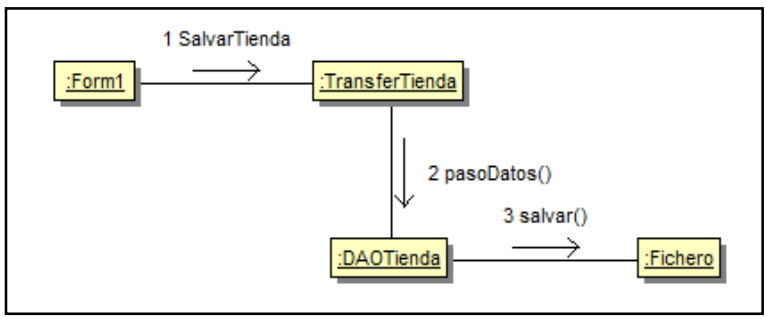






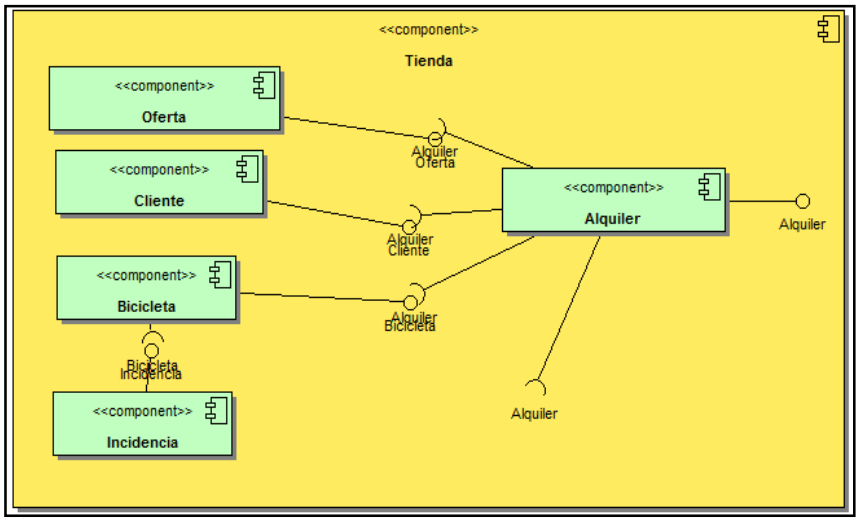
En este diagrama también cabe distinguir la parte encargada de cargar y salvar datos realizada solamente al iniciar y cerrar aplicación respectivamente.





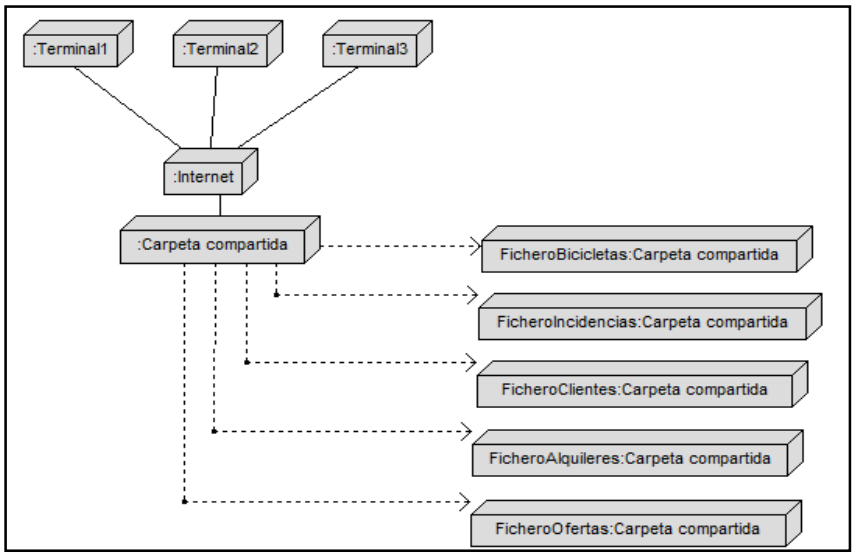
2.2.6. Componentes

El diagrama de componentes es un diagrama general que trata de enlazar los componentes durante el proceso general de la aplicación que es el alquiler de bicicletas a clientes. Se realiza un alquiler, gestionas al cliente, eliges la bicicleta deseada y comprueba que no tenga incidencia, y al realizar el pago se gestiona la oferta a aplicar si corresponde.



2.2.7. Despliegue

El diagrama de despliegue se centra en la estructura ya no tanto de los datos sino en general del sistema. Como se muestra, la aplicación se instala en los terminales de las tiendas, y mediante una carpeta compartida (siendo necesario así Internet) se accede a ella para localizar los ficheros necesarios y poder cargar y salvar los datos.



3. DESARROLLO APLICACION

En el desarrollo de la aplicación se utilizan diferentes patrones de diseño para una mejor gestión de los datos. Para este desarrollo se ha utilizado el lenguaje C++ con la herramienta "Visual Studio 2010", donde el formulario de usuario esta hecho en "Windows Forms" con CLR. A continuación describimos los diferentes patrones utilizados con una muestra de código de la implementación pertinente.

3.1. MVC (Modelo Vista Controlador)

El Controlador se encarga de mapear los eventos que llegan de los formularios con su correspondiente servicio de aplicación. Su función es controlar todas las operaciones entre los distintos módulos de gestión y realizar la cohesión para su correcto funcionamiento. En este caso el controlador se llama "Tienda", y los datos los recoge en el atributo "_tienda" para el cargado y salvado de datos. A continuación se muestra un trozo de código donde se observa que realiza varias operaciones.

```
class Tienda
{
public:
	Tienda();
	bool anyadeBici(const Bicicleta &nuevaBici);
	bool buscaBiciPorRegistro(int numRegistro, Bicicleta &bici)const;
	bool borraBici(int numRegistro);
 ……
```

3.2. TRANSFER

El TRANSFER se encarga de transferir los datos del controlador al DAO y viceversa. Para que los datos lleguen correctamente al DAO, el TRANSFER trata el atributo "_tienda" para que se carguen o salven correctamente. A continuación se muestra un trozo de código donde se observa que realiza el trato de "tienda".

```
void Transfer::cargar(Tienda &tienda,ifstream &f2)
{
		if(System::IO::File::Exists("BykeRent.bin"))
		{
			f2.read((char *)(&tienda), sizeof(Tienda));

		}
}
```

3.3. DAO (Data Access Object)

El DAO se encarga de interactuar con el sistema de almacenamiento, en el caso de "BIKERENT" el sistema de archivos es fichero, con lo cual lo que le pasa el TRANSFER se encarga de ejecutar las funciones de cargar y salvar. Sobre esta clase se realiza el Singleton que será explicado a continuación.

```
void Dao::cargar(Tienda &tienda)
{
		if(System::IO::File::Exists("BykeRent.bin"))
		{
			ifstream f2;
			f2.open("BykeRent.bin",ifstream::binary);
			Transfer::cargar(tienda,f2);
			//f2.read((char *)(&tienda), sizeof(Tienda));
			f2.close();
		}
}
```

3.4. SINGLETON

Aplicado al DAO, garantiza que una clase solo tenga una instancia, y proporciona un punto de acceso global a ella. En este caso se realiza acceso al DAO solo al iniciar el programa para cargar los datos y al cerrar para salvar. Se muestra la instancia a la clase para el cargado y salvado de información al fichero.

```
//cargar();
Dao::cargar(tienda);

//guardar();
 Dao::guardar(tienda);
```